Cross-Layer and Multi-Objective Programming Approach for Next Generation Heterogeneous Parallel Computing Systems

Project Number 688146

D4.2 – Preliminary release of integrated monitoring platform, infrastructure integration and resource management software stack

Version 2.0
5 November 2017
Final

Public Distribution

University of Stuttgart, University of York, Unparallel Innovation, The Open Group

Project Partners: **Easy Global Market, GMV, Intecs, The Open Group, University of Stuttgart, University of York, Unparallel Innovation, WINGS ICT Solutions**

PROJECT PARTNER CONTACT INFORMATION

Easy Global Market Philippe Cousin 2000 Route des Lucioles Les Algorithmes Batiment A 06901 Sophia Antipolis France Tel: +33 6804 79513 E-mail: philippe.cousin@eglobalmark.com	**GMV** José Neves Av. D. João II, Nº 43 Torre Fernão de Magalhães, 7º 1998 - 025 Lisbon Portugal Tel. +351 21 382 93 66 E-mail: jose.neves@gmv.com
Intecs Silvia Mazzini Via Umberto Forti 5 Loc. Montacchiello 56121 Pisa Italy Phone: +39 050 9657 513 E-mail: silvia.mazzini@intecs.it	**The Open Group** Scott Hansen Rond Point Schuman 6 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Stuttgart Bastian Koller Nobelstrasse 19 70569 Stuttgart Germany Tel: +49 711 68565891 E-mail: koller@hlrs.de	**University of York** Neil Audsley Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325571 E-mail: neil.audsley@cs.york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	**WINGS ICT Solutions** Panagiotis Vlacheas 336 Syggrou Avenue 17673 Athens Greece Tel: +30 211 012 5223 E-mail: panvlah@wings-ict-solutions.eu

DOCUMENT CONTROL

Version	Status	Date
0.1	Internal draft	03/04/2017
0.2	Finalized structure of the release documentation	09/05/2017
0.3	Input from all partners	20/05/2017
0.4	Internal QA version	30/05/2017
1.0	Final version	31/05/2017
2.0	Revised version incorporating EC reviewers' comments (Section 8)	05/11/2017

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The purpose of this document is to describe the preliminary release (0.1) of the system software stack of the PHANTOM platform. The components that are included in the release are the Monitoring Framework for heterogeneous hardware platforms and applications, the Resource Management service as well as a set of tools that enable the integration of reconfigurable hardware resources into a common infrastructure – the FPGA Linux system and the IP Cores Marketplace. The released tools constitute an important central part of the PHANTOM platform framework and are used to support the core platform components such as the Multi-Objective Mapper, the Deployment Manager, and other (see D2.1). Some of the components are outcomes of several successful EU projects, such as JUNIPER, EXCESS, DreamCloud, and other. In the frame of PHANTOM, a number of adaptation, extension, and improvement actions have been undertaken in order to fulfill the challenging requirements of the PHANTOM platform on monitoring, real-times, etc.

Most notably, the release includes:

- **The PHANTOM Monitoring Client –** a tiny client software running on the device and collecting performance- and energy-specific metrics. The Client was designed in the EXCESS and DreamCloud project and had a **substantial improvement** in PHANTOM (aiming to support a separation of infrastructure- and application-specific metrics, simultaneous monitoring of heterogeneous hardware platforms, possibility to specify user-specific metrics, etc.)

- **The PHANTOM Monitoring Server –** a storage and database in which the metrics from all clients are aggregated and centrally analyzed. Same as the Client, the Server is an outcome of the DreamCloud project and had a **substantial improvement** in PHANTOM (aiming to improve the configurability, extend the analytics functionality with regard to the gathered metrics values, etc.)

- **Resource management service –** a service that defines schemata (leveraging the JSON format) that describes the status of each hardware device included in the PHANTOM architecture as well as provides general scripts to manage the resource status. The schema is hosted on the web-server and database of the Monitoring Server. This is a pure PHANTOM development.

- **The PHANTOM Linux system** which manages the software components mapped to the processors of a reconfigurable device. This is responsible for passing data to and from components and for coordinating their execution. The Linux system is partially based on the outcomes of the JUNIPER project but goes far beyond their core functionality, among others – **the PHANTOM FPGA architecture** is now included, which is responsible for hosting the hardware parts of components mapped to the FPGA.

- **The PHANTOM IP Cores Marketplace** which contains several logic blocks specially designed to be used as application-specific accelerators.

This IP cores will have specific functions, commonly used mathematical algorithms (e.g. Fast Fourier transform – FFT, Finite impulse response – FIR, etc.), and image processing filters (e.g. Sobel Filter, Discrete wavelet transform – DWT), etc.).implemented in FPGA logic fabric. The PHANTOM IP cores are the pure PHANTOM development.

The release documentation in Section 2–7 gives necessary details on the technological background and PHANTOM advances for all major tools and provides necessary installation and usage guides for them. **Section 8 provides details on the implementation work that has been performed and planned for each of the released tools**.

1. INTRODUCTION

1.1 RATIONALE

The modern software applications have to struggle with a great variety of the available hardware platforms, ranging from the commodity Intel's and low-power ARM's CPUs to the accelerators like NVIDIA's GPUs, reconfigurable-logic systems like Xilinx's FPGAs or dedicated systems like Movidius' Myriad2. The selection of the most proper platform for the specific application, which has to fulfil the user-imposed functional and non-functional requirements, is a very challenging task, even without considering the required development efforts. Moreover, the challenge is getting even more complex when assuming all those hardware working in a collaborative way within the common infrastructure (which might range from the "system-on-chip" to the cluster-like distributed systems). The term "Cloud" has become common even for such restrictive in terms of hardware domains like embedded systems.

The component-based PHANTOM programming model (see D1.2 and D2.1) allows the PHANTOM application to be deployed and executed on several hardware devices, which might be of different types and nature. The PHANTOM project carries out the mission to provide a platform that allows the components constituting the application (within the specified control- and data-flow) to be executed in heterogeneous, parallel, and distributed hardware environments (see Figure 1) without any hardware-specific adaptation of the source code, needed to be done by the developers.

Operation of such a heterogeneous infrastructure imposes several challenges, in particular:

- Monitoring of the infrastructure components status (e.g. utilization of the CPU and memory, cache misses, etc.)
- Management of the infrastructure components (availability for allocation of new application components, operational power mode, etc.)
- Availability of a run-time environment for the application execution (with the major challenge lying on reconfiguration-enabled hardware)

In order to succeed in the intention to solve these challenges, the PHANTOM platform requires middleware for monitoring, management, and integration into a common infrastructure of the heterogeneous hardware resources, tracking and managing their status, integration of the heterogeneous components into a common infrastructure. The corresponding middleware has been developed in the frame of PHANTOM by the work package WP4, see next section (Section 1.2).

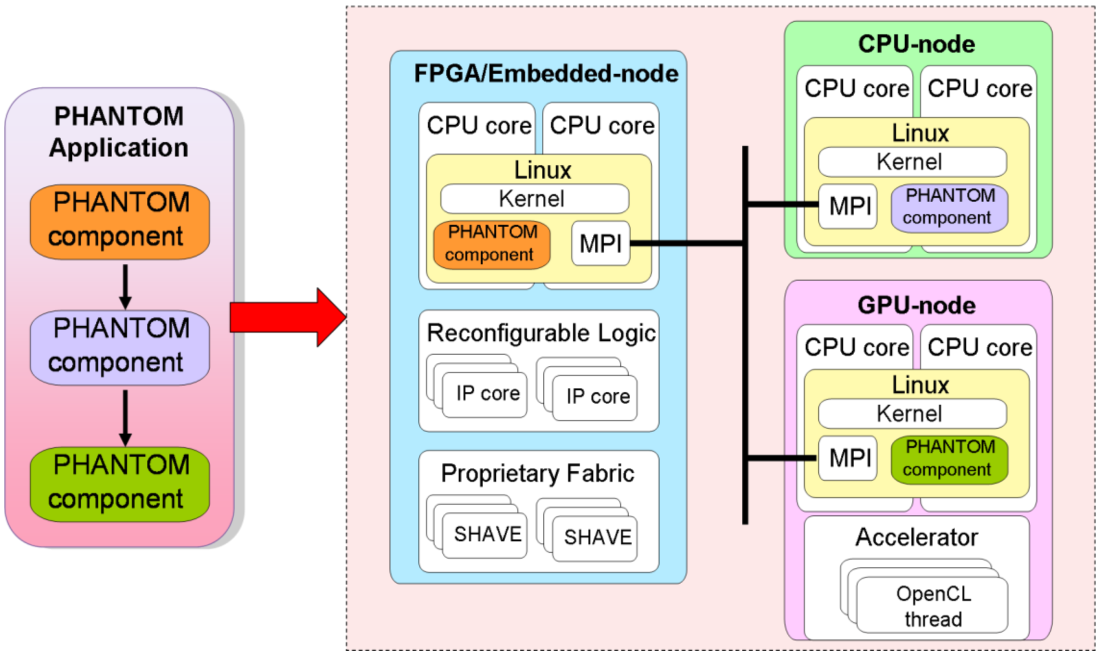


Figure 1: PHANTOM application execution in heterogeneous hardware environment

1.2 Release Information and Scope

The current (preliminary) release is comprised of the following components:

- Monitoring and resource management framework

  The framework offers a set of tools and services for collection and centralized analysis of a broad scope of metrics that characterize the state of the hardware resources (utilization, overall energy consumption, etc.) as well as of the applications (performance, application-specific energy consumption, etc.) over the time. The framework is used by the PHANTOM Multi-Objective Scheduler (MOM, for details please refer to deliverable D2.1) for elaborating the optimal resource allocation strategies for the applications as well as by the end-users of the applications in order to evaluate the performance characteristics and optimize the infrastructure utilization. The framework also hosts a resource management service that fosters the collection, publishing, and setting the status of the controlled infrastructure devices.

