THOR: A Transparent Heterogeneous Open Resource Framework

J.L. Vázquez-Poletti
Dpto. de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
28040 Madrid (Spain)
Email: jlvazquez@fdi.ucm.es

J. Perhac, J. Ryan and A.C. Elster
Dept. of Computer and Information Science
Norwegian University of Science and Technology
NO-7491 Trondheim (Norway)
Email: {perhac,ryanjp,elster}@idi.ntnu.no

***Abstract*—Heterogeneous computing which includes mixed architectures with multi-core CPUs as well as hardware accelerators such as GPU hardware, is needed to satisfy future computational needs and energy requirements. Cloud computing currently offers users whose computational needs vary greatly over time, a cost-effect way to gain access to resources. While the current form of cloud-based systems is suitable for many scenarios, their evolution into a truly heterogeneous computational environments is still not fully developed.**

This paper describes THOR (Transparent Heterogeneous Open Resources), our framework for providing seamless access to HPC systems composed of heterogeneous resources. Our work focuses on the core module, in particular the policy engine. To validate our approach, THOR has been implemented on a scaled-down heterogeneous cluster within a cloud-based computational environment. Our testing includes an Open CL encryption/decryption algorithm that was tested for several use cases. The corresponding computational benchmarks are provided to validate our approach and gain valuable knowledge for the policy database.

I. Introduction

In recent years, high-performance computing (HPC) has changed dramatically and in many different ways. The petascale computational level has been reached, and exceeded, successfully mostly thanks to the distinctive combination of heterogeneous architectures, including multi-core CPUs (Central Processing Units) and accelerators, which are in many cases based on GPU (Graphics Processing Unit) processors and related GPGPU (General-Purpose computing on GPU processors) technology. Nevertheless, problems related to the overall power consumption and associated heat dissipation issues cannot be solved without seriously impacting peak performance; in fact they are becoming even more visible and more demanding. Effectively, maximising power efficiency is becoming the most important factors for reaching exascale levels of computation. While the power consuption of high-end graphical accelerators is two times higher than the power consumption of high-end server-based CPU processors, the computational power of these accelerators is compared to CPU processors higher by factor of 10 to 100, if conditions related with programming model of these devices are fulfilled. This means that these accelerators are the ideal computational source for power-efficient HPC computing. Thus, it can be speculated that mobile device-like architectures and corresponding technologies, that were designed with power efficiency as a key consideration, will play an important role in future generations of heterogeneous HPC systems. It can be also assumed that the same trends hold for virtualization and next generation of cloud systems, so the focus and business model of commercial cloud providers will also change as these systems are built as they will be increasingly leased by HPC customers. As HPC computation will become more of a utility, new frameworks will be required in order to deal with this new environment.

One of the biggest HPC related issues that can be found in research organizations is the difficulties for users in accessing the required computational resources to solve their problems. As is the case with computational sciences, the volume and resolution of applications is constantly increasing and the capacity of an organization's resources to solve these problems quickly becomes insufficient. Typically, capacity is increased with the purchase of additional physical resources which tend to be more heterogeneous, scattered and less accessible by users.

OpenCL [1] is an open, royalty-free standard for cross-and-multi-platform programming of modern processors found in personal computers, servers and also embedded devices [2]. With the possibility to address and to program many different types of processor, OpenCL is a good candidate to be used within a future heterogeneous cloud computing environment [3]. However, in order to obtain a proper utilization of multiple different resources within a cloud, a pluralistic task assigning mechanism is required to be developed.

In this paper, we are addressing the necessity of unified access to HPC systems composed of heterogeneous resources, specially those based on GPU hardware. A framework is proposed in order to provide a solution, being its services described in Section II and its architecture in Section III. In order to illustrate its features, some general use cases are explained in Section IV. Section V analyzes the execution of a specific OpenCL application using different resources and the expected behavior of the framework. This paper then ends with some concluding remarks and pointers to future work.

