Cluster Comput (2014) 17:139–152
DOI 10.1007/s10586-013-0341-0

Virtualizing high-end GPGPUs on ARM clusters for the next generation of high performance cloud computing

Raffaele Montella · Giulio Giunta · Giuliano Laccetti

Received: 2 December 2013 / Accepted: 20 December 2013 / Published online: 5 February 2014

Abstract High performance cloud computing is behind the scene powering "the next big thing" as the mainstream accelerator for innovation in many areas. We describe here how to accelerate inexpensive ARM-based computing nodes with high-end GPGPUs hosted on x86_64 machines using the GVirtuS general-purpose virtualization service. We draw the vision of a possible next generation computing clusters characterized by highly heterogeneous parallelism heading to a lower electric power demanding, less heat producing and more environmental friendliness. Preliminary but promising performance data suggest that this solution could be considered as part of the foundations of the next generation of high performance cloud computing components.

Keywords HPC · Hybrid · ARM · GPU · Virtualization · Cloud computing · Internet of Things

1 Introduction

The availability of computing resources and the need for high quality services are rapidly evolving the vision about the acceleration of knowledge development, improvement, and dissemination. During the so-called web 1.0, the Internet of hyperlinks, HPC, and grid computing technologies rose. According to the common shared knowledge, we are living the final part of the web 2.0, the Internet of Social, powered by the elastic cloud computing technology. The Internet of Things is growing up: the developers and then the users have the power to integrate computation with real stuff control.

1.1 High performance cloud computing

In cloud computing, or Infrastructure as a Service (IaaS) as it is sometimes termed, virtualized computing resources are provided as a network-accessible, pay-as-you-go service [4]. Users employ intuitive, low-touch interfaces to request access to, configure, and manage a virtual and dynamically scalable set of resources that they can views as dedicated to their needs [18]. On-demand access means that resources are available whenever needed. Pay-as-you-go charging means that users only pay for the resources that they consume. Economics of scale allow cloud providers to operate large numbers of computers at modest cost.

High performance computing (HPC) has become an essential technology for a wide range of demanding applications across:

- different kind of sciences (high-energy physics, ocean, weather, climate, computational chemistry, astrophysics, medicine, bio-informatics and genomics);
- engineering (computational fluid dynamics, aerospace, energy);
- economy (econometric methods, market simulations);
- creative arts (virtual sets, image restoration, massive 3d rendering).

In each of these areas, HPC systems have permitted new discoveries and advances.

R. Montella (✉) · G. Giunta
Department of Science and Technology, University of Napoli Parthenope, CDN Isola C4, 80143 Naples, Italy
e-mail: raffaele.montella@uniparthenope.it

G. Giunta
e-mail: giulio.giunta@uniparthenope.it

G. Laccetti
Department of Mathematics and Applications, University of Napoli Federico II, Via Cintia, Complesso Monte S. Angelo, 26, 80125 Naples, Italy
e-mail: giuliano.laccetti@unina.it

However, traditional HPC resources have significant limitations. HPC systems are expensive and thus access is often restricted. Batch scheduling algorithms mean that jobs must frequently wait in a queue for execution. At the same time, it can happen that individual machines are under-utilized, depending on time-varying demand within a particular institution or community.

Such considerations have motivated interest in high performance cloud computing (HPCC) [13].

The goal of HPCC is to combine the powerful processors and high-speed, low-latency interconnection networks of the high performance computing with the virtualized, elastic access of IaaS [15].

In so doing, we may increase flexibility relative to traditional HPC while also improving efficiency in terms of cost, energy consumption, and environmental friendliness.

In particular, management methods that permit the creation of elastic virtual clusters may provide users with capacity and environment that meet the demands of specific workloads without the need to purchase and operate dedicated hardware and software [10].

The use of virtual machines (VMs) can provide users with administrative privileges within the guest operating system and thus allow them to customize the runtime environment according to their specific requirements [8].

The limitations related to the performance requirements will be mitigated in the close future thanks to the increase of acceleration technologies as GPGPUs [17] and MICs devices both based on a massive many-core approach and, above all, different CPU architectures [26].

1.2 Accelerator devices

Highly parallel graphics-processing units (GPUs) are rapidly gaining maturity as a powerful engine for computationally demanding applications.

Researchers and developers have become interested in harnessing this power for general-purpose computing, an effort collectively known as General-Purpose computing on the GPU (for GPGPU) [20].

Especially for parallel computing applications, virtual clusters instantiated on cloud infrastructures suffer from poor message passing performance between virtual machine instances running on the same real machine. Furthermore, they cannot access hardware-specific accelerators such as GPUs.

Virtualization allows a transparent use of accelerators such as nVidia CUDA-based GPUs using split-driver based components as GVirtuS [6,7,25].

Intel Xeon Phi coprocessors offer all standard programming models that are available for Intel Architecture, including OpenMP [23], POSIX threads, and MPI. The Intel Xeon Phi coprocessor plugs into a standard PCIe slot and provides a well-known, standard shared memory architecture.

For programmers of higher level programming languages like C/C++ or Fortran using well-established parallelization paradigms like OpenMP, Intel Threading Building Blocks, or MPI, the coprocessor appears like a symmetric multiprocessor (SMP) on a single chip [24].

Compared to accelerators this reduces the programming effort a lot, since no additional parallelization paradigm like CUDA or OpenCL needs to be applied (although Intel Xeon Phi coprocessors also supports OpenCL) [27].

1.3 The rise of the Internet of Things

Recently, the Internet of Things (IoT) gained the power of world spread programmable microcontrollers and credit card size (and extremely low cost) full-featured computers. Raspberry Pi (RPi) is the other main actor on the contemporary stage of the today scene of making the IoT as a sort of another lightning strike in the open-source hardware movement. RPi is equipped with an ARM processor powered with a GPU, a RAM of 256M or 512M, a SD card slot as primary storage unit, an USB and an Ethernet connection.

The new generations of developers are building the IoT using this kind of devices as construction bricks; the hungriness and the foolishness of the creative are raising the need for computing power. This need could be satisfied by the cloud computer technology that meantime evolved in a stable and business oriented form as the accelerator device for science investigation, engineering calculations and common life media sharing as well.

In the past, data centers relied on purpose-built servers with highly powerful processors, whilst today the dominant approach is to build datacenters from commodity hardware components.

The same processors used in general purpose computing, e.g. workstations, are now used in servers. Following Moores law, the performance of these general-purpose processors has greatly improved. Although their energy-efficiency has also improved, low energy consumption has, until recently, been a secondary objective.

The performance of embedded processors naturally lags the performance of general-purpose processors.

It is interesting to ask if a large number of such low-power, low-performance processors could be used to build a data center with similar processing power but smaller energy consumption.

1.4 Advanced RISC macine CPUs for HPC

General-purpose processors have already overtaken powerful purpose-built processors in data centers.

ARM processors, designed for the embedded mobile market, operate at about 1 GHz and consume just 0.25 W. There is already a significant trend towards using ARM processors in data servers and cloud computing environments in which workloads are limited by I/O and memory systems, not by CPU performance.

Recently, ARM processors have also taken significant steps towards increased double precision (DP) floating point (FP) performance, making them competitive with state-of-the-art servers [1].

The ARM Cortex-A15, targeted as the computing unit in the Barcelona Supercomputing Center Mont Blanc project, will increase super-scalar issue to two arithmetic instructions per cycle, and has a fully pipelined FMA unit, delivering 4 GFLOPS at 1 GHz, on potentially the same 0.25 W budget, achieving 16 GFLOPS/W.