  The following components are released as parts of the monitoring and resource management framework:

  - **Monitoring client** – a tiny client software running on the device and collecting performance- and energy-specific metrics
  - **Monitoring server** – a storage and database in which the metrics from all clients are aggregated and centrally analysed

    - **Resource management service** – a service, which is currently hosted by the monitoring server, that aims to track and control the status of all hardware devices included in the infrastructure

- Reconfigurable applications run-time environment

  This environment provides technologies for executing applications on the hardware constituted by reconfigurable (FPGA) logic devices. The environment includes following components:

    - **The PHANTOM Linux system** which manages the software components mapped to the processors of a reconfigurable device. This is responsible for passing data to and from components and for coordinating their execution. It also implements the process isolation and monitoring requirements of the platform.

      The system also includes **the PHANTOM FPGA architecture** which is responsible for hosting the hardware parts of components mapped to the FPGA. This architecture also implements further isolation and monitoring features. The architecture is automatically assembled and built based on the input from the PHANTOM Multi-Objective Mapper.

    - **The PHANTOM IP Cores Marketplace** which contains several logic blocks specially designed to be used as application-specific accelerators. This IP cores will have specific functions, commonly used mathematical algorithms (e.g. Fast Fourier transform – FFT, Finite impulse response – FIR, etc.), and image processing filters (e.g. Sobel Filter, Discrete wavelet transform – DWT), etc.).implemented in FPGA logic fabric.

The release includes tarballs with the source code and installation scripts as well as all necessary documentation and user guides. All released components are provided under open source licenses (see Table 1) and are available for free downloading on the PHANTOM's public hosting platform GitHub (https://github.com/phantom-platform).

Table 1: Released PHANTOM components

Component	Version	License	Size	Link
Monitoring and Resource Management Framework	0.1	Apache 2.0	173 KB	https://github.com/PHANTOM-Platform/Monitoring/archive/0.1.zip
PHANTOM Linux System	0.1	GPLv3	5,2 MB	https://github.com/PHANTOM-Platform/PHANTOM-FPGA-Linux/archive/0.1.zip

PHANTOM IP core marketplace	0.1	Apache 2.0	10 MB	https://github.com/PHANTOM-Platform/PHANTOM-IP-Core-Marketplace/archive/0.1.zip

1.3 Structure of This Document

The remainder of the deliverable is organized as follows. Section 2 gives an overview of the heterogeneous hardware resources available to the project consortium for the pilot prototyping of tools and applications. Sections 3-7 describe the released components. Section highlights the innovation aspects of the released tools and identifies upcoming actions plan. Section 9 provides major findings and conclusions.

2. PHANTOM HETEROGENEOUS PARALLEL INFRASTRUCTURE TESTBED

In order to evaluate the functionality of the PHANTOM system software components as well as to provide a heterogeneous parallel infrastructure testbed for the deployment of the developed PHANTOM application components, the PHANTOM project partners have granted access to some hardware resources that are available on their sites and also that are required by the pilot applications (see D1.1). The resource access is only granted for the PHANTOM project consortium under the policies of the respective partners. Below an overview of the most typical resources is given.

2.1 STANDARD CPU-BASED DEVICES

The devices of this category constitute the most common on the market general-purpose x86, amd64 and ARM hardware.

2.1.1 Server-Solutions

The partner USTUTT-HLRS provides access to its cluster testbed, which was set up in the frame of the previous EU-projects EXCESS and DreamCloud. The cluster was used as a testbed for evaluating energy-efficiency of High-Performance Computing applications.

The cluster (see Figure 2) consists of 3 compute nodes and a common networking file system, interconnected with the Infiniband network. Each node is built with NUMA (Non-Uniform Memory Access) technology with the characteristics summarized in Table 2.

Table 2: HPC cluster hardware specification

Amount	Hardware components (node01 and node02)
2	Intel Xeon CPU: E5-2690 v2 (Ivy Bridge) - 10 cores; 25MB L3 Smart Cache; 4 Mem. Ch. DDR3 1866 MHz = 59.7 GB/s
8	HP Memory 4 GB DDR3 MFG 708633-B21 - 1866 MHz PC3-14900
1	GPU: Tesla K40c - GPU Clock: 745 MHz; Shading Units: 2880; GDDR5 12288 MB; 288 GB/s; PCIe3.0 8GT/s x16
1	Hard disk: 500GB WD5003AZEX Black
1	SSD: 128GB Vertex450
	Hardware components (node03)
2	Intel Xeon CPU: E5-2680 v3 (Haswell) - 12 cores; 30MB L3 Smart Cache; 4 Mem. Ch. DDR4 2133 MHz = 68 GB/s
8	SAMSUNG DRAM 16GB Samsung DDR4-M393A2G40DB0-CPB- 2133 MHz
1	Hard disk: 500GB WD5003AZEX Black
1	SSD: 240GB Vertex460A

All 3 nodes are interconnected by the Infiniband network. A shared file system (NFS over Infiniband) is available. There is a common front-end for accessing the cluster and management of jobs on Nodes 01-03.

A dedicated power measurement system is installed on each of the nodes in order to collect power and energy infrastructure profiles.

The cluster will be used for the demanding in terms of performance applications without hard real-time requirements, such as the simulation application of the USTUTT-HLRS partner or the surveillance one of GMV.

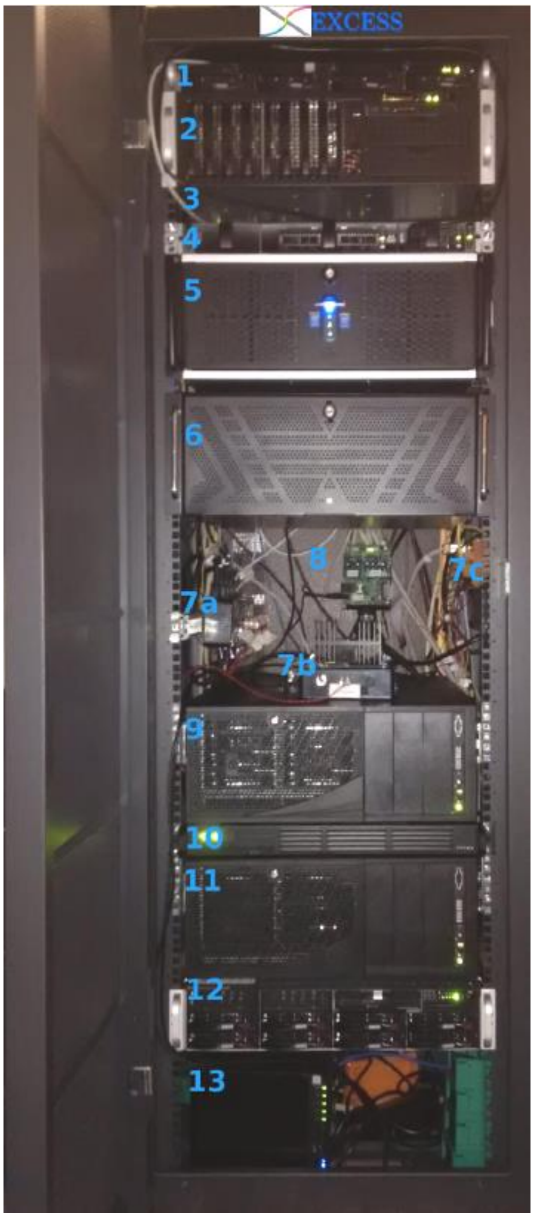
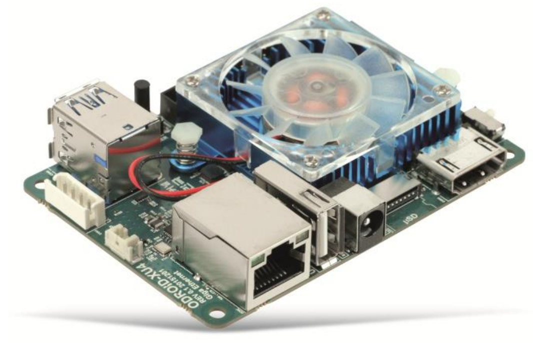


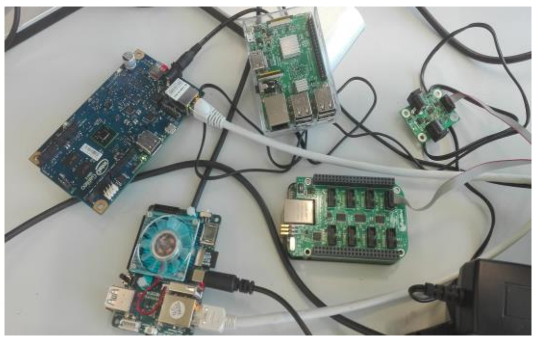
Figure 2: EXCESS cluster of USTUTT-HLRS

2.1.2 Low-Power and Mobile Devices

For evaluation purposes, the consortium partners will use a range of portable ARM-based platforms such as ODROID-XU4 (Figure 3). ODROID is a one-board SAMSUNG Exynos5 Octa CPU with 2 GB DDR3. The CPU consists of 4 more powerful Cortex-A15 and 4 less powerful Cortex-A7 cores. A dynamic load balancing system of the board decides on which core every thread should be executed. This board is planned to be used for all 3 pilot PHANTOM use cases. The MOM (Multi-Objective Scheduler) will be able to instruct the native scheduler of the board about the better allocation properties and the impact of the static vs. the native dynamic scheduling will be evaluated. There are a variety of other boards that will be evaluated, including Raspberry PI-3 (1,2GHz, 4x Cortex-A53v8), Intel Galileo Gen 2, and some other.