978-1-4244-8396-9/10/$26.00 ©2010 IEEE.

II. Services provided

Typical computational problems imply execution of a certain code or using interactively a machine with specific attributes. In many situations, users simply do not have the knowledge or time to understand how an organization's resources are configured, so their harnessing is often not done in the most optimal fashion. Yet it can also transpire that having correctly identified the necessary resources, they are then not available (nor equivalent ones if specified) to meet the deadline for the users work.

For this reason we have taken the principle of cloud computing into account for the design of THOR, which stands for Transparent heterogeneous Open Resources, a platform providing flexible and on-demand services. In this way, a user will simply specify application requirements, any deadline for completion and if applicable, maximum (real or notional) cost and then let an abstraction level resolve the scheduling and execution of the job.

As it can be deduced, there are two main services offered by the THOR framework:

1) Access to a collection of pre-existing codes from a wide variety of research domains. Each code has its minimum requirements (such as a particular variety of GPU), but a user may also request execution on a resource with superior specifications if available. Once the machine is located, input data supplied by the user will be uploaded for code execution. When output data is available, the user is presented with a handle to download it.
2) Interactive access to a machine with required characteristics, the user is provided with a handle to login credentials.

With both services it may happen that sufficient resources from the users organization may not be available, so the framework will be able to provide an outsourced solution to an external computational grid or commercial cloud provider.

III. Framework architecture

The THOR framework is composed of the modules depicted in Figure 1. This modularized structure allows the framework to be technology independent, so integration with new developments is easily facilitated.

The user interacts with the system using either a command line interface or through an API, and on top of which more user-friendly interfaces such as web portals can be built [4] [5]. As explained in the previous section, two types of services are offered, and both the User Interface and the API provide mechanisms for requesting and interacting with them.

The service offering pre-existing applications functionality is handled by a module called TrustedApps and is contacted by the framework core. This module is a repository for the codes and their associated architectural requirements such as type or number of cores. Only jobs utilising these trusted codes can be scheduled to run on scavenged resources (such as BOINC) available to the framework. The framework core is responsible for coordinating all the actions within the framework and bases its decisions on a *policies* database and the constant monitoring of all available resources. The policies refer to both resource provisioning and job scheduling, and examples of these policies can include the ability to specify equivalent resources if those requested are not available, or preferring the use of certain resources in order to meet pre-defined environmental, power efficiency and economic requirements [6] [7] [8], and where appropriate to redirect computational tasks to an external infrastructure [9]. Interactions with the infrastructure are performed by the core via the MachineNow and OverFlow modules.

A. MachineNow module

The MachineNow module is responsible of harnessing the local infrastructure. This infrastructure traditionally is composed of heterogeneous resources which may respond to different computing paradigms such as cluster and collaborative computing. For example, a typical infrastructure may consist in a dedicated GPU cluster with a batch scheduler such as Sun Grid Engine [10] and a number of desktop PCs with BOINC clients [11].

In this context, the MachineNow module becomes an abstraction layer allowing the core to interact transparently with these local resources. To accomplish this, the module relies on plugins which translate core instructions to implementation specific commands. This approach has proven to be decisive in computing paradigms such as grid for coordinated harnessing of incompatible infrastructures [12].

For this reason, and even if at this moment virtualization techniques do not yet allow an optimal interaction between CPU and GPU preventing with the use of virtual infrastructure managers [13] [14], the MachineNow module is prepared for when this does becomes available [15]. In the meanwhile, work has been done on establishing virtualized cloud infrastructures without virtualization [16].

B. OverFlow module

There may be situations where local resources cannot meet requirements coming direct from the user or the TrustedApps module. The only option was to wait until required resources become available, but now an alternative option is to outsource the computational power needed and this is provided by the OverFlow module.

This module is a special implementation of MachineNow, where public clouds such as Amazon's AWS[1] can be contacted for outsourcing computational needs when local resources are not sufficient, or where core policies dictate otherwise. Again, different providers can be utilized based on a plugin architecture similar to MachineNow.