The new ARMv8 instruction set, which will be implemented in future generations of ARM cores, features a 64-bit address space, and adds DP to the NEON SIMD ISA1, allowing for 8 ops/cycle on an A15 pipeline: 8 GFLOPS at 1 GHz, for 32 GFLOPS/W.

1.5 Computing power for HPCC by hybrid GPU/x86_64/ARM

We present here preliminary results of a project that aims to create HPC clusters dedicated (but not limited) to HPCC service provision using low-power ARM-based computing nodes grouped in sub-clusters leveraging one or more high-end GPGPU devices hosted on so-called accelerator nodes.

We report on experiments conducted in a controlled testing environment that we have constructed to imitate the core of a more complex architecture and based on a Intel x86_64 based accelerator node acting as I/O manager for a ARM-based sub-cluster built using Raspberry-Pi boards and powered by two high end nVidia Tesla C1060.

The results of these experiments are extremely promising, showing the tiny impact of GVirtuS latency on the overall performance balanced by an embarrassing reduction of the wall-clock time.

We also report on experiments conducted on an expanded experiment setup in which additional R-Pi computing nodes are used to imitate a sub-cluster in which each node shares the GPGPUs hosted on the x86_64 machine.

Finally, thanks to the lesson learnt by this experiment, we carried out some discussions about the implications of this technology on the high performance cloud computing.

The rest of this paper is organized as follows: in Sect. 2, we expand upon our vision of the next generation of hybrid HPC clusters constructed with ARM-based components and high-end GPUs. Section 3 deals with design and technical issues on GVirtuS, while the Sect. 4 is about the hybrid GPU / x86_64 / ARM software architecture using GVirtuS as transparent bridge between the ARM living applications and the GPUs. Section 5 covers implementation details and in Sect. 6 some tests and preliminary results are described and discussed. Section 7 discusses next-generation HPCC infrastructures and applications that that these new technologies may support. Finally, Sect. 8 covers conclusions and future directions.

2 Vision and context

The world of supercomputing has evolved rapidly since its first steps in the mid 1980s to the second decade of the new millennium.

As the two top charts for computer brute power (Top 500) and best compute/cost efficiency (Green 500) show, we have two trends: the number of cores is increasing thanks to the use of dedicated accelerators (GPUs, CPU array boards) and power efficiency is of increasing importance. Indeed, in the future the two charts may merge, with the environmental (and economic) footprint of a HPC iron giant as a primary requirement [23].

The development of high performance cloud computing permits a democratization of science acceleration thanks to its pay-per-use model: a person who wants to use HPC need not place a Top 500 computer under their desk but instead just requires access to a cloud provider.

For many applications as operational computations [19] or for the cloud hosting providers the energy saving is no more a freak item but a mandatory issue.

Nowadays, as the power of CPUs increases (clock frequency, number of cores, cache size) the need for electric power rises as well as feeds the vicious circle of the need for system cooling.

As previously stated, the availability of Internet of Things derived from ARM CPUs in their high performance incarnation (64bit, multicore) lead the HPC world to ARM based clusters powered with on chip or on board GPUs.

The regular computing nodes of a HPC cluster (still executing jobs at the higher parallel level) act as Input/Output dedicated machines respect to ARM based inexpensive, less energy hungry and cooler sub-clusters (Fig. 1).

In this way the amount of heat producers decreases, while the high computing power demanding applications have to be refactored in order to fit this new heterogenic approach. The contribute to this vision, partially yet on the HPC mainstream, is the novelty of the accelerator nodes that represent the new big guy in this scenario.

The accelerator nodes, as the I/O nodes, are x86_64 architecture based and host one or multiple high-end, but relatively low-cost, GPGPU devices.

Thanks to the software component shown in this paper, these devices are seen by each of the ARM based sub-cluster

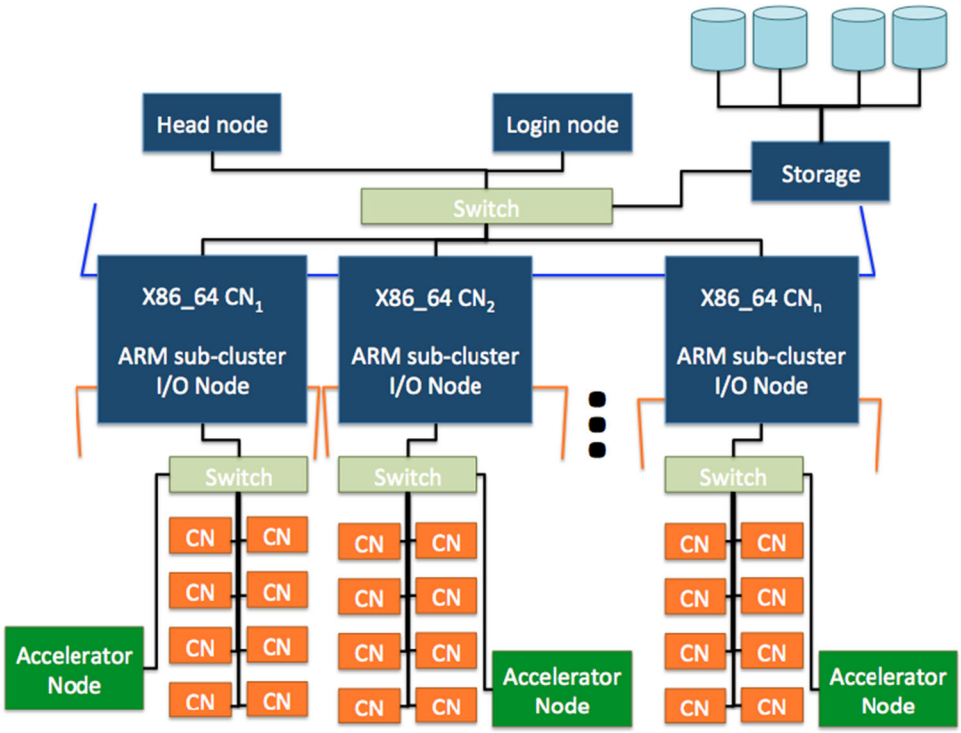

Fig. 1 The proposed architecture with a hybrid mix of x86_64 and ARM computing nodes accelerated by high-end GPGPUs

computing nodes as directly connected to them in a transparent way.

This vision permits to gain more computing power reducing the expensive, power hungry and heat producer x86_64 based computing nodes, increases the parallelism at the sub-cluster level and, last but not the least, unchains the high-end GPGPU power to ARM based computing nodes.

Finally, a really high parallel heterogenic scenario just appeared behind the curtain: shared memory in multicore ARM based nodes, distributed memory between sub-cluster nodes, shared memory in multicore I/O x86_64 based nodes, distributed memory among I/O nodes, distributed memory, via GVirtuS, among ARM (or x86_64) nodes and the GPU devices hosted by the accelerator nodes.

3 The generic virtualization system: GVirtuS

The latest implementation of GVirtuS extends and generalizes a previously developed GPU virtualization solution proposed in our past works.

The main motivation of the first generation of GVirtuS was to address the limitations of transparently employing accelerators such as CUDA-based GPUs in virtualization environments. GVirtuS could be considered as a generic virtualization framework for facilitating the development of split-drivers for virtualization solutions.

The brightest GVirtuS feature is the independence from all involved technologies: the hypervisor, the communicator and the target of the virtualization, as demonstrated later in this work.