a)



b)

Figure 3: Low-power hardware testbed: a) ODROID-XU4 board of USTUTT-HLRS, b) mini-embedded cluster made of Raspberry Pi-3, Galileo, and Odroid

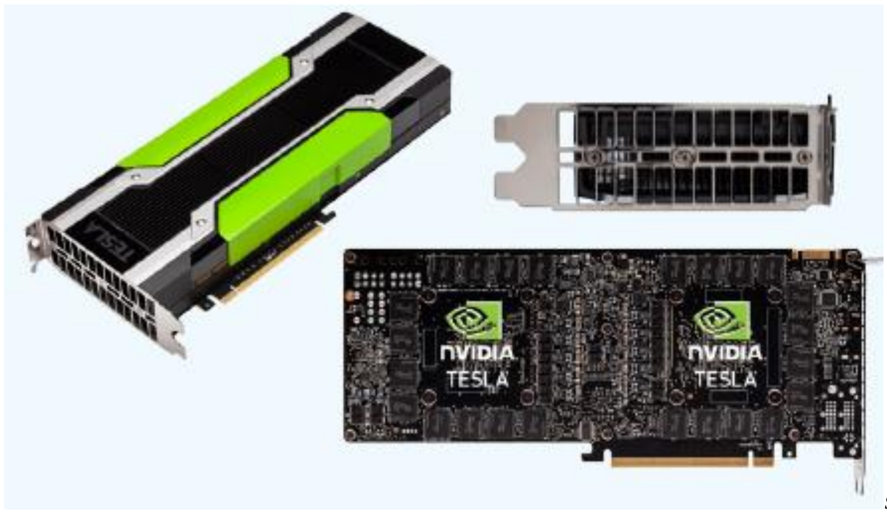
2.2 ACCELERATION DEVICES

2.2.1 GPU-Based Accelerators

The GPU-based accelerated compute resources are enjoying an increasing popularity in many infrastructure communities from embedded systems to high-performance computing. Piz Daint – the Europe's fastest supercomputer[1], located in Switzerland, is made up of Tesla P100 GPUs from NVIDIA.

In the project, we are going to use the GPU facilities that are provided by the EXCESS cluster, described in Section 2.1. Its Node01 includes an NVIDIA Tesla K40 graphic card, and node03 – an NVIDIA Tesla K80 one. Tesla K40 can reach the performance of up to 1.43 TFLOPS on 2.880 CUDA cores and provide 12 GB of the local memory (GDDR5). Tesla K80 (Figure 4) consists of 2 Tesla K40 and offers the nearly double performance (Figure 4).

[1] https://www.top500.org/lists/2016/11/



source: NVIDIA

Figure 4: NVIDIA Tesla K80 GPU of USTUTT-HLRS

The NVIDIA GPUs will be used for the massively-parallel and data-centric (thousands of independent threads working on shared data sets) parts of the computationally-intensive parts of the USTUTT-HLRS and GMV workflows.

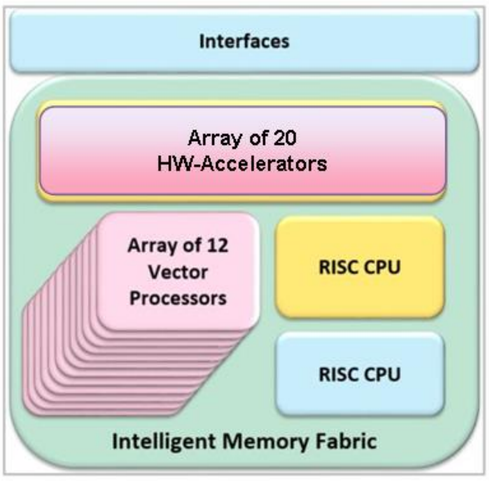
2.2.2 Specialised Acceleration Devices

Myriad 2[2] (Figure 5a) is a specialised SoC (system-on-chip) hardware platform of a company called Movidius (currently owned by Intel) that provides solutions for mobile image processing, for which the CPU- and GPU-based solutions prove ineffective (due to the mixed image rendering and processing workflows as well as the very high requirements on portability and energy efficiency).

The Myriad 2's SoC architecture (Figure 5b) provides on the same silicon alongside with low-power general-purpose SPARC Leon cores (one for scheduling within the SoC and the other for running the user code within a real-time operating system) a set of 20 hard-coded (ASICS) for HW-acceleration of some typical image processing operations as well as 12 SHAVE 128-bit SIMD vector units. A common memory for all SoC resources of 2 MB is provided by the board that functions as a hybrid L3 cache. Additionally, up to 1 GB of external DRAM memory can be supported by the board.

[2] http://www.tomshardware.com/news/movidiud-myriad2-vpu-vision-processing-vr,30850.html

a)


Interfaces
Array of 20 HW-Accelerators
Array of 12 Vector Processors
RISC CPU
RISC CPU
Intelligent Memory Fabric

b)

Figure 5: Movidius' Myriad 2 platform of USTUTT-HLRS: a) board view, b) architecture

This board is planned to be used by all use cases. The advantages of the heterogeneous SoC components, including the acceleration parts, will be leveraged by the PHANTOM platform components to optimise the application performance and power consumption characteristics.

More details on the Myriad2 board, including the software stack information, are provided in Appendix 2.

2.3 Reconfigurable Devices

These devices include the reprogrammable-logic hardware that allows the hardware-based application development.

2.3.1 FPGAs

An FPGA (Field Programmable Gate Array) is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks. In this way one can devise special purpose functional units that may be very efficient for this limited task. Moreover, it is possible to configure a multiple of these units on an FPGA that work in parallel. So, potentially, FPGAs may be good candidates for the acceleration of certain applications. Because of their versatility, it is difficult to specify where they will be most useful. In general, though, they are not used for "heavy" 64-bit precision floating-point arithmetic. Excellent results have been reported in searching, pattern matching, signal and image-processing, encryption, etc. The clock cycle of FPGAs is low as compared to that of present CPUs: 100-550 MHz which means that they are very power efficient. All vendors provide runtime environments and drivers that work with Linux as well as Windows.

Traditionally, FPGAs are configured by means of a hardware description language (HDL), like VHDL or Verilog. This is very cumbersome for the average programmer as one not only has to explicitly define such details as the placement of the configured devices but also the width of the operands to be operated on, etc. Also, the integration normally uses a custom interface, which depends on the application and the algorithms being implemented on the FPGA. This obligates the software designers to know the hardware in detail, in order to code the drivers and applications to exploit the accelerators. This problem has been recognized by FPGA-based vendors and a large variety of programming tools and SDKs have come into existence. Unfortunately, they differ enormously in approach and the resulting programs are far from compatible.

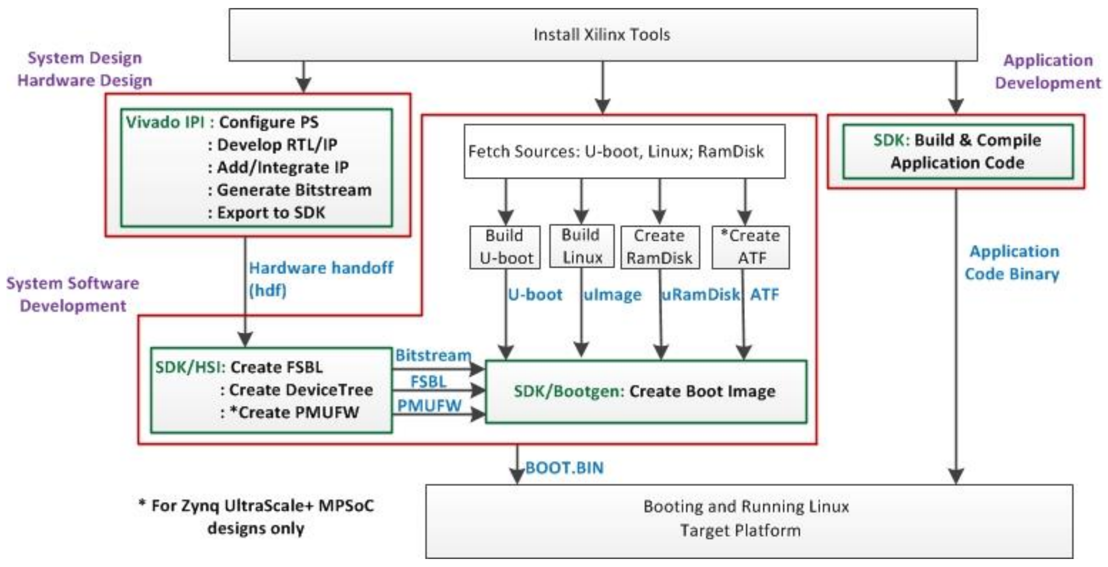


Figure 6: Xilinx Design Flow

The Xilinx workflow, shown in Figure 6, has a slow learning curve and can be difficult to comprehend, since it encompasses many steps that are very complex by themselves and uses several different tools for HDL generation (Vivado High-Level Synthesis), design integration (Vivado, ISE), software development (Xilinx SDK) and Linux integration (Xilinx SDX, Petalinux tools). The PHANTOM platform will ease the process of integrating hardware accelerators based on FPGA IP cores with user applications without the user having to understand about heterogeneous hardware accelerators, HDL coding, FPGA synthesis and design integration with IP cores.