IV. General use case types

In this section some typical use cases of the framework are examined. Applications can be submitted to a batch system or require the resources to be interactive, in the latter case

[1] http://aws.amazon.com/

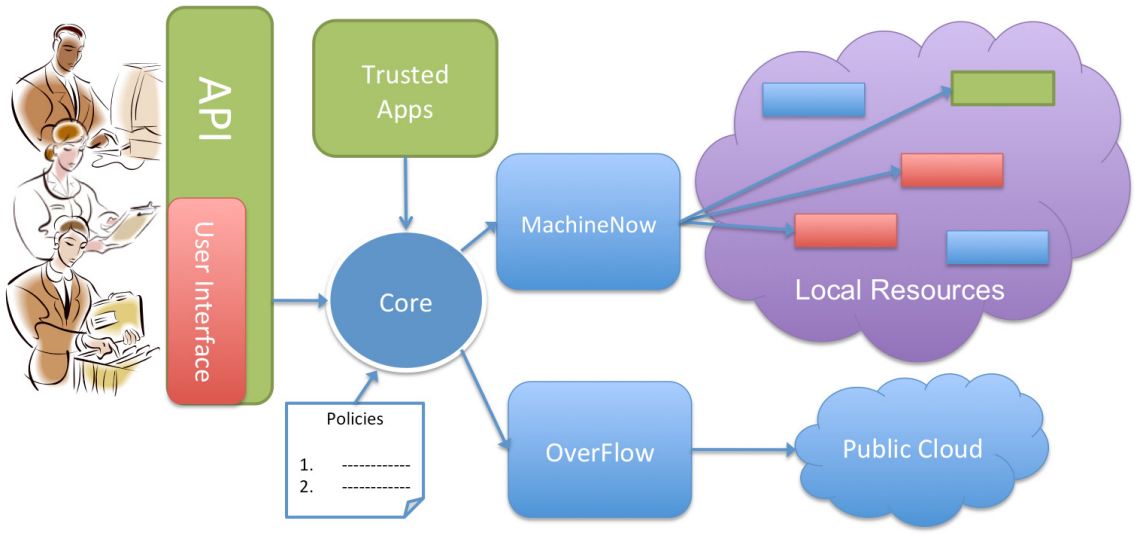

Fig. 1. The THOR framework architecture

immediate provisioning of the resources is required, but in the former this may not be such a constraint.

A. Interactive command line

In this example a user wishes to have interactive access to the resource, it will be the job of the framework to provide the user with a login to a resource with the requested capability. THOR will provide a command line instance either on a dedicated cluster as provided by the MachineNow module or a VM from an external cloud provider via the OverFlow module. The OverFlow module will be available where it is implicitly assumed that the user is not willing to suffer considerable delay in getting hold of the resources. If a dedicated cluster is unable to provide access within an acceptable period – an accurate start-time can usually be obtained from cluster management systems such as *SGE* and *torque* – then a VM can usually be provisioned from a cloud provider such as *AWS* within seconds by the OverFlow module.

B. Scientific Application

The second application usecase of the THOR Framework examines a situation where a user wishes to execute an application with significant input data requirements, one such application domain is the processing of seismic data typically found in oil exploration. A typical input dataset of seismic data can be many gigabytes in size, so will be impractical in some cases to move the data to the location of the computational resource, or may be impossible to a resource with insignificant resources to perform the task i.e. an embedded device with constrained local storage.

C. Bulk Submission of Applications

Many types of scientist deal with computationally intensive applications requiring numerous submissions with different input datasets, termed parameter sweeps, where all instances are independent and can be executed in parallel. One common application used in materials research is the *LAMMPS* molecular dynamics (MD) application [17]. This application is available in serial and parallel forms, and in MPI and GPU versions. As this application is non-interactive it can be scheduled on any computational resource if the requirements are met i.e. if parallel version then on the dedicated cluster, if the GPU version then execution can take place on the scavenging system, dedicated cluster, and in the case of a serial non-accelerated version, scheduling can occur on any resource providing it meets deadline and other scheduling requirements.

