PROCEEDINGS OF SCIENCE

Parallel Computing Workshop

Enol Fernandez-del-Castillo[*]
Instituto de Fisica de Cantabria (IFCA), CSIC-UC, Spain
E-mail: enolfc@ifca.unican.es

John Walsh[†]
Trinity College Dublin, Ireland
E-mail: John.Walsh@scss.tcd.ie

Alvaro Simon[‡]
CESGA, Spain
E-mail: asimon@cesga.es

The European Grid Infrastructure (EGI) offers a platform to execute parallel applications using a wide range of technologies and paradigms such as message passing with MPI, shared memory programming with OpenMP, or GPGPU programming.
The objective of this workshop is to present the latest advances in the support for MPI and parallel jobs on the European Grid Infrastructure. The following topics are covered: the MPI Virtual Team objectives and status; an update of the latest features and developments of MPI-Start; the use of MPI-Start for generic parallel jobs; and a summary of the current status of the support for GPGPU programming.

EGI Community Forum 2012 / EMI Second Technical Conference,
26-30 March, 2012
Munich, Germany

[*]Speaker.
[†]Speaker.
[‡]Speaker.

PoS(EGICF12-EMITC2)057

1. Introduction

Execution of parallel applications in clusters and HPC systems is commonplace. Grid infrastructures are mainly used for the execution of collections of sequential jobs although most Grid sites are clusters where execution of parallel applications is possible. The EGI Infrastructure [3] offers a platform to execute parallel applications using a wide range of technologies and paradigms such as message passing with MPI, shared memory programming with OpenMP, or GPGPU programming.

Previous workshops have covered the support for MPI in the infrastructure and middleware [20] and provided users with tutorials to execute their MPI jobs using the available tools [8]. In this workshop we will cover the following topics: Section 2 describes the work of the MPI Virtual Team of EGI-InSPIRE, created to improve the usage of MPI in the infrastructure. Next, Section 3 presents the status of the MPI-Start development and new features that may improve the users' experience, including the use of MPI-Start for hybrid OpenMP/MPI programming. The execution of non-MPI parallel jobs is considered in the following sections. On Section 4, we describe how MPI-Start may be used for the execution of generic parallel jobs, while on Section 5 an introduction to GPGPU programming is given. Last, some Conclusions are given on Section 6.

2. The MPI Virtual Team

The Virtual Team (VT) framework has been created within the NA2 activity of the EGI-InSPIRE project to improve the efficiency and flexibility of the interaction between the NGIs and EGI.eu User Communities. The original idea was to create short living projects that focus on well defined, non-operational activities around the production infrastructure. Different VTs cover different areas [4] (such as Marketing and Communication, Technical outreach, etc) and are open for NGIs, countries and regions that are not involved in NA2 or in EGI-InSPIRE. These entities can join existing Virtual Teams or propose new Virtual Teams.

One of these new created VTs is the MPI VT [17]. This new team has different objectives:

- Collaborate with user communities and projects that use MPI resources.
- Promote MPI deployment.
- Enhance MPI experience and support for EGI users.
- Produce materials (tutorials, white papers, etc) about successful use cases of MPI on EGI that can be used by new communities.

The new VT is a short living project (it has a life time of 6 months). It was started the past November 2011 and it will end at the end of May 2012. This work will provide an output at the end of the VT timeline:

- Improve the communications between MPI users and developers within EGI SA3.
- Improve/create new MPI documentation for EGI users and administrators.
- Find and detect current issues and bottlenecks related with MPI jobs.

- Set up a VO on EGI with site committed to support MPI jobs.

Due to the complexity of this project it was split in six tasks. These new tasks were assigned to different MPI VT members to be accomplished during the next months.

2.1 Task 1: Documentation

MPI documentation was fragmented in different places and in most cases only site admin oriented. MPI user communities are demanding new and improved user oriented documentation. To solve this documentation gap MPI VT has created an specific task to solve this issue. In the last months the current MPI EGI documentation was improved to include these new features. Currently the MPI VT group is working to include in the same endpoint a MPI user guide to help MPI user communities. The tasks undertaken are:

- Review and extension of documentation by MPI VT members.
- MPI Administrator guide: https://wiki.egi.eu/wiki/MAN03.
- Working on an MPI user guide.
- Propose to concentrate documentation under a single endpoint.