Using a plug-in based design, GVirtuS offers virtualization support for generic libraries such as accelerator libraries (OpenCL, OpenGL, CUDA) and parallel file systems and communication libraries (MPI/OpenMP).

GVirtuS could be seen as an abstraction layer for generic virtualization in HPC on cloud infrastructures [22].

In GVirtuS the split-drivers are abstracted away, while offering developers abstractions of common mechanisms, which can be shared for implementing the desired functionality.

In this way, developing a new virtualization driver is simplified, as it is based on common utilities and communication abstractions.

For each virtualized device the frontend and the backend are cooperating, while both of them are completely independent from the communicator. Developers can focus their efforts on virtual device and resource implementation without taking care of the communication technology.

3.1 The front-end

The front-end is a kernel module that uses the driver APIs supported by the platform.

The interposer library provides the familiar driver API abstraction to the guest application. It collects the request parameters from the application and passes them to the back-

end driver, converting the driver API call into a corresponding front-end driver call.

When a callback is received from the front-end driver, it delivers the response messages to the application. In GVirtuS the front-end runs on the virtual machine instance and it is implemented as a stub library. A stub library is a virtualization of the physical driver library on the guest operating system.

The stub library implements the driver functionality in the guest operating system in cooperation with the back-end running on the host operating system.

3.2 The communicator component

The communication between the front-end and back-end is done via abstract communicators.

GVirtuS provides several communicator implementations including a TCP/IP communicator we used in this kind of application.

In GVirtuS, the use of TCP/IP based communicator is not feasible for HPCC application because the performance is strongly impacted by the protocol stack overhead.

The communicator maps the request parameters from the shared ring and converts them into driver calls to the underlying wrapper library.

Once the drivers call returns, the backend passes the response on the shared ring and notes the guest domains.

The wrapper library converts the request parameters from the backend into actual driver API calls to be invoked on the hardware. It also relays the response messages back to the backend.

Finally, the driver API is the vendor provided API for the device.

3.3 The back-end

The back-end is a component serving front-end requests through the direct access to the driver of the physical device.

This component is implemented as a server application waiting for connections and responding to the requests submitted by frontends.

In an environment requiring shared resource, the back-end must offer a form of resource multiplexing. Another source of complexity is the need to manage multithreading at the guest application level.

4 GVirtuS on ARM

An ARM port of GVirtuS is motivated raised from different application fields such as high performance Internet of Things (HPIoT) and cloud computing.

In HPC infrastructures, ARM processors are used as computing nodes often provided by tiny GPU on chip or integrated on the CPU board.

Nevertheless, for most massively parallel processing applications, as scientific computing, are too compute intensive to run well on the current generation of ARM chips with integrated GPU.

In this context we developed the idea to share one or more regular high-end GPU devices hosted on a small number of x86 machines with a good amount of low power/low cost ARM based computing sub-clusters better fitting into the HPC world.

4.1 Architecture

From the architectural point of view this is a big challenge for reasons of word size, endianess, and programming models.

For our prototype we used the 32-bit ARMV6K processor supporting both big and little endian so we had to set the little endian mode in order to make data transfer between the ARM and the x86 full compliant. Due to the prototypal nature of the system all has been set to work using 32 bits.

The solution is the full recompilation of the framework with a specific reconfiguration of the ARM based system. As we will migrate on 64 bits ARMs this point will be revise.

In order to fit the GPGPU/x86_64/ARM application into our generic virtualization system we mapped the back-end on the x86_64 machine directly connected to the GPU based accelerator device and the front-end on the ARM board(s) using the GVirtuS TCP/IP based communicator.

GVirtuS as nVidia CUDA virtualization tool achieves good results in terms of performances and system transparency.

In the work presented in this paper, we choose to design and implement a GVirtuS plugin implementing OpenCL. This has been strongly motivated by several issues:

1. Since CUDA version 4, the library design appears to no longer fit with the split driver approach leveraged by GVirtuS and other similar products;
2. OpenCL is intrinsically open and all interfaces are public and well documented and, above all, work with nVidia devices, but is not limited to a particular vendor or architecture as GVirtuS itself;
3. OpenCL applications can be compiled directly on the ARM board without any installation of ad hoc libraries.

The OpenCL applications are executed on the ARM board through the GVirtuS front-end.

Thanks to the GVirtuS architecture, the front-end is the only component needed on the guest side.

This component acts as a transparent virtualization tool giving to a simple and inexpensive ARM board the illusion

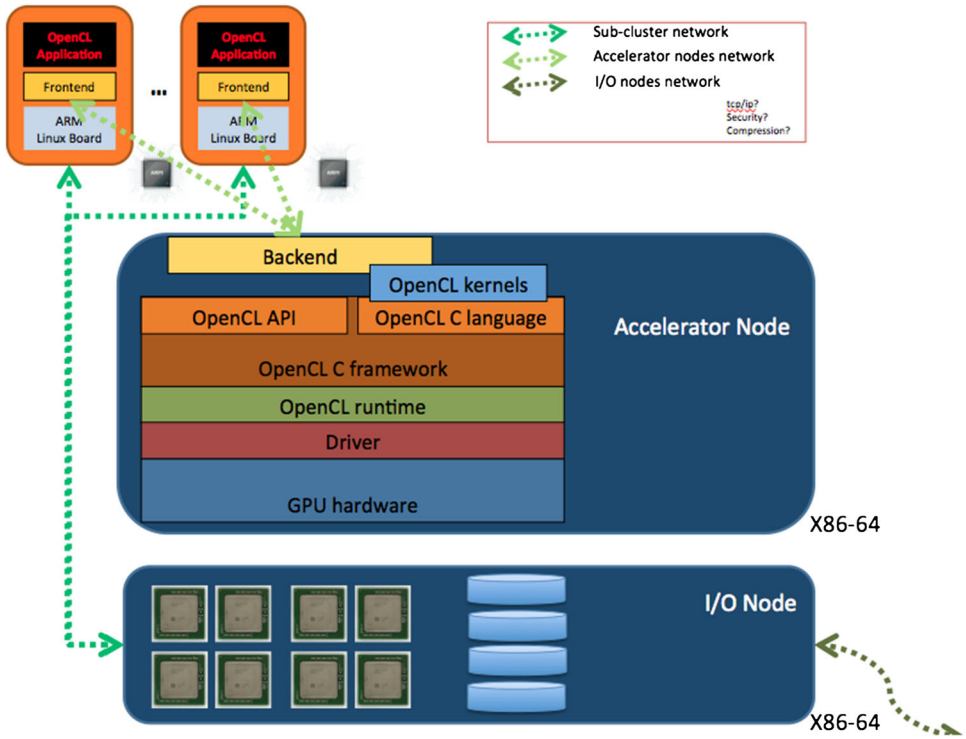

Fig. 2 The GVirtuS on ARM schema diagram

to be directly connected to a high-end OpennGL compatible GPGPU device or devices (Fig. 2).

4.2 GVirtuS–OpenCL plugin

Open computing language (OpenCL) is an open and royalty-free standard allowing to perform multi/single core general purpose programming on highly heterogeneous systems.

OpenCL allows developers to write their code once and run on CPUs and GPUs and different accelerator boards such as the Mic-based Intel Phi.

In order to access a GPU in a virtual environment, we have developed a wrapper for libOpencl.so.

The virtualized library has the same interface as the original and the independence from the communicator is guaranteed. Compatibility between the virtualized interface and libOpenCL.so allows the users to obtain a transparent virtualization system to run OpenCL applications. Indeed it is possible to run any OpenCL application without writing or recompiling any code.