D. GPU on demand

In the final use-case we imagine a scenario where a user is executing an application on a constrained device and requires the compute resources of one or more accerators/GPUs. Via the THOR Framework these resources can be made available transparently to the user using a mechanism such as rCUDA [18]. rCUDA can be used to provide the user with a virtual GPU in order to run CUDA computational kernels where the user may be using a constrained device such as a mobile device/tablet wgere the resources are reserved for the local rendering. Implicit requirements of the application would be a resource providing a suitable GPU.

A similar use may arise when a user wishes to execute a workflow where only a small percentage of the overall computational requirements of the job require access to GPU resources, and proving inefficient to schedule the whole workflow to occupy the resources. Here the GPU resources could be potentially shared among a number of concurrent applications.

V. Demonstration through a specific Use Case

In order to better illustrate the frameworks behavior, a specific application written using the OpenCL standard is executed in a set of heterogeneous resources, pertaining to different computing paradigms. The idea is to show which

TABLE I
LOCAL INFRASTRUCTURE COMPONENTS SPECIFICATION

Component	GPU Type	CUDA Cores	Memory	Bus Width
TESLA S1050	GT200	240	4096MB	512bit
GTX 470	GF100	448	1280MB	320bit
9800 GTX	G92	112	512MB	256bit

TABLE II
CHARACTERISTICS OF THE DIFFERENT AMAZON EC2 MACHINE TYPES

Machine Type	Cores	C.U.	Memory	Platform	Cost ($/hr)
Standard On-Demand Instances					
Small (Default)	1	1	1.7GB	32bit	0.085
Large	2	2	7.5GB	64bit	0.34
Extra Large	4	2	15GB	64bit	0.68
High CPU On-Demand Instances					
Medium	2	2.5	1.7GB	32bit	0.17
Extra Large	8	2.5	7GB	64bit	0.68

provisioning and scheduling decisions THOR would take according to performance and throughput metrics.

Let AESEncryptDecrypt, which comes with the OpenCL implementation from ATI[2], be registered in the TrustedApps module. AESEncryptDecrypt accepts an $512x512$ pixel image as input and produces an encrypted one using the AES encryption algorithm after a given number of steps.

A. Infrastructure

As the proof of concept for the THOR framework a reduced local infrastructure consisting of two types of resources was tested: a dedicated cluster of NVIDIA Tesla S1070 nodes with Sun Grid Engine scheduler, and a group of desktop computers with different generations of graphics accelerators, namely NVIDIA GeForce 9800 GTX and Fermi based NVIDIA GeForce GTX 470 cards, as detailed in Table I. Being impossible to integrate the desktop computers in a cluster due to local policies, collaborative computing was chosen for harnessing them so BOINC clients were installed. These local resources are accessed by the core via the MachineNow module with the required plugins. On the other hand, the OverFlow module provides access to the Amazon EC2 public cloud.

The Amazon EC2 public cloud provides users a large quantity of machine images that can be booted in a number of different instance types. These types have different characteristics such as number of cores, memory and storage, as detailed in Table II. As it is a public cloud, resources are provided for a fee and depending on the infrastructure location, the price paid for an instance per hour is different. Table II shows the cost for the USA region, which was the one utilized in this demonstration. The *CU* field in the table corresponds to EC2 Compute Units per core, a unit being equivalent to a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor. While only certain EC2 types are suitable for multi-node parallel MPI class of workloads (the *Cluster Compute Instance*, not shown in Table II), others are adequate for low-core count shared memory applications and particularly suitable for bulk submission of serial applications [19].

As it can be seen, the infrastructure accessed by the framework comprises a set of heterogeneous resources. These resources correspond to complete different computing paradigms, offering many possibilities that will be shown next.