2.3.2 Tightly-Coupled Heterogeneous Boards

The FPGA boards targeted by the PHANTOM platform are all Zynq-class boards from Xilinx (Figure 7). Zynq FPGAs are comprised of an ARM system-on-chip with attached reconfigurable logic. The ARM SoCs are 32-bit or 64-bit multiprocessor systems, clocked from 800Mhz upwards, and consisting of currently 2 or 4 cores. They contain full MMUs and are capable of running a standard Linux mainline kernel. The FPGA logic is tightly coupled to the CPUs, allowing for very high-speed data transfer. Therefore, the CPUs can be used for varied software tasks, and to invoke custom hardware designs on the reconfigurable logic for high-volume, low-latency data processing.

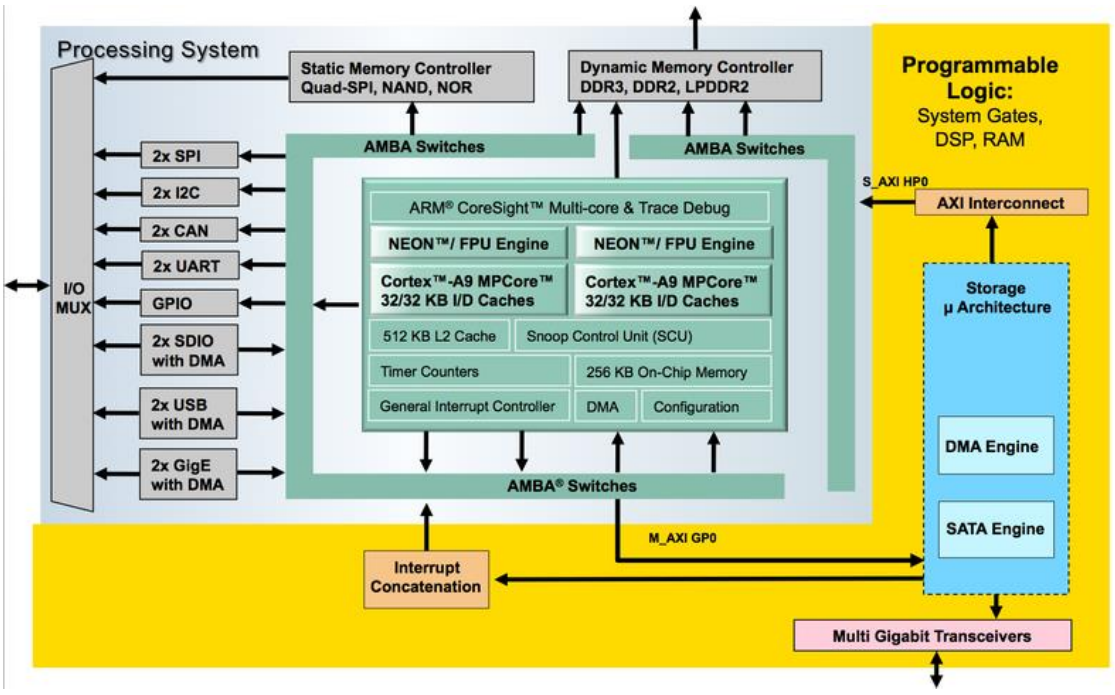


Figure 7: The Zynq reference block diagram (from www.xilinx.com)

In the context of the PHANTOM project, this means that FPGA targets are actually an FPGA with tightly-coupled CPU SoC. PHANTOM does not target a “raw” FPGA but requires a CPU-host instead. This allows FPGA targets to host both the software and hardware parts of IP cores.

3. Released Component: Monitoring Client

3.1 Functionality

3.1.1 Overview

Monitoring Client is a part of the PHANTOM monitoring framework (see Figure 8) that is installed on each of the controlled devices of the infrastructure testbed in order to collect the values of the monitored metrics. With this aim, the client is implemented as a lightweight software stack that is running as a separate process running in the background of the operating system (Linux). At every time interval, which can be defined by the user (in the range from several milliseconds to minutes), the Client collects the data from the controlled device and transmits them to the Monitoring Server (see Section 4).

In order to minimize the impact on the performance of the monitored device, the client relies heavily on the information obtained from the hardware counters such as PAPI, RAPL, etc. The evaluation of the Monitoring Client overhead on the standard laptop (dual-core AMD processor) showed **the additional CPU usage of less than 0.1% and the memory utilization of about 4,5 MB**, which is acceptable even for the embedded devices.

As a further optimization, the client offers a possibility to adjust the interval of data collections (at the millisecond range), which can be done by the user manually or by the other components of the PHANTOM platform.

The Client allows collection of 3 categories of metrics:

- Hardware performance metrics
- Hardware power consumption metrics
- Custom application-specific metrics

The first two categories are collected either for the whole device (infrastructure-level) or for a particular application run (application-level). The application-level monitoring constitutes a fraction of the monitoring data that are specifically consumed by the application; it only happens if the user has registered the application run by the Monitoring Framework (Figure 9), for which a special registration service is provided by the Monitoring Framework (see details in the usage examples below). The detailed list of the default metrics that can be collected by the Client is provided in D2.1 and in Appendix 1. The Monitoring Client plug-ins that perform the metrics collection are described in Appendix 2.

The custom metrics have been introduced in order to allow the application developer to store any application-specific information, such as the number of simulation steps performed, their duration, etc.

The power consumption is obtained either from the dedicated hardware counters (as provided by some server CPUs like Intel Haswell) or from an external power measurement systems, such as ACME (Figure 10).

As the data transmission between the client and the server is performed through the network, the client implements a buffering mechanism in order to minimize the amount of the network transactions, thus not overloading the network traffic.

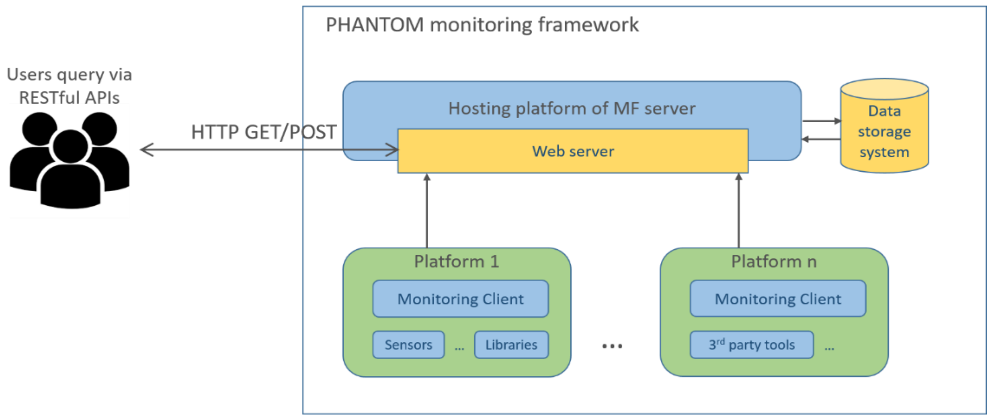


Figure 8: Architecture of PHANTOM Monitoring Framework

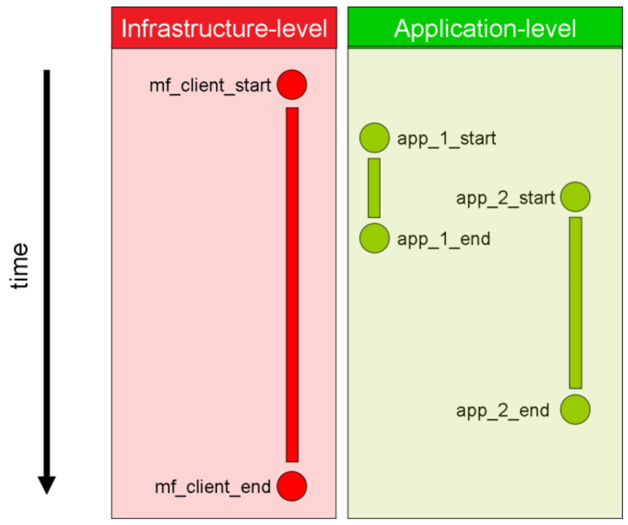


Figure 9: Infrastructure- and application-level monitoring with the Monitoring Client

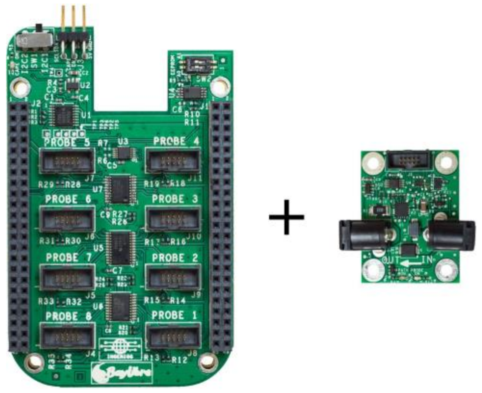


Figure 10: External power measurement board from ACME

3.1.2 PHANTOM Advances

The Monitoring Client was initially provided as a part of ATOM – a monitoring framework solution developed in the frame of EXCESS and DreamCloud projects. In PHANTOM, the Client was majorly refactored with the aim to support the distribution of the monitored metrics to infrastructure- and application levels, facilitate inclusion of new infrastructure devices such as FPGAs, improve the usability and configurability on the user-side.

A short list of the major extensions is as follows:

- Support of buffering in order to decrease the network interception that is introduced by transmitting the monitoring data from the Clients to the Server.
- Developed libraries and corresponding APIs to enable the application-specific metrics definition and gathering.
- Enabled seamless support of external measurement systems along with the counter-based hardware capabilities.
- Introduced dynamic configuration of metrics to be monitored, which implies the user can define the metrics at the application run-time via an intuitive user interface.

3.2 DEPENDENCIES AND INSTALLATION

As clarified above, plug-ins' implementation depends on various libraries, hardware availabilities or 3rd-party models, like PAPI, RAPL counters, Linux monitoring sensors library, and etc. The Monitoring Client's dependencies are summarized in Table 3.