The GVirtuS–OpenCL Plugin comprises two main components: the front-end, running on the guest machine, and the back-end, running on the host machine.

For each OpenCL routine invoked by the calling program, the front-end serializes the data and sends them to the back-end. It requests the execution of the routine and then it tries to get the exit code of the routine.

The back-end intercepts a call made by the fronted, it reads up and de-serializes the parameters, it executes the routine of the library and then it sends back to the frontend the results.

In more detail, following the GPU virtualization operation as workflow, we could spot on each working component.

Front-end side: for each OpenCL routine a stub method has been implemented with the same interface of the original one. All the stub methods have a common implementation consisting in the next five steps:

- Create a connection between back-end and front-end and flush all the buffers;
- Each parameters will be sent to the back-end through the input buffer;
- Request the execution of a routine using its name as parameter;
- Get and Use the exit code only if the execution is successful;
- Return the exit code the same one as the OpenCL routine.

There are tree main input parameter types available:

- Host Pointer: back-end and front-end have different addressing space; a valid pointer on the front-end is invalid on the back-end and vice-versa. Aligning the addressed region makes the address translation.
- Device Pointer: the memory address is sent to the back-end (or front-end). There is no need for translation because both, back-end and front-end, refer to the device addressing space.

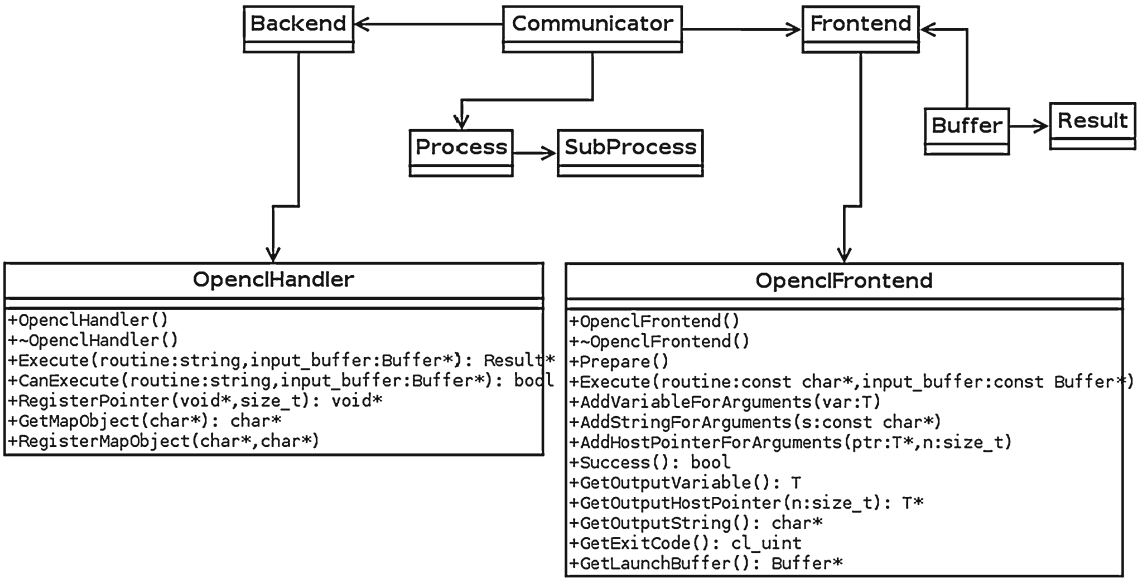

Fig. 3 A high level view of the GVirtuS class diagram with some GVirtuS ARM OpenGL details

– Variables: it is really simple to add a scalar variable as a parameter.

In order to make the implementation effective and high performance, but with a good trade off in development straightforwardness, we deeply used an OOP coding approach (Fig. 3).

5 Implementation

The implementation, in C++ for all components, on the back-end side is related to an x86-based multi-core hardware platform with multiple accelerators attached via PCIe devices, running Linux as both host and guest operating system.

In the font-end we used the same core running in a similar, but ARM based, Linux environment. According to the underlying idea of high performance cloud computing applications, we implemented the virtual transparent accelerator in a really architecture independent fashion and in a fully configurable way working in both hypervisor and non-hypervisor configuration.

The GPU is attached to the host system, the accelerator node in this context, and must run its drivers at a privileged level directly accessing the hardware device.

Memory management and communication methods for efficient sharing of the GPU device by multiple guest users have to be implemented at the same run level.

The activity diagram (Fig. 4) represents the big picture of the virtualization/multiplexing process.

5.1 OpenCLFrontend

The OpenCLFrontend class establishes connections with the back-end and executes the OpenCL routine through the compiled library libGvirtus-frontend.

The constructor method creates an object of the class Frontend from the libGvirtus-frontend library using the method GetFrontend using a factory/instance design pattern.

This instance of the class will be alive through all the life cycle of the application and it will be used any time we need a method from the OpenCLFrontend class.

The stub methods all have a common schema. Every stub follows the same interface of the handled OpenCL routine.

The description of this method is significant for the explanation of any other method.

The first step is to get the unique instance of the GVirtuS Frontend class. This task is accomplished by the constructor method.

The Prepare method resets the input buffer that will contain the parameters to send to the back-end.

After that all the parameters are inserted into the input buffer.

The execute method forwards the request for the routine using the name of the routine as parameter.

If the method is successfully executed, we can get the output parameters.

At last the method GetExitCode returns the exit code of the routine executed by the back-end.

The CLGetDeviceIDs routine can be used to obtain the list of available devices on a platform.

This simple explicative schema is common to all the stubs coded.