2.2 Task 2: Information System

The information system is a critical gear for any grid infrastructure. One of the most common failure reasons for MPI jobs is due to sites information system misconfiguration. To fix this issue and improve MPI sites reliability, a Information System task was created within the MPI VT. If this information is not published correctly MPI jobs may fail or queued indefinitely. This task has the following objectives:

- Detect if grid sites are publishing correctly MPI tags and flavours.
- Inspect which GLUE 1.3/2.0 variables could be used in the MPI context and ask for its correct implementation.

MPI VT members have detected a valid GLUE variable that can be used by MPI jobs: *MaxSlotsPerJobs*. This value reflects the maximum number of slots which could be allocated to a single job. Unfortunately this value is not filled by the current LRMS Information Providers and only an static ”999999999” value is published by default. MPI VT is in contact with the different Technology Provides to show this information by the information providers.

2.3 Task 3: MPI monitoring

Current EGI monitoring system does not detect if a MPI site is really working fine or not (only an MPI ”Hello word” is submitted). A new set of nagios probes specifications were created by MPI members to check if MPI sites are working correctly. The new specifications were reviewed by MPI members and it will be developed by SA3 team to be included into EGI nagios framework. The new probes will include these tests:

- Environment sanity check: To detect if a site has the *GlueCEPolicyMaxSlotsPerJob* variable set to a reasonable value and detect The information published by the (MPI or Parallel) service.
- Check the MPI functionality with a minimum set of resources (run MPI job using two slots in different machines).
- Complex Job check. This probe will requires 4 slots with 2 instances running in different dedicated machines.

The new nagios MPI probes will be able to detect any MPI issue or misconfiguration and will improve the MPI availability and reliability metrics.

2.4 Task 4: Accounting

EGI accounting system based on APEL is not ready yet to include parallel jobs usage records. This is an open task inside EGI project (with a specific task JRA1.4). The only way to recognise a parallel job in the accounting system is by checking jobs with efficiency > 1, although this does not guarantee a parallel job execution. APEL developers are working to provide new plugin to include parallel jobs accounting. To help in this task MPI VT members are in contact with EMI and APEL developers to provide a complete MPI accounting able to correctly record parallel usage at the end of 2012.

2.5 Task 5: LRMS status

Another important task for MPI Virtual Team members is to detect new potential failures in different batch systems. In the last months several issues that affect directly to the parallel job execution were detected .

One of these bugs was detected in MAUI, typically used in conjunction with the Torque scheduler. Due to this issue, Torque was not able to submit jobs that require more than a single node. VT members have contacted directly to EMI maintainers and a new fix will be available in the next EMI release. An important point is that some LRMS are supported by third parties, are not maintained directly by the technology providers like EMI. This issue was raised by MPI VT to take into account of the different LRMS releases and updates.

2.6 Task 6: MPI user communities and VO

VTs were created to improve the interaction between the NGIs and EGI.eu. The MPI VT is regularly in contact with different NGIs to gather information about different MPI use cases and site administrator feedback. In the first months of life of the VT several reports and surveys were submitted to the NGIs. These reports are discussed and reviewed by the MPI members in order to try to help user communities and site administrators. In order to help MPI site admin to configure parallel environments in their sites it was created a new MPI VO within VT. The new VO will bring together sites and users interested in MPI but it is not a permanent VO. It was created for testing purposes to collect experience that will be later adopted by regular VOs. MPI VO configuration is available in the MPI VT wiki page and could be used by any site to support it and test MPI functionalities.

3. MPI-Start

In order to provide a uniform interface for running MPI applications in grid environments, MPI-Start project was started in the frame of int.eu.grid [16]. Currently its development continues in the framework of the EMI project [6]. MPI-Start is a unique layer that hides the details of the resources and application frameworks (such as a MPI implementation) to the user and upper layers of the middleware. The use of such layer removes any dependencies in the middleware related to starting parallel applications.

MPI-Start is composed by a set of scripts that ease the execution of MPI programs by using a unique and stable interface to the middleware. The scripts are written in a modular way: there is a core and around it there are different plug-ins. Three main frameworks are defined in the MPI-Start architecture: *Scheduler*, that deals with the local batch systems; *Execution*, that manages the special parameters needed to start each MPI implementations; and *Hooks*, that provide additional features and is customisable by both sites and final users. A detailed description of the architecture and main features of MPI-Start is available at [2]

The latest released version of MPI-Start in EMI is 1.2.0, that was made available as part of EMI-1 update 11 [1] on 12th of December 2011. This release is an enhancement of the MPI-Start 1.x series that are included in the first EMI-1 release and subsequent updates. The main features of this release are:

- Support for MPI implementations: Open MPI [9], MPICH [12] (including MPICH-G2 [13]), MPICH2 [11], and LAM-MPI [19].
- Support for Local Resource Management Systems: SGE [10], PBS/Torque [1], LSF [21] and Condor [15].
- Support for controlling the placement of processes on the allocated machines, including setting processor and memory affinity (for Open MPI and MPICH2). This allows the execution of Hybrid MPI/OpenMP applications as described in the next section.
- Detection of the appropriate compilers for the MPI implementation in use.
- Use of command line parameters instead of environment variables for defining the MPI-Start behaviour.
- Refactored configuration that allows sites and users to provide their own plugins for the scheduler or execution framework.

3.1 Hybrid MPI/OpenMP Applications

Parallel applications using the shared memory paradigm are becoming more popular with the advent of multi-core architectures. MPI-Start default behaviour is to start a process for each of the slots allocated for an execution. However, this is not suitable for applications using a hybrid architecture where several threads access to a common shared memory area in each of the nodes.

[1] `http://www.eu-emi.eu/emi-1-kebnekaise-updates/-/asset_publisher/Ir6q/content/update-11-15-12-2011`

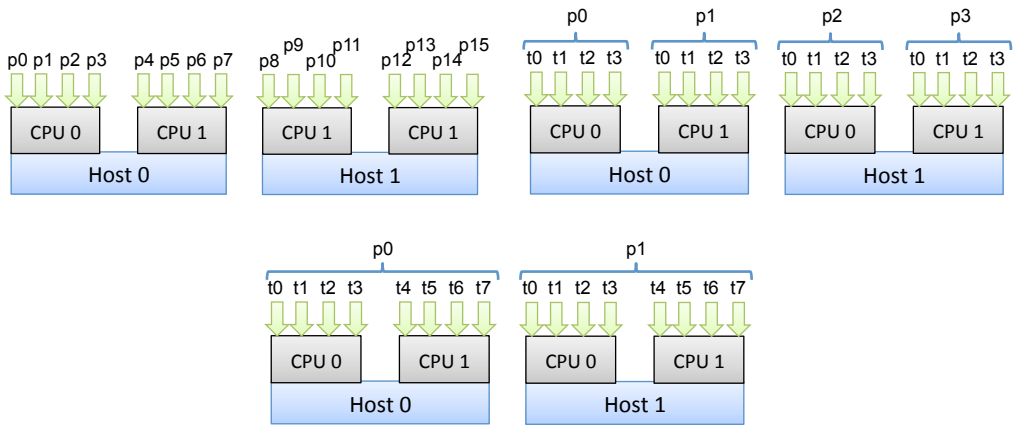

Figure 1: MPI-Start mapping of processes. (a) Top left: per core; (b) Top right: per socket; (c) Bottom: per node.

In order to support more use cases, the latest report of MPI-Start includes support for better control of how the processes are started, allowing the following behaviours:

- Define the total number of processes to be started, independently of the number of allocated slots.
- Start a single process per host. This is the usual use case for hybrid jobs with and MPI applications. MPI-Start prepares the environment to start as many threads as slots available in the host.
- Start a single process per CPU socket. In this case, an hybrid application would start as many threads as cores are available for each CPU.
- Start a process per CPU core, independently of the number of allocated slots.
- Define the number of processes to be started in each host, independently of the number of allocated slots at each host.

All these modes must be used with caution, since they could interfere with other jobs running in the machines. It is recommended that node exclusivity is enforced on the job submission (e.g. using the `WholeNodes` attribute of JDL).

Figure 1 shows the different possible mappings for two hosts with two quad-core CPUs. In the per core case, there would be 16 MPI processes, numbered from $p0$ to $p15$, each assigned to one CPU core. In the case of using a per socket mapping, each host would have two different processes – $p0$ and $p1$ in the first host, $p2$ and $p3$ in the second host – and for each of these processes, 4 different threads ($t0$ to $t3$). Finally, if the per node mapping is used, only one processes would be started at each host – $p0$ in first host, $p1$ in the second one — and 8 threads would be started for each of them, numbered from $t0$ to $t7$.