Table 3: Monitoring Client dependencies

Component	Version	Link
PAPI-C	5.4.0	http://icl.cs.utk.edu/papi/
CURL	7.37.0	http://curl.haxx.se/download/
Apache APR	1.5.1	https://apr.apache.org/
Apache APR Utils	1.5.3	https://apr.apache.org/
Nvidia GDK	352.55	https://developer.nvidia.com/gpu-deployment-kit/
bison	2.3	http://ftp.gnu.org/gnu/bison/
flex	2.5.33	http://prdownloads.sourceforge.net/flex/
sensors	3.4.0	https://fossies.org/linux/misc/
m4	1.4.17	https://ftp.gnu.org/gnu/m4
libiio	1.0	https://github.com/analogdevicesinc/libiio.git
hwloc	1.11.2	https://www.open-mpi.org/software/hwloc/v1.11/downloads/
EXCESS queue	release/0.1.0	https://github.com/excess-project/data-structures-library.git

To ease the process of environment setup, the release provides a bash script in the source code repository, which downloads all dependencies and installs them locally in the project directory. Thus, the client setup and compiling is performed in a sandbox without affecting the current status of the operating system. After the prerequisites installation, the PHANTOM monitoring client can be compiled, built, and installed by using the provided Makefile.

Following commands (Listing 1) concludes the preparation and installation process, which results in new directories named as /bin and /dist, with the later one holding the built executable binaries and all required libraries. Please note that it is tested with GNU (version 4.9.2) compiler on the HPC cluster.

Listing 1: Monitoring Client setup

```
$ git clone https://github.com/phantom-monitoring-framework/phantom_monitoring_client
$ cd  phantom_monitoring_client
$ ./setup.sh
```

If the added device is new, i.e. has not been registered by the local instance of the Monitoring Server yet, then the user should define a unique name for the device (e.g. "my_new_device") and provide it as "platform_id" option in the configuration file *.src/mf_config.ini*. After that, the monitoring client installation can be completed as follows (Listing 2):

Listing 2: Monitoring Client final installation steps

```
$ make clean-all
$ make all
$ make install
```

The newly installed Monitoring Client configuration can be specified in a special configuration file named *mf_config.ini*, in which the location (URI) of the Monitoring Server can be defined, the plug-ins can be activated/deactivated, plug-in parameters specified, the monitoring update time intervals set, etc. (see example in Listing 3).

Listing 3: Example of Monitoring Client configuration

```
mf_config.ini

[generic]
server = http://localhost:3033/v1
...

[plugins]
mf_plugin_Board_power = on
mf_plugin_CPU_perf = off
…

[timings]
default                = 1000000000ns
update_configuration  = 360s
mf_plugin_Board_power = 1000000000ns
…

[mf_plugin_Board_power]
ACME_BOARD_NAME = baylibre-acme.local
device0:current = on
device0:power = on
device0:vshunt = off
```

```
device0:vbus = off
…
```

3.3 USAGE EXAMPLES

Before starting the Monitoring Client instance on the newly configured device, it should be ensured that the local instance of the Monitoring Server is properly installed and running (see Section 4 for details).

Infrastructure-level monitoring

The infrastructure-level monitoring of the new device is enabled by the command (Listing 4), in which the option "*:platform_id*" corresponds to the device id that was selected at the previous step (installation):

Listing 4: Registration of a new platform by Monitoring Client

```
$ curl -H "Content-Type: application/json" -XPUT
localhost:3033/v1/phantom_mf/workflows/infrastructure -d
'{"application":"infrastructure","author":"Henry
Schmidt","optimization":"Time","tasks":[{"name":":platform_id","exec":"mf_client","cores_nr": "1"}]}'
```

The following setting is used to provide the Monitoring Client with some details on the monitored device architecture, which are necessary for a proper monitoring (Listing 5):

Listing 5: Specification of new platform settings by Monitoring Client

```
$ curl -H "Content-Type: application/json" -XPUT localhost:3033/v1/phantom_rm/configs/:platform_id -
d
'{"parameters":{"MAX_CPU_POWER":"24.5","MIN_CPU_POWER":"6.0","MEMORY_POWER":"2.016","L2
CACHE_MISS_LATENCY":"59.80","L2CACHE_LINE_SIZE":"128","E_DISK_R_PER_KB":"0.0556","E_DISK_W
_PER_KB":"0.0438","E_NET_SND_PER_KB":"0.14256387","E_NET_RCV_PER_KB":"0.24133936"}}'
```

The Client can now be started with a special shell script (Listing 6):

Listing 6: Starting Monitoring Client instance

```
$ cd scripts
$ ./start.sh
```

The following query is used to discover the individual ids of the launched Client instance:

Listing 7: Obtaining the list of launched Monitoring Client instances

```
$ curl -XGET localhost:3033/v1/phantom_mf/experiments
```

Those ids can be used to query the monitoring data, e.g. the CPU usage or the energy consumption (Listing 8):

Listing 8: Requesting the CPU usage and energy consumption metrics values

```
$ curl -XGET 'localhost:3033/v1/phantom_mf/statistics/infrastructure/excess_node01/AVtmz-r3arTJxoIfKFpP?metric=CPU_usage_rate&from=2017-04-13T12:17:00.000&to=2017-04-13T12:18:00.000'
$ curl -XGET 'localhost:3033/v1/phantom_mf/statistics/infrastructure/excess_node01/AVtmz-r3arTJxoIfKFpP?metric=device0:power&from=2017-04-13T12:17:00.000&to=2017-04-13T12:18:00.000'
```

Application-level monitoring

Following pseudo-code (Listing 9) gives a brief impact on how to use the client APIs for the C language. If all preconditions could be satisfied by the target platform, users are able to gather metrics about CPU, memory, disk I/O and power consumptions of various devices for the running code block. In addition to these predefined metrics, users are allowed to send also user-defined metrics. Taking the given source code as an example (Listing 9), the user-defined metrics are execution duration of each loop, and the number of loops after the simulation is finished. There are no limitations about the data type of the user-defined metrics, however, to realize this property, users are required to convert the metrics value to a string before calling the corresponding API.

Listing 9: Example of HPC simulation application with monitoring APIs integrated

```
#include ...
extern "C" {#include "../src/mf_api.h"}
...
int main()
{
/* initialization of various application elements, and parameters */
    ...
    initNetwork("TestNet", &netparams, 8, 6);
    initBranches(&(*branches), netparams);
    initVertexes(&(*vertexes), netparams);
    float integrationStep = 0.001; // simulation steps
    int nrLoops = 5000; // simulation loops

/* MONITORING START */
    metrics m_resources;
    m_resources.num_metrics = 3;
    m_resources.local_data_storage = 1;
    m_resources.sampling_interval[0] = 1000; // 1000 ms
    strcpy(m_resources.metrics_names[0], "resources_usage");
    m_resources.sampling_interval[1] = 1000; // 1000 ms
    strcpy(m_resources.metrics_names[1], "disk_io");
    m_resources.sampling_interval[2] = 1000; // 1000 ms
    strcpy(m_resources.metrics_names[2], "power");
    char *datapath = mf_start("141.58.0.8:3033", "node01", &m_resources);

/* simulation process */
    for (int n = 0; n < nrLoops; n++) {
            auto begin_time = std::chrono::high_resolution_clock::now();
            simulation_loop(&(*branches), &(*vertexes), netparams, n,
                        integrationStep);
            auto end_time = std::chrono::high_resolution_clock::now();
            std::chrono::duration<double, std::milli> duration = end_time-begin_time;
    /* MONITORING USER-DEFINED METRICS → duration of each loop */
            char metric_value[8] = {'\0'};
            sprintf(metric_value, "%f", duration);
            mf_user_metric("duration", metric_value);
    }

/* MONITORING END */
    mf_end();

/* MONITORING USER-DEFINED METRICS →total nr. of completed loops */
    char metric_value[8] = {'\0'};
    sprintf(metric_value, "%d", nrLoops);
    mf_user_metric("nrLoops", metric_value);

/* MONITORING SEND */
    char *experiment_id = mf_send("141.58.0.8:3033", "dummy", "t1", "node01");
    printf("\n> experiment_id is %s\n", experiment_id);
    cout << "Simulation finished";
    return 0;
}
```

After the execution of the HPC simulation application, a unique execution ID will be displayed as an output in the terminal. Thus we could retrieve all sampled metrics and associated simple statistics via the server provided RESTful APIs (cf. D2.1). Here only the user-defined metrics statistics are shown as follows:

Listing 10: Requesting the application-specific metrics values

```
$ curl
141.58.0.8:3033/v1/phantom_mf/statistics/dummy/t1/Avskow_O7rO13ZBQKWc0?metric=duration

$ [{"workflow":{"href":"http://fe.excess-
project.eu:3033/v1/mf/workflows/dummy"},"metric":"duration","statistics":{"count":5000,"min":0,"ma
x":1,"avg":0.0052,"sum":26,"sum_of_squares":26,"variance":0.00517296,"std_deviation":0.0719232924
7191065,"std_deviation_bounds":{"upper":0.1490465849438213,"lower":-
0.1386465849438213}},"min":{"TaskID":"t1","type":"user_defined","host":"platform_default","local_ti
mestamp":"2017-03-31T15:51:45.883","duration":0.474054,"server_timestamp":"2017-03-
31T15:51:58.515"},"max":{"TaskID":"t1","type":"user_defined","host":"platform_default","local_timest
amp":"2017-03-31T15:51:49.204","duration":1.151286,"server_timestamp":"2017-03-
31T15:52:13.521"}}]

$ curl
141.58.0.8:3033/v1/phantom_mf/statistics/dummy/t1/AVskow_O7rO13ZBQKWc0?metric=nrLoops

$ [{"workflow":{"href":"http://fe.excess-
project.eu:3033/v1/mf/workflows/dummy"},"metric":"nrLoops","statistics":{"count":1,"min":5000,"max
":5000,"avg":5000,"sum":5000,"sum_of_squares":25000000,"variance":0,"std_deviation":0,"std_deviati
on_bounds":{"upper":5000,"lower":5000}},"min":{"TaskID":"t1","type":"user_defined","host":"platform
_default","local_timestamp":"2017-03-31T15:51:57.892","nrLoops":5000,"server_timestamp":"2017-
03-
31T15:52:44.410"},"max":{"TaskID":"t1","type":"user_defined","host":"platform_default","local_timest
amp":"2017-03-31T15:51:57.892","nrLoops":5000,"server_timestamp":"2017-03-31T15:52:44.410"}}]
```

3.4 UPCOMING ACTIONS

The following major actions will be performed by the final release:

Infrastructure support. The target infrastructure for PHANTOM would be reconfigurable and heterogeneous, including CPU, GPU, embedded system and FPGA-based hardware. However, with respect to FPGA-based infrastructure, currently, no specific plug-ins and metrics in terms of performance and power consumption are supported. We will devote more time and effort for the specific platform and meet the demands of use cases before the final release.