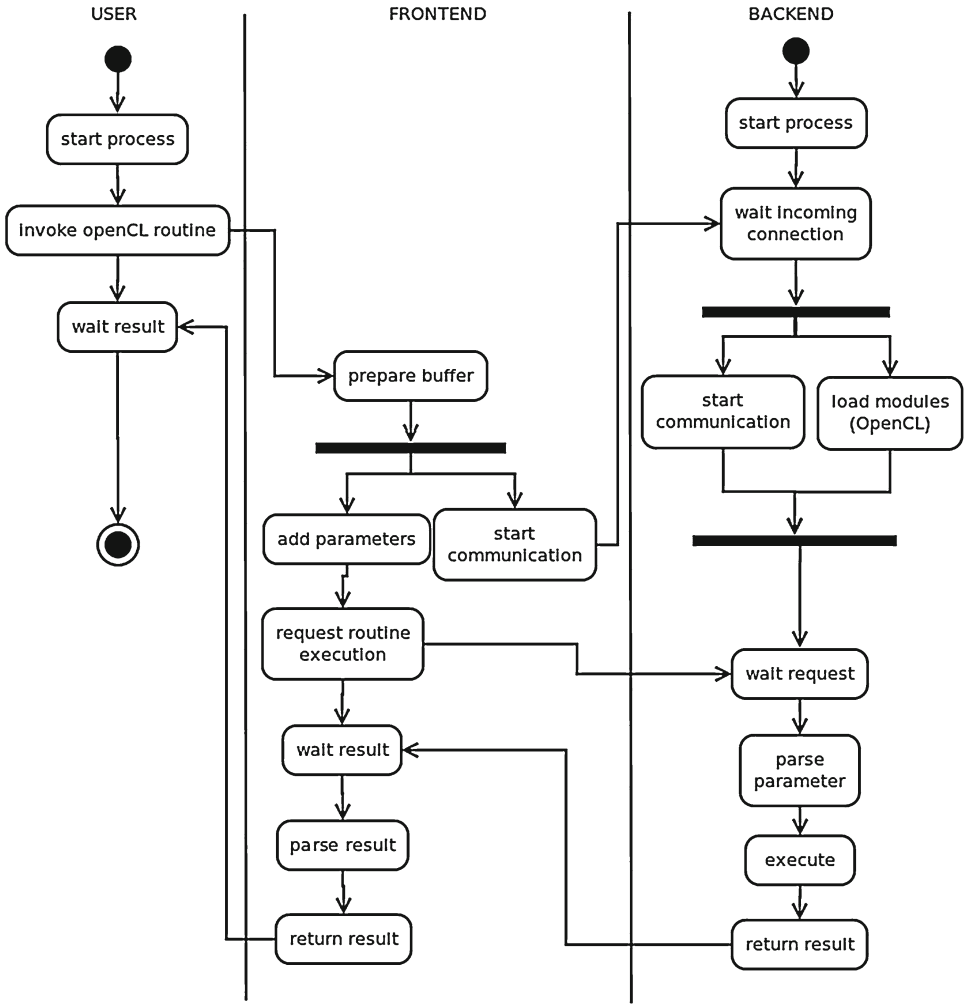

Fig. 4 The GVirtuS ARM OpenCL activity diagram

5.2 OpenCLBackend

An ad-hoc file named gvirtus.properties configures GVirtuS. GVirtuS only handles two parameters at this stage of development, communicator and plugin.

The first parameter selects which kind of communicator has to be used choosing from a list of available communication mechanisms, the second one selects which plugin must be loaded.

The main task of GVirtuS back-end is to start a communication in server mode and waiting then accepting new incoming connections.

It handles the loading of plugins previously installed. GVirtuS back-end invokes the GetHandler method in order to create a new instance of OpenclHandler class containing all the methods needed in order to serve the requests of OpenCL routine execution.

In this class it is possible to find all the methods to handle the execution of OpenCL routines.

In the OpenclHandler class there is a table, mpsHandlers, associating function pointers to the name of the routines, so any routine can be handled in the right way.

As in the front-end there is a stub method for each OpenCL method, in the back-end there is a function managing the execution of each method.

6 Evaluation

We set a prototypal hardware environment in order to evaluate the performance on ARM acceleration using external x86_64 GPUs, the GVirtuS overhead and the result reliability of a software testing suite.

That evaluation process has two specific goals:

- check the software stack accountability;
- gather results on performance test.

The OpenCL SDK provides a software suite in which each component performs computations in both CPU and GPU modes, checking the result coherence and showing the brute performance results.

All tests available on the standard OpenCL SDL have been successfully run using the GVrtuS-OpenCL SDK.

6.1 Accelerator node and single ARM computing node

We used a Raspberry Pi Mod.B rev.2 ARM 11 equipped with Wheezy Raspbian Linux as the compute node, a Genesis GE-i940 Tesla powered by an i7-940 2.93 GHz fsb, Quad Core HT 8Mb cache with one nVIDIA Quadro FX5800 4Gb as the graphic device and two nVIDIA Tesla C1060 4Gb as GPGPU devices as accelerator nodes.

For those tests no I/O node has been provided and the setup is related on a single node sub-cluster.

In this context the GVirtuS front-end was run on the ARM computing nodes and the back-end on the accelerator node.

We used the OpenCL version of the testing software known as MatrixMul, DotProduct and Histogram.

- DotProduct computes k scalar products of two real vectors of length m.
  Notice that an OpenCl thread on the GPU executes each product so no synchronization is required.
  During the DotProduct testing process we change the problem dimension from 2^{20} to 2^{22}.
  The ARM performance is varying with the same problem dimension trend. The wall clock remains almost constant when is used the GPU acceleration.
  This demonstrates that the GVirtuS-OpenCL is working fine and the performances are not affected by the communication time. In order to perform these tests, DotProduct ran on vectors of 1M, 2M, 4M of elements.
- MatrixMul computes a matrix multiplication.
  The matrices are m n and n p, respectively. It partitions the input matrices in blocks and associates a CUDA thread to each block. As in the previous case, there is no need of synchronization.
  In the MatrixMul test the dimension problem has been varied in this steps $2^6 \times 2^9$, $2^9 \times 2^{12}$ and $2^{10} \times 2^{11}$.
  The performance results are pretty similar to the previous case with the GPU version having wall clock times almost unchanged.
  The size of the problem increases at every execution, MatrixMul ran on matrices of 16, 524 k and 2 M of elements.
- Histogram returns the histogram of a set of m uniformly distributed real random numbers in 64 bins.
  The set is distributed among the CUDA threads, each computing a local histogram.
  The final result is obtained through synchronization and reduction techniques.
  The Histogram has been used varying the problem size to 2^4, 2^5 e 2^6.
  The meaning of the performance tests results is trivially the same running Histograms on vectors of 4, 8 and 16 M of elements.

Table 1 summarizes the results obtained considering the regular ARMV6K as reference. Comparing the charts we can underline the difference in order of magnitude.

The overall runtime using the GPU acceleration through GVirtuS and OpenCL, needs less than 1 % of the time taken by the standalone ARM CPU, through this comparison the effectiveness of the proposed solution is shown (Fig. 5).

The best results have been achieved with the MatrixMul benchmark in which the GPU runs scored an average time of 2,500 better than the non GPU runs.

6.2 Accelerator node and ARM sub-cluster

We expanded our experiment setup to an ARM based sub-cluster of four computing nodes, each with the same characteristics as the single node used previously. Due to the limitations related with the ARM boards used in these experiments, we set up a simple network connection using a 100 Mbps Ethernet over copper infrastructure.

In this scenario, some other actors enter the stage, namely MPICH (http://www.mpich.org, version 3.0.4) for ARM to ARM and ARM to x86_64 message passing, OpenMP for intra ARM board parallelism and, above all, one or more GPU devices hosted on the accelerator node have to be multiplexed by several ARM processes.

Algorithm 6.1: MATRIXMULOPENCLMPI(n, A, B)

```
if !amIaNode()
then
    nodes ← []
    for i ← 0 to n
      do nodes[i] ← prepareNode(i)
    n_rows ← A.n_rows/n
    n_cols ← B.n_cols/n
    for i ← 0 to n
      do
        send(
          nodes[i],
          A.rows[(n_rows ∗ (i − 1))...
            (n_rows ∗ i)])
        send(
          nodes[i],
          B.cols[(n_cols ∗ (i − 1))...
            (n_cols ∗ i)])
    C ← matrixAllocate(A.n_rows, B.n_cols)
    for i ← 0 to n
      do
        nodeC ← receive(nodes[i])
        C[(n_rows ∗ (i − 1))...
          (n_rows_i)][n_cols ∗ (i − 1)...
          (n_cols ∗ i)] ← nodeC
    return (C)
else
    localA ← receiveFromMaster
    localB ← receiveFromMaster
    deviceA ← copyOnGPU
    deviceB ← copyOnGPU
    deviceC ← matrixMulOnGPU(
      deviceA, deviceB)
    localC ← copyFromGPU(deviceC)
    sendToMaster(localC)
```

Table 1 Best results in accelerator node and single ARM computing node

Test	Maximum input size (MB)	Relative (%)	Increment
MatrixMul	4	0.04	× 2,500
DotProduct	4	0.27	× ∼370
Histogram	16	0.65	× ∼154

The performances are computed respect the not accelerated run with the same problem size

We developed an ad hoc benchmark software that implements a matrix multiplication algorithm.