The jobs depicted in Fig. 1 can be submitted with a JDL that specifies the `NodeNumber`, `SMPGranularity` and `WholeNodes` attributes, as shown below:

```
Executable = "/usr/bin/mpi-start";
Arguments = "-t openmpi user_app arg1 arg2";
NodeNumber = 2;
SMPGranularity = 8;
WholeNodes = True;
InputSandbox = {"user_app", ...};
```

The JDL requires 2 different hosts, each with 8 cores and used exclusively by the job. With the arguments shown, mpi-start would execute the `user_app` with Open MPI using the process distribution of case (a) in the Figure, without any special treatment of the allocated resources. Case (b) would be achieved by adding the `-psocket` option to the `Arguments` attribute:

```
Arguments = "-t openmpi -psocket user_app arg1 arg2";
```

The behaviour of case (c) is obtained by using the `-pnode` option in the `Arguments` attribute:

```
Arguments = "-t openmpi -pnode user_app arg1 arg2";
```

In this case, using the `-pcore` option would be equivalent to not using it since there is a slot allocated per each of the available cores as defined in the JDL.

MPI-Start also provides a set of variables accessible from the user application that describe the allocated resources by the batch system as well as the placement of processes that is used for the execution.

3.2 Integration with ARC and UNICORE: towards EMI-ES

MPI-Start was originally developed for the integration with the gLite [14] middleware, although the design and architecture of MPI-Start is completely independent of this middleware. With the latest releases, MPI-Start can be integrated with the ARC and UNICORE stacks using the native mechanisms of each platform.

In the case of ARC [5], a new Runtime Environment that invokes MPI-Start is available. In the case of UNICORE [7] a Execution Environments is defined using a XML file where the different options available to the users are described. In order to use MPI-Start in such way, we introduced the possibility of setting the parameters via command line arguments instead of environment variables.

In order to provide a unified interface for the different middleware stacks in EMI, the EMI Execution Service (EMI-ES) [18] was developed. This specification includes a *Parallel Environment* section that will provide a simple and user-friendly way of setting up parallel jobs. Each parallel environment is identified by a type (e.g. generic MPI, Open MPI, MPICH2) and a version. Additional option tags allow users to specify non default behaviour and resource specification field allows setting the number of processes per slot or threads per processes to be used. Once the EMI-ES implementations are available, MPI-Start will be adapted to act as back-end of the Parallel Environments, thus all the MPI-Start features will be easily accessible from the new interface.

4. Using MPI-Start for Generic Parallel Jobs

The MPI frameworks (OpenMPI, MPICH) are the most successful methods for managing parallel workloads distributed across a local network. However, numerous legacy applications

using the parallel virtual machine (PVM) framework are still in general production use, and more importantly, several newer paradigms for handling distributed workloads have also emerged. These include Map/Reduce and Charm++.

The gLite JDL *NodeNumber* variable allows a user's job to request multiple job slots on a Computing Element. This facilitates allocation of worker-node resources, but not execution of the workload across those resources. The MPI-Start framework, however, was developed and has been successfully used for many years on the EGI infrastructure for exactly this purpose.

The authors investigated whether MPI-Start could be used as a way to handle non-MPI parallel workloads. The experiments were carried on an MPI enabled site with shared filesystem (i.e. publishing *MPI_SHARED_HOME*). However, the methodology can be extended to accommodate other resource centre setup scenarios.

4.1 From mpiexec hacks to building blocks

The OSC mpiexec FAQ[2] describes how mpiexec can be used to execute remote processes and to copy files between nodes. These recipes can be used as a primitive building blocks to develop more complicated applications. Moreover, all mpiexec implementations may be used to execute non-mpi executables. We recommend, however, that users should continue to use the MPI-Start framework as this provides environment setup, file-copying, and explicitly launches mpiexec with the correct parameters.

4.2 Master/Slave Applications

Both OpenMPI and MPICH2 define a number of environment variables that are available to every MPI process. In particular, they export variables which relate to the number of process slots allocated to the job and to the *MPI Rank* of the processes. Using this information one can nominate a "master" or coordinator process in the set of processes. This allows us to accommodate master/slave use-cases. Table 1 lists a selection of MPI related environment variables.