Efficiency and overhead evaluation. The monitoring sampling frequency is required to be fine-grained, approaching approximately microseconds range in some extreme circumstances. Thus we will need to analyze the client-server communication's efficiency and overhead in order to ensure a high scalability and a minimum overhead to the system's performance.

Improved documentation. We will provide monitoring handbooks for the major hardware platforms that are supported by the Monitoring Framework. Such a handbook will contain typical usage strategies of the monitoring components, getting-started guides for the end-users as well as troubleshooting instructions.

4. RELEASED COMPONENT: MONITORING SERVER

4.1 FUNCTIONALITY

4.1.1 Overview

The PHANTOM monitoring server is composed of two components: a web server and a data storage system. The web server provides various functionalities for data query and data analysis via RESTful APIs with documents in JSON format. The server's URL is "localhost:3033" by default.

The monitoring server receives data from the all registered Monitoring Clients (see Section 3) via RESTful interfaces. The server is implemented using Node.js web server suite and connects to the Elasticsearch database to store and access metric data.

4.1.2 PHANTOM Advances

The PHANTOM Monitoring Server is based on the previously available ATOM solution, which has been extended to support the requirements that are specific for the PHANTOM platform. Some of the major extensions are as follows:

- Support of the metric data buffering in order to reduce the network traffic overhead between the Clients and the Server
- Advanced analytical functions for the collected data analysis, as required by the Multiobjective Mapper (MOM)

4.2 DEPENDENCIES AND INSTALLATION

The Monitoring Server's dependencies are summarized in Table 4.

Table 4: Monitoring Server dependencies

Component	Version	Link
Elasticsearch	1.4.4	https://www.elastic.co/products/elasticsearch
Node.js	0.9	https://apr.apache.org/
npm	1.3.6	https://www.npmjs.com/

To install all the prerequisites, a setup script is provided. The following commands are used for this (Listing 11).

Listing 11: Monitoring Server installation scripts

```
$ https://github.com/phantom-monitoring-framework/phantom_monitoring_server.git
$ cd phantom_monitoring_server
$ ./setup.sh
```

The PHANTOM monitoring server is already deployed on one node of the HPC cluster, being publicly accessible via the IP address 141.58.0.8 with port 3033. With the created daemon script, which is kept in /etc/init.d, the server can be controlled easily as a Linux service by the following commands (Listing 12)

Listing 12: Monitoring Server management scripts

```
$ sudo service phantom_server start
$ sudo service phantom_server stop
$ sudo service phantom_server restart
$ sudo service phantom_server status
```

To simplify the process of environment setup, the release provides a bash script in the source code repository, which downloads all dependencies and installs them locally in the project directory. After the prerequisites installation, the PHANTOM monitoring server can be compiled, built, and installed by using the provided Makefile. Further installation instructions are provided at https://github.com/phantom-monitoring-framework/phantom_monitoring_server

4.3 USAGE EXAMPLES

The PHANTOM Monitoring Server can be reached by means of RESTful GET/PUT requests in order to obtain the information/statistics about the controlled hardware devices and applications. Listing 13 shows some basic queries that are often used by the customers (the platform's MOM or the end-users).

Listing 13: Basic queries to Monitoring Server

```
# APPLICATIONS/WORKFLOWS

GET  /v1/phantom_mf/workflows

GET  /v1/phantom_mf/workflows/:application_id

PUT  /v1/phantom_mf/workflows/:application_id -d '{...}'

# EXPERIMENTS

GET  /v1/phantom_mf/experiments

GET  /v1/phantom_mf/experiments/:execution_id?workflow=:application_id

POST /v1/phantom_mf/experiments/:application_id -d '{...}'

# METRICS

GET  /v1/phantom_mf/metrics/:application_id/:task_id/:execution_id

POST /v1/phantom_mf/metrics -d '{...}'
POST /v1/phantom_mf/metrics/:application_id/:task_id/:execution_id -d '{...}'

# PROFILES

GET /v1/phantom_mf/profiles/:application_id

GET /v1/phantom_mf/profiles/:application_id/:task_id

GET /v1/phantom_mf/profiles/:application_id/:task_id/:execution_id

GET /v1/phantom_mf/profiles/:application_id/:task_id/:execution_id?from=...&to=...

# RUNTIME

GET /v1/phantom_mf/runtime/:application_id/:execution_id

GET /v1/phantom_mf/runtime/:application_id/:task_id/:execution_id

# STATISTICS

GET /v1/phantom_mf/statistics/:application_id?metric=...

GET /v1/phantom_mf/statistics/:application_id?metric=...&host=...

GET /v1/phantom_mf/statistics/:application_id?metric=...&from=...&to=...

GET /v1/phantom_mf/statistics/:application_id?metric=...&host=...&from=...&to=...

GET /v1/phantom_mf/statistics/:application_id/:execution_id?metric=...

GET /v1/phantom_mf/statistics/:application_id/:execution_id?metric=...&host=...

GET /v1/phantom_mf/statistics/:application_id/:execution_id?metric=...&from=...&to=...

GET /v1/phantom_mf/statistics/:application_id/:execution_id?metric=...&host=...&from=...&to=...

GET /v1/phantom_mf/statistics/:application_id/:task_id/:execution_id?metric=...

GET /v1/phantom_mf/statistics/:application_id/:task_id/:execution_id?metric=...&host=...

GET /v1/phantom_mf/statistics/:application_id/:task_id/:execution_id?metric=...&from=...&to=...

GET
/v1/phantom_mf/statistics/:application_id/:task_id/:execution_id?metric=...&host=...&from=...&to=...
```

4.4 UPCOMING ACTIONS

The following major actions will be performed by the final release:

Interface extension. We will continue the implementation of RESTful interfaces for users query and management requests. It is also possible to use the monitoring server for storing and managing platform's configuration and deployment data.

Resource manager integration. The configuration of the Monitoring Client, including the plug-in information, should be done via the Resource Manager Service (currently performed by the user manually by editing the configuration file).

Improved documentation. We will provide monitoring handbooks for the major hardware platforms that are supported by the Monitoring Framework.

5. Released Component: Resource Management Service

5.1 Functionality

5.1.1 Overview

Resource manager aims to track and control the status of the infrastructure (hardware) resources supervised by the PHANTOM platform with the goal of improving the application performance and decreasing the hardware device power consumption

These goals should be achieved by applying the following techniques which have been studied by several past projects, such as EXCESS and DreamCloud:

- Dynamic identification of the application bottlenecks with the aim of better hardware utilization and thus decreased execution time (by leveraging the Monitoring Framework services)

- Dynamic Power Management aiming to apply techniques like DVFS (Dynamic Voltage and Frequency Scaling) to decrease the power consumption of the hardware while ensuring the acceptable Quality Attributes like the execution time or the met deadlines

- Implementation of security management options as provided by the Security Manager

Unlike the OpenStack middleware, which was envisioned as a possible solution in the preparation phase of the PHANTOM project, our chosen resource management alternative is based on the in-house solutions of the project partners and provides a more light-weight, modular, and flexible solution that leverages the monitoring infrastructure facilities. A flexible JSON schema is used to keep track of all resource-related data.

In order to perform the identified actions, the PHANTOM Resource Manager (RM) is constituted of 3 main services:

- **Resource configuration service** – the web-service deployed at the premises of the Monitoring Server that keeps the hardware-specific (static) configuration of the managed devices. This configuration usually contains the information which is provided by the vendors of the hardware, such as the number of processing elements (cores), their cache latency, the maximum and minimum power of CPU and memory, and other device-specific information is stored. In particular, this information is used to configure the plug-ins of the monitoring framework. This information is filled whenever a new hardware device is added. In the current release, this operation has to be done manually by the users (infrastructure operators). The RM status tracking service is developed as a RESTful service and deployed by default on the facilities of the Monitoring Server. The communication with the tracking service is performed by means of a RESTful interface (Table 5) that allows setting (by means of PUT command) and obtaining (GET) the information

- **Status tracking service** – the web-service (and, as such, deployed at the premises of the Monitoring Server) that provides the most actual (dynamic) status of the managed hardware devices and the execution of applications. The resource allocation information includes the status of each elementary logical unit (e.g. the CPU core) of the managed device, such as the operational power mode, tact frequency, availability (e.g. whether the unit is occupied for the execution of an application or not). The service also provides scripts that allow the other components of the PHANTOM platform (e.g. the Deployment Manager) to manage (set) the status of the hardware devices that were selected by MOM for the deployment of application components.