B. Behaviour

In this example, the user requests the execution of the AESEncryptDecrypt application via the UI or the API for 500, 1000 and 1500 steps. The core retrieves all related information from the TrustedApps module such the minimum requirements which are required by the application and an estimation of execution time for each available resource type. This estimation can be calculated with a regression taking previous execution times as shown in Figure 2, where times for the Amazon EC2 *Small* instance do not appear because of being out of boundaries.

As highlighted from the results, execution in the local resources will be the preferred option by the core, where it will choose the Tesla nodes because as a dedicated resource, execution starts immediately if free slots are available. On the other hand, availability of desktop PCs is unpredictable so job submission to this resource will be prevented in the case where a deadline was specified in the requirements.

Nevertheless, it may occur that a Tesla cluster node is not available due to other calculations in progress, and with deadline specified the OverFlow module will be used to contact the Amazon EC2 public cloud. The module will start a virtual machine with a custom image where the OpenCL library is installed and so CPU cores can be used instead of GPU ones. Virtual machine allocation and boot sequence do not take more than 30 seconds.

Depending on the instance type chosen (Table II), execution times will vary according to the number of cores and core capability. From Figure 2 it is understood that either *XLarge* and *HighCPU XLarge* instances would be chosen by the core for executing the application if an urgent deadline is specified in the requirements.

However, execution time is not the only factor to evaluate when using commercial cloud resources, as usage cost must be also taken into consideration. In order to provide a compromise between execution time and prices from Table II, a metric called Cost per Performance (C/P), which was explored in [20], is calculated by the core. From values for this metric shown in Figure 3 its clear that the core will choose to start a *HighCPU-Medium* instance for executing the application.

Where the user requires the execution of the application a certain number of times different economics come into play. Throughput can be calculated for each usage hour of Amazon EC2 resources (Figure 4), so the most appropriate instance can be started depending the requirements.

But again, usage costs may be considered when evaluating throughput. For this reason, a Cost per Throughput (C/T) metric can be used. This metric is similar to C/P and its values