MPI distribution	**Rank Identifier**	**Comm Size**
OpenMPI	OMPI_COMM_WORLD_RANK	OMPI_COMM_WORLD_SIZE
MPICH2	MPIRUN_RANK	MPIRUN_NPROCS

Table 1: Example MPI related environment variables

A simple indication of how this can be used in practice can be found in the listing below. Further information and full examples can be found on the EGI Parallel Computing Support User Guide web page[3].

```
#!/bin/bash
if test x"$OMPI_COMM_WORLD_RANK" = x"0" ; then
 ### Code for coordinating master process
else
 ### Code for slave processes
fi
```

[2] http://www.osc.edu/~djohnson/mpiexec/index.php#Cute_mpiexec_hacks

[3] https://wiki.egi.eu/wiki/Parallel_Computing_Support_User_Guide

4.2.1 A Charm++ master/slave application

Charm++ is an object based parallel programming system. It's execution model is message-driven, with Charm++ computations triggered based on the reception of associated messages. Moreover, it has an adaptive runtime system, and supports *message-driven migrateable object*. To install Charm++ it must first be compiled from source. The authors were able to package Charm++ 6.21 into a compatible Redhat Package Management (RPM) format, and then deployed it at the Trinity College Dublin grid resource centre.

A simple example using the 3darray example code from the Charm++ distribution can be found below. Note in particular that we must convert the PBS machinefile into a Charm++ compatible format.

```
#!/bin/bash

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/charm/lib
export PATH=$PATH:/opt/charm/bin
if test x"$OMPI_COMM_WORLD_RANK" = x"0" ; then
echo "Now building Charm++ 3darray example"
make -f Makefile
cat ${MPI_START_MACHINEFILE} | sed  's/^/host /g' > charm_nodelist
charmrun ++remote-shell ssh ++nodelist charm_nodelist \
 ./hello +p${OMPI_UNIVERSE_SIZE} 32 +x2 +y2 +z2 +isomalloc_sync
fi
# The slave node does not actively do anything, it's processes are
# launched by the Charm++ "remote-shell" invocation.
```

4.2.2 Map/Reduce - Hadoop on Demand

It is not hard to see how the "coordinator" model can be used in a much more general sense. The authors are currently investigating how to use this method to implement Hadoop-on-Demand [22], a system for provisioning virtual Hadoop clusters using existing batch systems, in the grid environment.

5. GPGPU programming

The increasing capabilities of general purpose graphics processing units (GPGPUs) over the past few years has resulted in a huge increase in their exploitation by all the major scientific disciplines. With three of the top five clusters in the current November 2011 Top500[4] supercomputers list using NVIDIA GPGPUs, we would expect the number of GPGPU deployments at grid resource centres to grow significantly over the next few years. However, there are two major problems in supporting grid access to such resources. Firstly, there is currently no standardised way for resource centres to advertise/publish availability of these resources. Secondly, there are deficiencies in current batch scheduling systems which make it difficult to guarantee exclusive access to those resources.

[4] http://www.top500.org/lists/2011/11

GPGPU Application development is dominated by two API frameworks: OpenCL[23] and CUDA[24]. OpenCL supports heterogeneous compute environments such as AMD and Intel CPUs, AMD, Intel and NVIDIA GPGPUs, and Cell Broadband Engine processors. CUDA applications currently run on NVIDIA GPGPUs devices only. However, the GNU Ocelot project[25] is attempting to enable CUDA application development for non-NVIDIA processors. CUDA applications offer a performance advantage over OpenCL[26].

GPGPUs are highly efficient for "single instruction, multiple data" (SIMD) models of execution. The authors are able to report successfully running a hybrid MPI/OpenCL application on the grid infrastructure using 16 individual machines and 30 NVIDIA GPGPUs[5]. MPI was used for inter-node communication, and OpenCL was responsible for device allocation and workload distribution to Intel CPUs and NVIDIA GPGPU devices.

5.1 GPGPU Grid Integration Challenges

Although it is fairly straightforward to assemble grid-enabled worker nodes with GPGPUs, integrating them as first class resources, such as CPU and storage resources, into the grid infrastructure is a much more challenging problem. The fundamental issues can be classified as follows:

- Some batch system integration support,
- Inadequate Operating System hardware device level security,
- Lack of a standardised grid glue-schema support for GPGPUs, and
- Lack of information system plugins.

None of these issue are huge technical hurdles, and in particular, most recent versions batch systems such as LSF, Torque, SGE and Slurm have some level of support. The only batch scheduler which does have issues is Maui, but some experimental patches are available for this. Glue-schema support requires standardisation and agreement within the grid community.

6. Conclusions

The execution of parallel applications in grid environments is a challenging problem that requires the cooperation of several middleware tools and services. Although the support from middleware and the infrastructure is constantly improving, users still need guidance and support from experts that help them with they day to day issues. Both the SA3 task, devoted to provide support for MPI jobs, and the MPI Virtual Team recently created, help users to get their parallel jobs running in the available resources.