- **The dynamic power management service –** a set of scripts that are used to control the power mode in which the controlled device is operating. The scripts, as well as the underlying management tools, are outcomes of the DreamCloud project. For the CPU-based devices, for example, the scripts apply the DVFS (Dynamic Voltage and Frequency Scaling) technique in order to reduce the power consumption of the individual cores.

Table 5: RESTful APIs of PHANTOM Resource Manager service

configs – resource configuration service		
/configs	GET	Get a list of all registered platforms with links to their configuration details
/configs/:platform_id	GET	Get configuration details (e.g. monitoring-related parameters) for a specific platform
	PUT	Add/Change the configuration (e.g. monitoring-related parameters) for a specific platform
resources – status tracking service		
/resources	GET	Get a list of all registered platforms with links to their resources details
/resources/:platform_id	GET	Get information about the resources available for a specific platform
	PUT	Create/Update the resources information for a specific platform

5.1.2 PHANTOM Advances

The core of the Resource Management services is coming from the past DreamCloud and EXCESS projects. In PHANTOM, the JSON-based hardware description schema, which is the core of all services included in the Resource Manager, has been reworked in order to address the heterogeneity of the resources testbed.

5.2 DEPENDENCIES AND INSTALLATION

Due to the fact, that Resource Manager is implemented as a service, and the Monitoring Server already includes a web-server that can host many different services, it was decided to provide the Resource Manager as a part of the Monitoring Server release (see Section 4 for details on this release).

5.3 UPCOMING ACTIONS

The following major actions will be performed by the final release:

- **Identification services.** We plan to provide an automatic procedure for identification of new resources and automatic provisioning of the resource schema.

- **Improved documentation.** We will provide resource management handbook for the major hardware platforms that are supported by the Resource Manager.

6. RELEASED COMPONENT: PHANTOM FPGA LINUX DISTRIBUTION AND FPGA INFRASTRUCTURE

6.1 FUNCTIONALITY

6.1.1 Overview

The Linux distribution that is used for the PHANTOM FPGA targets is defined and standardized to aid coordination between partners. This component allows for the parts of the distribution to be easily rebuilt and extended, for example, to support new FPGA boards.

The main purpose of the PHANTOM Linux is to be able to execute PHANTOM IP cores that have been mapped to an FPGA platform (CPU plus FPGA) by the Multi-Objective Mapper. The aim is to allow the PHANTOM platform to automatically map application components to target FPGA devices, automatically construct and implement FPGA designs which include those components, and to execute them seamlessly as part of the running application.

The PHANTOM distribution also contains the scripts which create PHANTOM-compatible FPGA designs. A PHANTOM hardware design encapsulates a set of IP cores, makes them available to the application components running in the Linux distribution, and includes the various security and monitoring requirements of the PHANTOM platform.

The current version of the release contains facilities to construct the bootloaders, kernel, root filesystem, hardware design, and the various drivers and interfaces. It also implements initial monitoring requirements by providing access to power monitoring. The additional security requirements provided by the PHANTOM platform will be added in later releases.

6.1.2 PHANTOM Advances

A short list of the major features is as follows:

- The distribution includes support for PHANTOM monitoring actions, for example, to read current power use.

- Support for communications between a Linux user space process and PHANTOM IP cores. IP cores are mapped to the User-space I/O subsystem so processes can map the address space of the IP core into that of the user space process. This is encapsulated in a provided API, which is documented in the release. This API is used by the PHANTOM IP core developers, which writing the software part of their IP core, to implement the transfer of data in and out of the reconfigurable logic.

- Support for automatic creation of FPGA hardware designs which encapsulate the PHANTOM IP cores developed as part of the project. A hardware design consists of a set of PHANTOM IP cores, wired up appropriately, and a further set of supplementary cores for tasks such as clock management, monitoring and debugging, and bus arbitration. When the Multi-Objective Mapper assigns a set of IP cores to a given FPGA platform, the generation tools assemble the design from the specified PHANTOM IP cores and the necessary supplementary cores, and wires everything appropriately.

6.2 INSTALLATION

Full installation and building instructions are in the source repository. The README file explains how to build the kernel, device tree, and bootloaders for a given target FPGA board. It also explains the process of automating the bitfile generation for your target device. Begin by checking out the repository:

```
git clone https://github.com/PHANTOM-Platform/PHANTOM-FPGA-Linux.git
```

Note that building the Linux distribution and any hardware designs require the Vivado Design Suite from Xilinx. The Linux distribution is built using Multistrap (https://wiki.debian.org/Multistrap) which must be installed.

6.3 UPCOMING ACTIONS

The following major actions will be performed by the final release:

Action 1. Integration of MPI for the implementation of the PHANTOM communications API. This API is used by the Deployment Manager and allows components to communicate with other components on other parts of the platform.

Action 2. Further support for monitoring metrics, as determined by the consortium. Power use of both the CPUs and FPGA, and various kernel metrics (i.e. memory use, cache use etc.) are currently supported. This action will implement custom hardware to monitor bandwidth between the CPU and FPGA.

Action 3. Implementation of Security features. The FPGA platform should also be able to provide as much of the PHANTOM security as possible. The investigation will determine what is possible.

7. Released Component: PHANTOM IP Cores Marketplace

7.1 Functionality

7.1.1 Overview

The IP Core Marketplace can be seen as a part of the PHANTOM Repository specific for storing all the dedicated FPGA logic block. The IP Cores in the Marketplace will serve as accelerators for specific functions, commonly used mathematical algorithms (e.g. Fast Fourier transform (FFT), Finite impulse response (FIR), etc.), or image processing filters (e.g. Sobel Filter, Discrete wavelet transform (DWT), etc.) implemented in FPGA logic fabric.

7.1.2 PHANTOM Advances

A conjugation of the easy to use PHANTOM platform with the pre-designed IP cores can accelerate applications without the need to redesign from scratch the FPGA specific part.

A short list of the major extensions is as follows:

- Reusability of IP Cores that started from application-specific needs.
- Implementation of generic functions, that can be used outside PHANTOM application cases.

7.2 Dependencies and Installation

IP Core Marketplace is expected to be existent in the form of a Git repository.

```
$ git clone https://github.com/PHANTOM-Platform/PHANTOM-IP-Core-Marketplace
```

7.3 Usage Examples

Discrete Wavelet Transform (DWT)

A discrete wavelet transform (DWT) is any wavelet transform for which the wavelets are discretely sampled. As with other wavelet transforms, a key advantage it has over Fourier transforms is a temporal resolution: it captures both frequency and location information (location in time). Wavelets are often used to denoise two-dimensional signals, such as images. The original image is low-pass filtered, high-pass filtered and downscaled yielding four images, each describing local changes in brightness (details) in the original image.

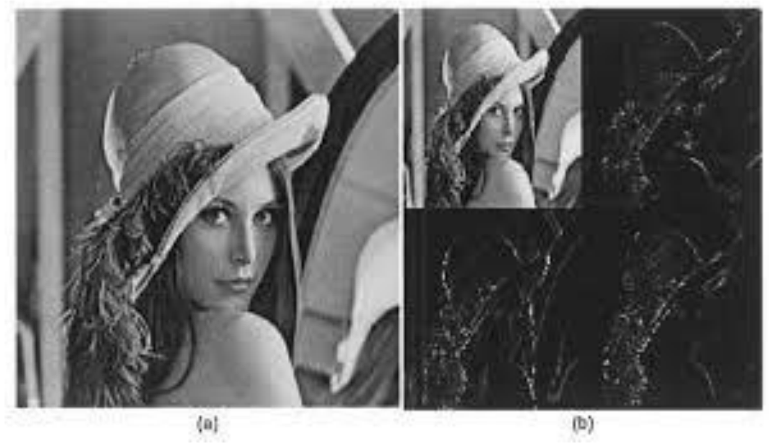


Figure 11: a) Image before DWT   b) Image after DWT

Design Flow

This filter is implemented in FPGA logic encapsulated in an IP Core that can be found in the PHANTOM IP Core Marketplace. This DWT IP Core (Figure 12) receives an image via an AXI Stream, does the processing transparently and outputs four images also via AXI Stream interfaces. The IP Core can be controlled (Start, Stop, etc.) via the axi_CONTROL_BUS.

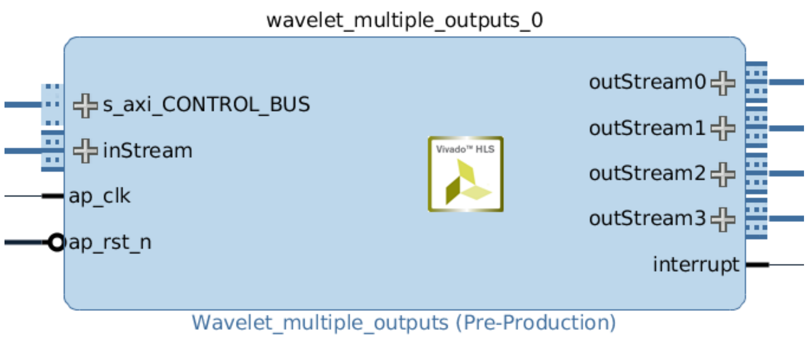


Figure 12: IP core interfaces example

AXI is part of ARM AMBA, a family of microcontroller buses first introduced in 1996. The first version of AXI was first included in AMBA 3.0, released in 2003. AMBA 4.0, released in 2010, includes the second major version of AXI, AXI4.

- There are three types of AXI4 interfaces:
- AXI4: For high-performance memory-mapped requirements.
- AXI4-Lite: For simple, low-throughput memory-mapped communication (for example, to and from control and status registers).