[2]http://developer.amd.com/gpu/

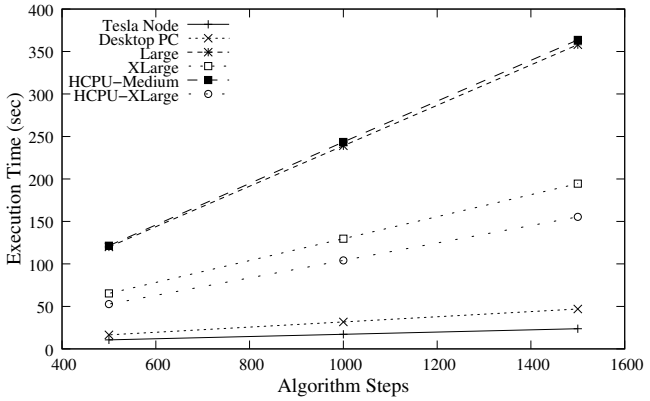

Fig. 2. AESEncryptDecrypt execution times

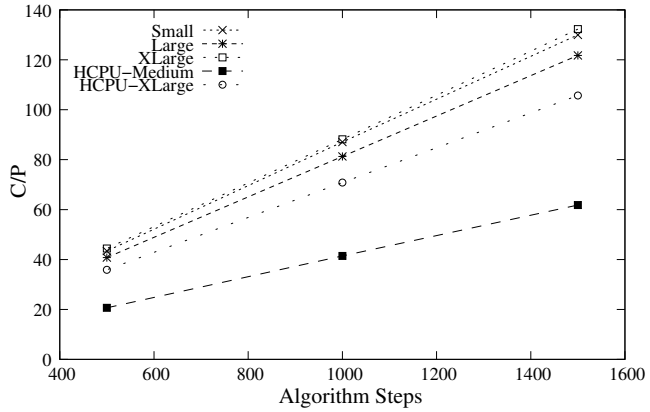

Fig. 3. Values of the C/P metric for AESEncryptDecrypt

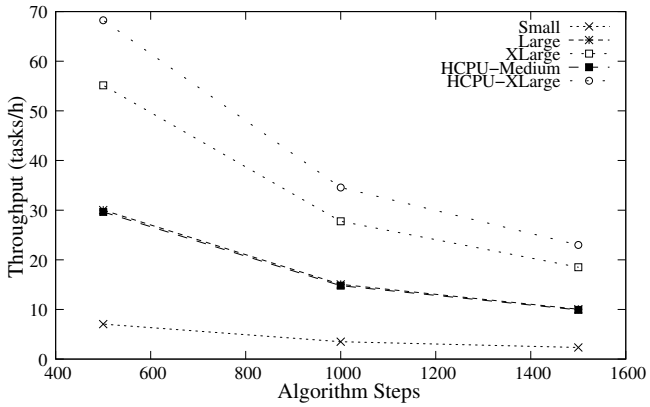

Fig. 4. Throughput (tasks per hour) for AESEncryptDecrypt

for the test case are shown in Figure 5. In this case, the core may choose different machine types depending on the number of executions required.

Not shown in these results are the execution times where only half the virtual cores of the *Extra Large High-CPU* instance (i.e. 4) where utilised by the OpenCL application. In this instance, execution time for the application is approximately 10% more than where all cores are utilised, i.e. scalability with this particular application is curtailed as

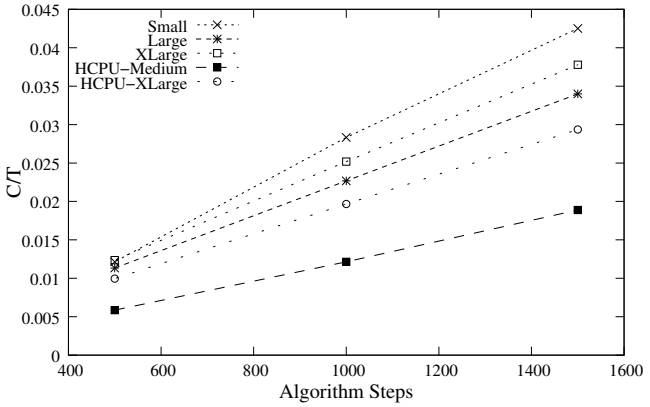

Fig. 5. Values of the C/T metric for AESEncryptDecrypt

the core count is increased. Where absolute performance is important the full complement of cores can be utilised, but for bulk submission jobs throughput performance is significantly increased as two jobs can run concurrently on one instance without oversubscribing the available cores. For a one-off application run by a user such in-depth knowledge of the scalability performance would be difficult, but for the commonly used applications of the TrustedApps module, prior benchmarking could be performed and optimal scheduling policies derived for all anticipated scenarios.

Another aspect to take into consideration is that there could be already started instances, so reuse is an option as public clouds such as Amazon EC2 charges for every hour of resource usage, then the core could decide to follow certain policies whether deploy more instances or wait for the existing ones to become available.

VI. Conclusions and Future Work

In this paper we have outlined a framework for harnessing all heterogeneous computational resources within an organisation, from the dedicated cluster, a local private cloud or a cycle scavenging system consisting of desktops withing an organization and presenting a user with one single access point. Where insufficient local resources are available the required resources can be provisioned from external providers such as a computational grid or commercial public cloud providers.

A sophisticated policy engine will allow users and system administrators to decide where and when the jobs are allocated and allow the consideration of such diverse requirements – for example where energy costs, or the availability of a spot market for compute resources, dictate it may be more efficient, or economical, for an organization to outsource computations. In order to implement the entire framework different queueing and collaborating systems can be integrated in MachineNow, as well as emerging cloud services in OverFlow. However, special efforts must be focused on the core module, specially regarding its policy engine. For this reason, a great number of applications and codes will be tested in order to provide valuable knowledge for the policy database.