This software uses a classic distributed memory approach to parallelization (Algorithm 6.1).

The first matrix is distributed by rows and the second by columns, and each process must perform a local matrix multiplication.

We used MPICH for message-passing among processes and the OpenCL library to perform the local matrix multiplication within each process.

Figure 6 shows the results of the performance test.

We measured the time taken on 1, 2, and 4 computing nodes to multiple square matrices of for different problem sizes, namely 1, 4, 16 MB and 64M.

The topside of Fig. 6 presents results obtained by running the MPI-based algorithm on the cluster.

The bottom side of Fig. 4 gives a comprehensive summary of the tests of GVirtuS based algorithm for the case CPU only and the case OpenCL-GPU.

These results show that the GVirtuS GPU virtualization and the related sharing system allow an effective exploitation of the computing power of the GPUs.

We note that without such component the ARM machine could not see the GPUs and it would be impossible to run this experiment (Fig. 6).

6.3 Discussion

The results of the two experiments discussed in 6.1 and 6.2 represent two sides of the same coin. The single node experiment setup (6.1) demonstrates that:

- Its technically possible making sharable a GPU accelerator device hosted by a x86_64 machine with an ARM based device;
- If the ARM has poor computing power (CPU, memory), the performance achieved is embarrassingly huge with the top scored with the matrix multiplication benchmark (x2500 respect to a not accelerated ARM board);
- Under these conditions, the network connection between the ARM board and the x86_64 accelerator board is a not so much relevant bottleneck and the GVirtuS latency is negligible.

Changing the experiment setup increasing the ARM device computing power could reduce de performance increment magnitude because the network connection and the relative weight of the GVirtuS latency.

Assuming a high-end ARM device sharing a high-end GPU, the best real world applications working in this environment have to be search in the class load ones, use many times data with a strong constraints given by the GPU(s) memory.

The lesson learnt by the experiment discussed in 6.2 gains interest if focusing on the next generation HPCC market. The experiment demonstrates:

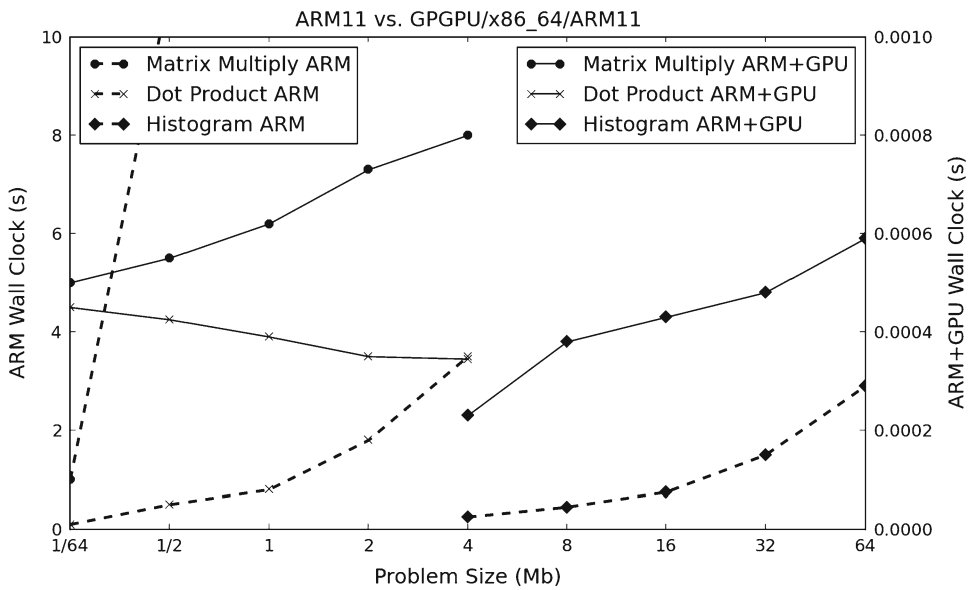

Fig. 5 Accelerator node and single ARM computing node. Performance tests with Matrix Multiply, Dot Product and Histogram benchmarks. *Left* ARM. *Right* ARM accelerated by the Tesla C1060

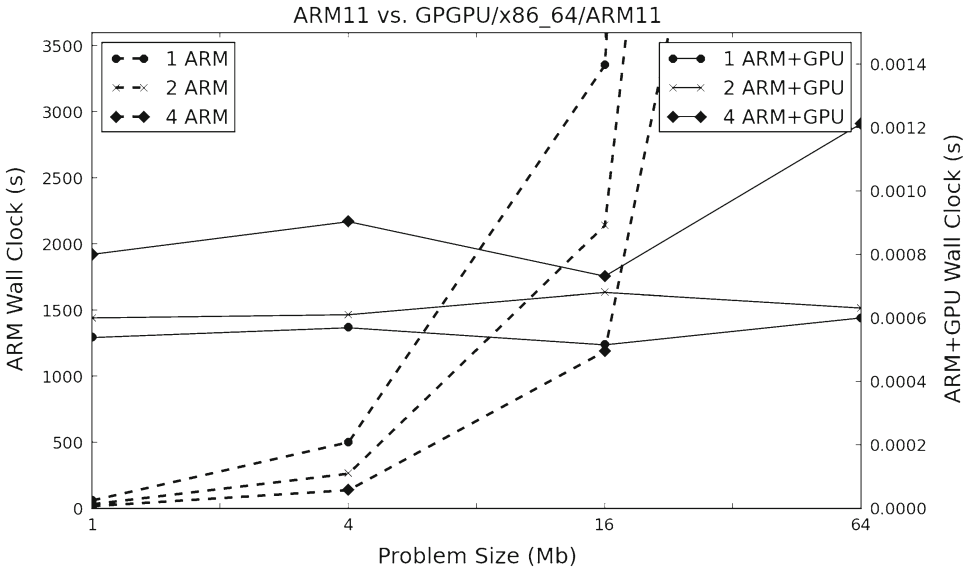

Fig. 6 Accelerator node and ARM sub-cluster. Performance tests with MPI Matrix Multiply. *Left* ARM. *Right* ARM accelerated by a Tesla C1060

- GVirtuS makes the hybrid architecture (GPU, x86_64, ARM) resource sharing ready for the prime time: it works, nevertheless with some important performance issues.
- The communication bottleneck has a huge weight on the overall performances. In this setup the network is low performance and shared by MPI message passing, GVirtuS remote invocation and the sub-cluster services as NFS.
- Comparing the performance of the same problem size using just one sub-cluster node and the related 6.1 case, we could carry out the effect of MPI latency.
  As the number of nodes increases, the network is over and over congested preventing a correct performance scaling.
- Finally, the data memory alignment could have a little, but appreciable effect in the case of a GPU shaded by different process on different computing nodes.

Nevertheless this open the path to a consistent number of improvement approaches with the main goal to render performance affordable what is technically possible.

Using more performing ARM based boards with more cores, more memory and a better network subsystem, the proposed solution could be interesting for a wide spread of applications with particular regards to computational fluid dynamics, machine learning and image recognition and processing.

7 Evolving the next generation of high performance cloud computing

The industry has seen a huge transition to hyper-scale as lower-cost ARM processors increase server density and push them further toward commodity status. Service organizations increasingly allow such processors [14].

In the future, data centers will increasingly be built from large numbers of ARM powered machines with a reduced need for service and maintenance that, as it happens with the redundant array of inexpensive disks, when they fail, you just mark them as bad and replace them.