The latest developments in MPI-Start, a tool that provides a unique layer for several MPI implementations, have introduced better control of job execution, the possibility of running hybrid MPI/Open MP applications and the integration with ARC, gLite and UNICORE middleware stacks. Users are totally abstracted by using the unique interface of MPI-Start and can easily migrate their application from one middleware provider to another.

[5]Each node had 2 GPGPUs, but the master node did not use any of its GPGPUs

The execution of parallel jobs is not limited to MPI. Other generic parallel jobs can be executed with the help of MPI-Start and MPI tools as shown with Charm++ or Hadoop. An introduction to the use of GPGPUs programming and possible integration methods in the grid infrastructure was also covered in the workshop.

Acknowledgments

The authors acknowledge support of the European Commission FP7 program, under contract number 261323 through the project EGI-InSPIRE (`http://www.egi.eu/`) and under contract number 261611 through the project EMI (`http://www.eu-emi.eu/`)

References

[1] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, *Portable batch system: External reference specification*, Technical report, MRJ Technology Solutions, 1999.

[2] K. Dichev, S. Stork, R. Keller, E. Fernandez, *MPI Support on the Grid*, Computing and Informatics, volume 27, pp. 213–222, 2008.

[3] European Grid Initiative (EGI) `http://www.egi.eu/`

[4] EGI Virtual Teams
`https://wiki.egi.eu/wiki/Overview_of_Virtual_Team_projects`

[5] M. Ellert et al., *Advanced Resource Connector middleware for lightweight computational Grids*, Future Generation Computer Systems, volume 23, pp. 219–240, 2007.

[6] European Middleware Initiative (EMI) `https://twiki.cern.ch/twiki/bin/view/EMI`

[7] D. Erwin, *UNICORE - A Grid Computing Environment*, Lecture Notes in Computer Science, volume 2150, pp. 825–834, 2001.

[8] E. Fernandez, *MPI Hands on Training*, EGI User Forum, April 2011.

[9] E. Gabriel et al, *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, Lecture Notes in Computer Science, 3241, pp. 97–104, 2004.

[10] W. Gentzsch, *Sun Grid Engine: towards creating a compute power grid*, Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 35–36, 2001.

[11] W. Gropp, *MPICH2: A new start for MPI implementations*, Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 7, 2002.

[12] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing, volume 22(6), pp. 789–828, 1996.

[13] N. T. Karonis et al, *MPICH-G2: A grid-enabled implementation of the message passing interface*, J. Parallel Distrib. Comput., volume 63(5), pp. 551–563, 2003.

[14] E. Laure et al., *Programming the Grid using gLite*, EGEE-PUB-2006-029, 2006.

[15] M. Litzkow, M. Livny, and M. Mutka, *Condor - a hunter of idle workstations*, Proceedings of the 8th International Conference of Distributed Computing Systems, 1988.

[16] J. Marco et al, *The Interactive European Grid: Project Objectives and Achievements*, Computing and Informatics, volume 27, pp. 161–173, 2008.

[17] MPI Virtual Team. `https://wiki.egi.eu/wiki/VT_MPI_within_EGI`

[18] B. Schuller et al., *EMI Execution Service Specification*, v1.07, 2011.

[19] J. M. Squyres, *A component architecture for LAM/MPI*, Proceedings of the 9th ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 379–387, 2003.

[20] J. Walsh et al., *Message Passing Interface - Current Status and Future Developments*, EGI Technical Forum, 2010.

[21] S. Zhou, *Lsf: Load sharing in large-scale heterogeneous distributed systems*, Proccedings of the Workshop on Cluster Computing, 2002.

[22] Hadoop On Demand *http://hadoop.apache.org/common/docs/r0.17.0/hod.html*

[23] M. Scarpino, *OpenCL in Action: How to accelerate graphics and computation* Manning Publications Co., ISBN 9781617290176, 2011

[24] D. Kirk *Programming Massively Parallel Processors: A Hands-on Approach* Morgan Kaufmann Publishers, ISBN 9780123814722, 2010

[25] GNU Ocelot Homepage *http://code.google.com/p/gpuocelot/*

[26] J. Fang, A.L. Varbanescu and H. Sips, *A Comprehensive Performance Comparison of CUDA and OpenCL*, The 40-th International Conference on Parallel Processing (ICPP'11), Taipei, Taiwan.