As the proof of concept for the outlined THOR framework a local infrastructure consisting of two types of resources was tested.

We understand that GPU computing is driving HPC to new levels, while on the other hand, Cloud computing has revolutionized the way services and computing infrastructures are provided. In the near future, an integration of these two technologies will occur, and for this reason we have developed the THOR framework described in this article.

ACKNOWLEDGEMENTS

This work was partially supported by the NILS mobility programme (EEA-ABEL-03- 2010), the ERCIM "Alain Bensoussan" Fellowship Programme and an NTNU (RSO) Post Doc grant. The first authors would also like to thank Prof. Ignacio Martín Llorente for his support and the support through his grants during this research.

REFERENCES

[1] Khronos OpenCL Working Group, *The OpenCL Specification, version 1.1*, 8 July 2010. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[2] H. Kim and R. Bond, "Multicore Software Technologies," *Signal Processing Magazine, IEEE*, vol. 26, no. 6, pp. 80 –89, november 2009.

[3] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, vol. 12, pp. 66–73, 2010.

[4] P. Kacsuk, T. Kiss, and G. Sipos, "Solving the Grid Interoperability Problem by P-GRADE Portal at Workflow Level," *Future Generation Comp. Syst.*, vol. 24, no. 7, pp. 744–751, 2008.

[5] M. Kloppmann, D. König, F. Leymann, G. Pfau, and D. Roller, "Business Process Choreography in WebSphere: Combining the Power of BPEL and J2EE," *IBM Systems Journal*, vol. 43, no. 2, pp. 270–296, 2004.

[6] S. Murugesan, "Harnessing Green IT: Principles and Practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, 2008.

[7] L. Wang, G. von Laszewski, J. Dayal, and T. R. Furlani, "Thermal Aware Workload Scheduling with Backfilling for Green Data Centers," in *Proc. 28th International Performance Computing and Communications Conference (IPCCC)*, 2009, pp. 289–296.

[8] N. C. Ciocoiu and C. E. Ciolac, "Automated Framework for Green IT Classification using Software Agents," in *Proc. 2nd International Conference in Computer Science and Information Technology*. IEEE Computer Society, 2009, pp. 279–283.

[9] E. Walker, "The Real Cost of a CPU Hour," *Computer*, vol. 42, pp. 35–41, 2009.

[10] W. Gentzsch, "Sun Grid Engine: Towards Creating a Compute Power Grid," in *Proc. 1st IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE Computer Society, 2001, pp. 35–39.

[11] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *Proc. 5th IEEE/ACM International Workshop on Grid Computing (GRID)*. IEEE Computer Society, 2004, pp. 4–10.

[12] J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente, "Coordinated Harnessing of the IRISGrid and EGEE Testbeds with GridWay," *J. Parallel and Distributed Computing*, vol. 66, no. 5, pp. 763–771, 2006.

[13] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE Computer Society, 2009, pp. 124–131.

[15] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "GViM: GPU-accelerated virtual machines," in *Proc. 3rd ACM Workshop on System-level Virtualization for High Performance Computing (HPCVirt)*. ACM, 2009, pp. 17–24.

[16] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized Cloud Infrastructure without the Virtualization," in *Proc. 37th annual International Symposium on Computer Architecture (ISCA)*. ACM, 2010, pp. 350–361.

[17] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, 1995.

[18] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla, "An Efficient Implementation of GPU Virtualization in High Performance Clusters," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, H.-X. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, and A. Streit, Eds., vol. 6043. Springer, 2009, pp. 385–394.

[19] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *USENIX Login*, vol. 33, no. 5, pp. 18–23, 2008.

[20] J. L. Vázquez-Poletti, G. Barderas, I. M. Llorente, and P. Romero, "A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure," in *Proc. 10th Conference on State of the Art in Scientific and Parallel Computing (PARA)*. Lecture Notes in Computer Science, 2010, p. to appear.