Data centers and processors are already being redesigned around the cloud with the goal to bring the benefits of cloud computing to the hyper-scale community.

Users can get the processes, familiarity and scalability that they like from the cloud but can control costs, energy and data locality of those services to suit them better than they can when they are throwing their workloads into a public environment.

In order to provide computing power in a context of high performance cloud computing, the whole system has to be trimmed to provide computation at an affordable cost per performance, taking into account how to optimize the subsystems for memory, storage and other accelerators.

The proposed solution mixes x86_64 machines, GPGPUs and, potentially, other accelerators as Xeon Phi with ARM computing resources that could interact following different scenarios and shape-shifting configurations unchaining new high performance cloud computing programming models and real world applications.

The scenarios described in the next subsections have a potentially huge impact on the development of the knowledge and to the democratization of the computational sciences because the focus is on performing complex science and engineering workflows just instancing pay-per-use resources.

7.1 VMs on ARM computing nodes with acceleration

No such hardware virtualization as the Intel VT and then AMD-V has been supported for ARM, since the introduction

of the XEN/ARM development has been accelerated after the introduction of the ARM Virtualization Extensions and Large Physical Address Extension (VEs, LPAE).

This two technologies enable the efficient implementation of virtual machine hypervisors for ARM architecture compliant processors in order to handle complex software with potentially large amounts of data, connected consumer devices, energy efficient demanding cloud computing resources, high performance systems [2].

This effort has been mainly driven out by the mobile device industry in order to increment the security and the cloud computing business.

The VEs standardizes the architecture for implementation of the hardware acceleration in ARM application processor cores, while high performance hypervisors from the worlds leading virtualization companies provide the software component upon which to build effective software combinations.

Cloud computing and other data or content oriented solutions increase the demands on the physical memory system from each virtual machine.

Virtual machines instances run on the ARM computing nodes using GPGPUs on the accelerator nodes as a shared, multiplexed and totally transparent shared resource [11].

In this scenario the ARM clusters are a (total/partial) replacement of old-style x86_64 machines.

7.2 ARM sub-clusters as x86_64 accelerators

In this scenario virtual machine instances run on x86_64 computing nodes sharing multiplexed GPGPUs hosted on the accelerator nodes that, novelty, mediate the use of the ARM sub-clusters as accelerators.

This computing resource could be enforced by different management philosophies.

A virtual machine instance could claim the exclusive use of one or more ARM sub-cluster(s) in order to perform computations natively interacting with a sort of hieratical and/or heterogenic local scheduler.

A second approach could be based on ARM processes embedded in VMs and then spawned by the x86_64 VM on the sub-cluster.

Technically, the native ARM or the ARM VMs could be accelerated by GPGPUs on the accelerator node, but the efficiency and the effectiveness of this kind of approach should be proven as affordable for specific class of applications.

In order to make the HPCC compliant with this scenario, the development of new programming models has to be performed.

In particular, the ad hoc software has to be bundled with binary code for the main VM and for each kind of accelerator as happens with GPGPU kernels [5].

7.3 High performance Internet of Things

This third scenario is concerned big data, sensor networks, and the Internet of Things (IoT), rather than conventional HPCC.

In this scenario, ARM-based sub-clusters act as proxy machines for complex data acquisition instruments that VMs hosted on classical-style x86_64 machines could see as dynamic and elastic resources.

In order to abstract different kinds of instruments based on a wide range of PC cards or microcontrollers (such as drones, rovers, ROVs, weather stations, surface current radars, and weather radars) a a plug-in framework is needed because of differences in both hardware interfaces and acquisition data rates [16].

Targeting our final goal of instrument sharing as IoT components, the use of commonly accepted and wide spread technologies and tools as web services is a mandatory approach.

Nevertheless, some things are characterized by behaviors difficulty matching with the latency of SOAP services or with the resource approach of REST APIs.

Pointing the attention of a social addressed Internet of Things where the main target is the increasing of the quality of human life, environmental data acquisition instruments interact with their proxy hardware in different ways: weather station data loggers could work in real-time or in batch mode in respect of the data link type, the coastal and weather radars work in a similar way using a power workstation as a proxy machine.

The use of RPi and Arduino could be a common playground for different actors of the world of IoT, thanks to the integration of GPIO interfaces and programmable microcontrollers in an elastic/on demand cloud scenario.

To face this different situations, we used a layered approach finding the boundary between the strictly hardware related interactions and the better place where to group common characteristics.

This ideal line is where to work in terms of virtualization on the lower side and abstraction on the upper side [3].

8 Conclusions and future works

In this work have been presented our preliminary results about the design and the implementation of an OpenCL wrapper library as GVirtuS framework plugin in order to accelerate sub-clusters of low power demanding ARM based boards using high end GPGPU devices.

We chose OpenCL as parallel programming computing model because it is independent from any kind of architectural constraints.

The most challenging result achieved is the implementation of a base tool unchaining the development of really

distributed and heterogenic hardware architectures dedicated (but not limited to) hosting HPCC middleware and software applications.

The experiments we performed demonstrate how the proposed approach is convenient.

The incredible performance results we achieved (up to x2500 in the MatrixMul benchmark), the wall clock using acceleration is less than the 1 % compared with the non-accelerated ARM board, have been affected by the computing power of the ARM side: they need for more investigation and developments.

The next step will be the development of sub-cluster made by high performance ARM based boards provided by multicore ARM 64bit CPUs with virtualization extensions and high bandwidth network interfaces.

In particular, using multicore, high memory, dual 1G ethernet ARM boards computing nodes we could split the MPI message passing communication and the GVirtuS data transfer and remote invocations on two different network fabrics increasing the scalability and better evaluating the latency of each software component.

With this kind of environment setup, we will investigate the possible performance improvements on the ARM side joined with a better scalability because of a more performing communication with both the accelerator and the input/output node.

Further and amazing research could be performed focusing on the development of a mixed x86_64/ARM general parallel file system [9].

More developments are planned on setup complete miniature HPCC provider using open source software integrating the different flowers of GPGPU virtualization and sub-cluster acceleration with the aim of next generation of real world applications [12,21].

Acknowledgments This work was supported by the fund for internal reseach projects of the University Parthenope of Napoli and by the SPACI/CNR project managed by the Department of Science and Technology—High Performance Scientific Computing Laborarty at UniParthenope (LMNCP, http://lmncp.uniparthenope.it). Portions of this effort were conducted within the Campania Region Marine and Atmosphere Monitoring and Modelling Centre (CCMMMA, http://meteo.uniparthenope.it). All the code produced under this research are or will be released as open source with GPL/LGPL license.

References

1. Abdurachmanov, D., Arya, K., Bendavid, J., Boccali, T., Cooperman, G., Dotti, A., Elmer, P., Eulisse, G., Giacomini, F., Jones, C.D., Manzali, M., Muzaffar, S.: Explorations of the viability of ARM and Xeon phi for physics processing. eprint arXiv:1311.1001,11/2013
2. Dall, C., Nieh, J.: Kvm for arm. In Proceedings of the Ottawa Linux Symposium, Ottawa, Canada (2010)
3. Di Lauro R., Lucarelli, F., Montella, R.: SIaaS-sensing instrument as a service using cloud computing to turn physical instrument into ubiquitous service. Tenth IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA). pp. 861–862, (2012)
4. Foster, I., Zhao, Y., Raicu, I., Lu, S.: 'Cloud computing and grid computing 360-degree compared'. In: Grid Computing Environments Workshop. GCE'08, pp. 1–10. IEEE, Austin (2008)
5. Giunta G., Montella, R., Laccetti, G., Isaila, F., Blas, F.J.G.: A GPU Accelerated High Performance Cloud Computing Infrastructure for Grid Computing Based Virtual Environmental Laboratory, Advances in Grid Computing, Dr. Zoran Constantinescu (Ed.), ISBN: 978-953-307-301-9, InTech (2011)
6. Giunta, G., Montella, R., Agrillo, G., Coviello, A.: GPGPU transparent virtualization component for high performance computing clouds. Euro-Par 2010-Parallel Processing, pp. 379–391. Springer, Berlin (2010)
7. Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., Ranganathan, P.: Gvim: Gpu-accelerated virtual machines. In: Proceedings of the 3rd ACM Workshop on System-Level Virtualization for HPC, p. 1724. HPCVirt 09, ACM, New York (2009)
8. Gupta A., Milojicic, D.: Evaluation of HPC applications on cloud. In Open Cirrus Summit (OCS), 2011 Sixth, pp. 22–26. IEEE, Atlanta (2011)
9. Isaila, F., Blas, F.J.G., Carretero, J., Liao, W., Choudhary, A.: A scalable message passing interface implementation of an Ad-Hoc parallel I/O system. Int. J. High Perform. Comput. Appl. **24**(2), 164–184 (2010)
10. Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M.: Science clouds: early experiences in cloud computing for scientific applications. Cloud Comput. Appl. **2008**, 825–830 (2008)
11. Lacceti G., Montella, R., Palmieri, C., Pelliccia, V.: The High Performance Internet of Things: Using GVirtuS to Share High-End GPUs with ARM Based Cluster Computing Nodes. Parallel Processing and Applied Mathematics, Springer, Berlin (2013) In press
12. Madduri, R. K., Sulakhe, P.D.D., Lacinski, L., Liu, B., Foster, I.T.: Experiences in building a next-generation sequencing analysis service using galaxy, globus online and Amazon web service. In: Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, p. 34. ACM, New York (2013)
13. Mateescu, G., Gentzsch, W., Calvin, J.R.: Hybrid computingwhere HPC meets grid and cloud computing. Futur. Gener. Comput. Syst. **27**(5), 440–453 (2011)
14. Mateusz, J., Varrette, S., Oleksiak, A., Bouvry, P.: Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In: Energy Efficiency in Large Scale Distributed Systems, pp. 182–200. Springer, Berlin, Heidelberg (2013)
15. Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Futur. Gener. Comput. Syst. **29**, 1408–1416 (2012)
16. Montella R., Agrillo, G., Mastrangelo, D., Menna, M.: A globus toolkit 4 based instrument service for environmental data acquisition and distribution. Proceedings of the Third International Workshop on Use of P2P, Grid and Agents for the Development of Content Networks, pp. 21–28. ACM, Boston (2008)
17. Montella, R., Coviello, G., Giunta, G., Laccetti, G., Isaila, F., Garcia Blas, F.J.: A general-purpose virtualization service for HPC on cloud computing: an application to GPUs. Parallel Processing and Applied Mathematics, pp. 740–749. Springer, Berlin (2012)
18. Montella R., Giunta, G., Laccetti, G:. Multidimensional environmental data resource brokering on computational grids and scientific clouds. In: Handbook of Cloud Computing, pp. 475–492. Springer, Berlin (2010)
19. Montella R., Foster, I.: Using hybrid grid/cloud computing technologies for environmental data elastic storage, processing, and

provisioning. In: Handbook of Cloud Computing, pp. 595–618. Springer, Berlin (2010)
20. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Comput. Gr. Forum **26**, 80113 (2007). doi:10.1111/j.1467-8659.2007.01012.x
21. Pham, Q., Malik, R., Foster, I., Di Lauro, R., Montella, R.: SOLE: linking research papers with science objects. In: Provenance and Annotation of Data and Processes, pp. 203–208. Springer, Berlin (2012)
22. Ravi, V.T., Becchi, M., Agrawal, G., Chakradhar, S.: Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework. In: Proceedings of the 20th International Symposium on High Performance Distributed Computing, p. 217228. HPDC 11, ACM, New York (2011)
23. Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., Sarrafzadeh, M.: Energy-aware high performance computing with graphic processing units. In Workshop on Power Aware Computing and System (2008)
24. Schmidl D., Cramer, T., Wienke, S., Terboven, C., Mller, M.S.: Assessing the performance of OpenMP programs on the intel xeon phi. In: Euro-Par 2013 Parallel Processing, pp. 547–558. Springer, Berlin (2013)
25. Shi, L., Chen, H., Sun, J.: vcuda: Gpu accelerated high performance computing in virtual machines. In: Proceedings of the 2009 IEEE IPDPS. Rome (2009)
26. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: Aview of scientific applications. In: Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, p. 416. ISPAN 09, IEEE Computer Society, Washington, DC (2009)
27. Wang, L., Tao, J., von Laszewski, G., Marten, H.: Multicores in cloud computing: research challenges for applications. JCP **5**(6), 958964 (2010)

Raffaele Montella works as an Assistant Professor, with tenure, in Computer Science, Department of Applied Science, School of Science and Technology, University 'Parthenope' of Naples, Italy since 2005. He received his degree (MSc equivalent) in (Marine) Environmental Science at the 'Parthenope' University of Naples in 1998, defending a thesis on the 'Development of a GIS system for marine applications', scoring laude and an award mention to his study career. He defended his PhD thesis on 'Environmental Modeling and Grid Computing Techniques' earning his PhD in Marine Science and Engineering at the University of Naples Federico II. The main topics of research and the scientific production are focused on tools and middleware for high performance computing, such as grid, cloud and GPUs with applications in the field of computational environmental science (multidimensional data/distributed computing for modelling).

Giulio Giunta is Professor of Scientific Computing at the Department of Science and Technology of the Parthenope University, Naples (Italy). He is directory of the Computing Center of the University and head of the Computer Science BSc Program. Prof. Giunta leads an interdisciplinary research group that investigates advanced computational science applications and scientific software production for novel computing systems. His current research focuses on numerical simulations of complex environmental systems, numerical methods for data analysis and models assessment, computer science techniques to create grid infrastructures and technologies for enabling resource sharing for operational environmental models.

Giuliano Laccetti is presently full professor of computer science at the University of Naples Federico II, in Naples, Italy. He received his Laurea degree (cum laude) in Physics from the University of Naples; his main research interests are Mathematical Software, Scientific Computing, High Performance Architecture for Scientific Computing, Distributed Computing, Grid Computing, Cloud Computing, Algorithms on emerging hybrid architectures (CPU+GPU,…). He is author (or co-author) of more than 70 papers published on refereed international Journals and Conference Proceedings. He has been involved in several EU funded Projects (EGEE, EGEE II, EGEE III; in this last case he "served" as University of Naples scientific coordinator). He has been involved also in National (EU funded) Projects as SCOPE, and, presently, RECAS; in this case, he is member of the Scientific and Management Board and he is also coordinator of the curriculum Master Degree about "Technologies for The High Performance Scientific Computing" of the University of Naples, funded by the RECAS Project itself. Presently, Giuliano Laccetti teaches Computer Progamming, and Parallel and Distributed Computing, to the University of Naples Computer Science Degree students Giuliano Laccetti is/has been member of ACM, IEEE-Computer Society, SIAM, GRIN, SIMAI, AICA.