- AXI4-Stream: For high-speed streaming data.

To be able to use this IP Core it needs to be integrated into a functional FPGA design that exploits these types of interfaces. The design in Figure 13 shows the normal usage of this DWT IP Core integrated with four DMAs to handle all the data streaming, interconnects to manage the connections between different blocks and the Zynq Processing System, to control the data transfer between the DMAs and the IP Core and also to control the IP Core functionality.

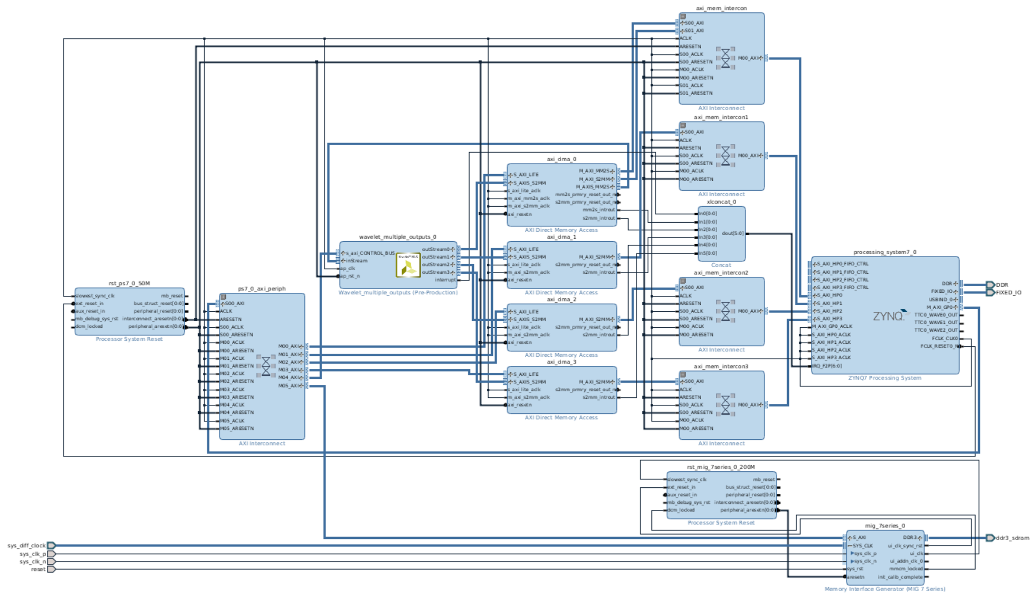


Figure 13: IP core design example

All the logic blocks are controlled using default AXI interfaces like the ones described above.

- S_AXI_LITE -> DMA Control
- S_AXI -> Memory Interface Control
- S_AXI_CONTROL_BUS -> IP Core Control

AXI Stream interfaces are used between the IP Core and DMAs to handle all the high speed data transfer that goes in and out of the IP Core.

- M_AXIS_S2MM -> Master Stream-to-Memory-Map (S2MM)
- M_AXIS_MMS2 -> Master Memory Map-to-Stream (MM2S)
- S_AXIS_S2MM -> Slave Stream-to-Memory-Map (S2MM)
- S_AXIS_MMS2 -> Slave Memory Map-to-Stream (MM2S)

The DMAs are also connected to the Zynq Processing System via S_AXI_HP (High-Performance Slave Interface) that gives them direct access to the physical DDR RAM to allow very fast memory transfers from the DDR to the IP Core and back to the DDR RAM.

- S_AXI_HP (HP0, HP1, HP2, HP3) -> High Performance Slave Interface

Software Flow

The code flow to use the IP Core and DMAs is as follows.

We start by initializing the DMAs and DWT IP Core.

```
CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
XAxiDma_CfgInitialize(&axiDma0,CfgPtr);
CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_1_DEVICE_ID);
XAxiDma_CfgInitialize(&axiDma1,CfgPtr);
CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_2_DEVICE_ID);
XAxiDma_CfgInitialize(&axiDma2,CfgPtr);
CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_3_DEVICE_ID);
XAxiDma_CfgInitialize(&axiDma3,CfgPtr);

CfgPtr=XWavelet_multiple_outputs_LookupConfig(XPAR_WAVELET_MULTIPLE_OUTPUTS_0_DEVICE_ID);

XWavelet_multiple_outputs_CfgInitialize(&waveletFilter0,CfgPtr);
```

Then we flush the cache to avoid sending garbage to the IP Core.

```
Xil_DCacheFlushRange((u32)m_dma_buffer_Src,SIZE_ARR*sizeof(u8));
```

Then the IP Core is started and the DMAs are instructed to start transferring data.

```
XWavelet_multiple_outputs_Start(&waveletFilter0);
XAxiDma_SimpleTransfer(&axiDma0,(u32)m_dma_buffer_Src,SEND_BLOCK*sizeof(u8),XAXIDMA_DMA_
TO_DEVICE);
XAxiDma_SimpleTransfer(&axiDma0,(u32)m_dma_buffer_Dst0,RECV_BLOCK*sizeof(u8),XAXIDMA_DEVI
CE_TO_DMA);
XAxiDma_SimpleTransfer(&axiDma1,(u32)m_dma_buffer_Dst1,RECV_BLOCK*sizeof(u8),XAXIDMA_DEVI
CE_TO_DMA);
XAxiDma_SimpleTransfer(&axiDma2,(u32)m_dma_buffer_Dst2,RECV_BLOCK*sizeof(u8),XAXIDMA_DEVI
CE_TO_DMA);
XAxiDma_SimpleTransfer(&axiDma3,(u32)m_dma_buffer_Dst3,RECV_BLOCK*sizeof(u8),XAXIDMA_DEVI
CE_TO_DMA);
```

We wait for the DMAs and IP Core to finish the work.

```
while(XAxiDma_Busy(&axiDma0,XAXIDMA_DEVICE_TO_DMA));
while(XAxiDma_Busy(&axiDma1,XAXIDMA_DEVICE_TO_DMA));
while(XAxiDma_Busy(&axiDma2,XAXIDMA_DEVICE_TO_DMA));
while(XAxiDma_Busy(&axiDma3,XAXIDMA_DEVICE_TO_DMA));
```

And then we invalidate the cache to avoid reading garbage.

```
Xil_DCacheInvalidateRange((u32)m_dma_buffer_Dst0,SIZE_ARR_OUT*sizeof(u8));
Xil_DCacheInvalidateRange((u32)m_dma_buffer_Dst1,SIZE_ARR_OUT*sizeof(u8));
Xil_DCacheInvalidateRange((u32)m_dma_buffer_Dst2,SIZE_ARR_OUT*sizeof(u8));
Xil_DCacheInvalidateRange((u32)m_dma_buffer_Dst3,SIZE_ARR_OUT*sizeof(u8));
```

7.4 UPCOMING ACTIONS

The following major actions will be performed by the final release:

Action 1. Extension of available IP Cores.

8. INNOVATIONS BEYOND THE STATE-OF-THE-ART

The tools that were presented in Sections 2-7 (the Monitoring and Resource Management Framework, the PHANTOM FPGA Linux Environment, the PHANTOM IP Core Marketplace) are results of the PHANTOM project partners' work undertaken for Tasks 4.1 – 4.4 of Work Package 4.

The development is based on a) outcomes of the past projects like Excess, DreamCloud, or JUNIPER, b) PHANTOM partners' owned open-source developments, but **required a substantial improvement** to address i) the heterogeneous landscape of the hardware resources (CPUs, GPUs, FPGAs, SoCs) and ii) the requirements of the PHANTOM exemplary use case applications.

The following subsections highlight the specific innovations endorsed by the work of the PHANTOM partners, description of the major developments and future roadmaps.

8.1 THE MONITORING CLIENT

8.1.1 Short summary of major innovations

Unified heterogeneous embedded hardware monitoring. Monitoring of heterogeneous hardware environment (CPU, GPU, **FPGA, SoCs**, etc.) can be performed as easily as of homogeneous infrastructures thanks to the flexible and modular design of the Monitoring Client architecture with configurable plug-ins, that correspond to the specific type of devices but endorse unified (service-oriented) interfaces. Dedicated plug-ins were developed to support SoC architectures (pure FPGA support is upcoming work). Each plug-in defines a set of metrics to be collected for each specific device type (CPU, GPU, etc.). Already developed plug-ins can be easily extended to support new devices. The Monitoring Client provides a platform for the integration of diverse plug-ins.

Measurement of the real energy consumption of the embedded, GPU-hosting, and FPGA-hosting devices. Unlike alternative solutions, which apply sampling-based techniques and estimations (all tools that rely on RAPL, PCM and other counters) at the processor-level only (!), the PHANTOM solution enables a true analysis of the power and energy metrics of the complete hardware device (or board, for the case of an embedded system) by leveraging the integration with the external high-precision measurement devices like ACME. The advantage of the employed measuring procedure over the other monitoring tools (like Zabbix, Nagios, etc.) is that the data collected represent the **real values** while data of the other tools are generated from estimations. Such estimations might be errorneous and include a bin error margin due to they are calculated from similar hardware on a similar set of applications; the estimations can also be erroneous when the hardware contain even minimal changes (like another memory module).

Security auditing additions. System auditing gathers selectable security-relevant events from trusted system software into a log in persistent store. PHANTOM unifies application, optimization, and security logging into a common framework and achieves economy of mechanism by implementing them with common mechanisms.

8.1.2 Background technologies utilized in development

The Monitoring Client is generally based on the solution proposed for the DreamCloud[3] and Excess projects (under the code-name ATOM), which were substantially extended to meet the specific requirements of the PHANTOM applications and the PHANTOM platform (see Figure 14). Table 6 lists the PHANTOM-specific extensions.

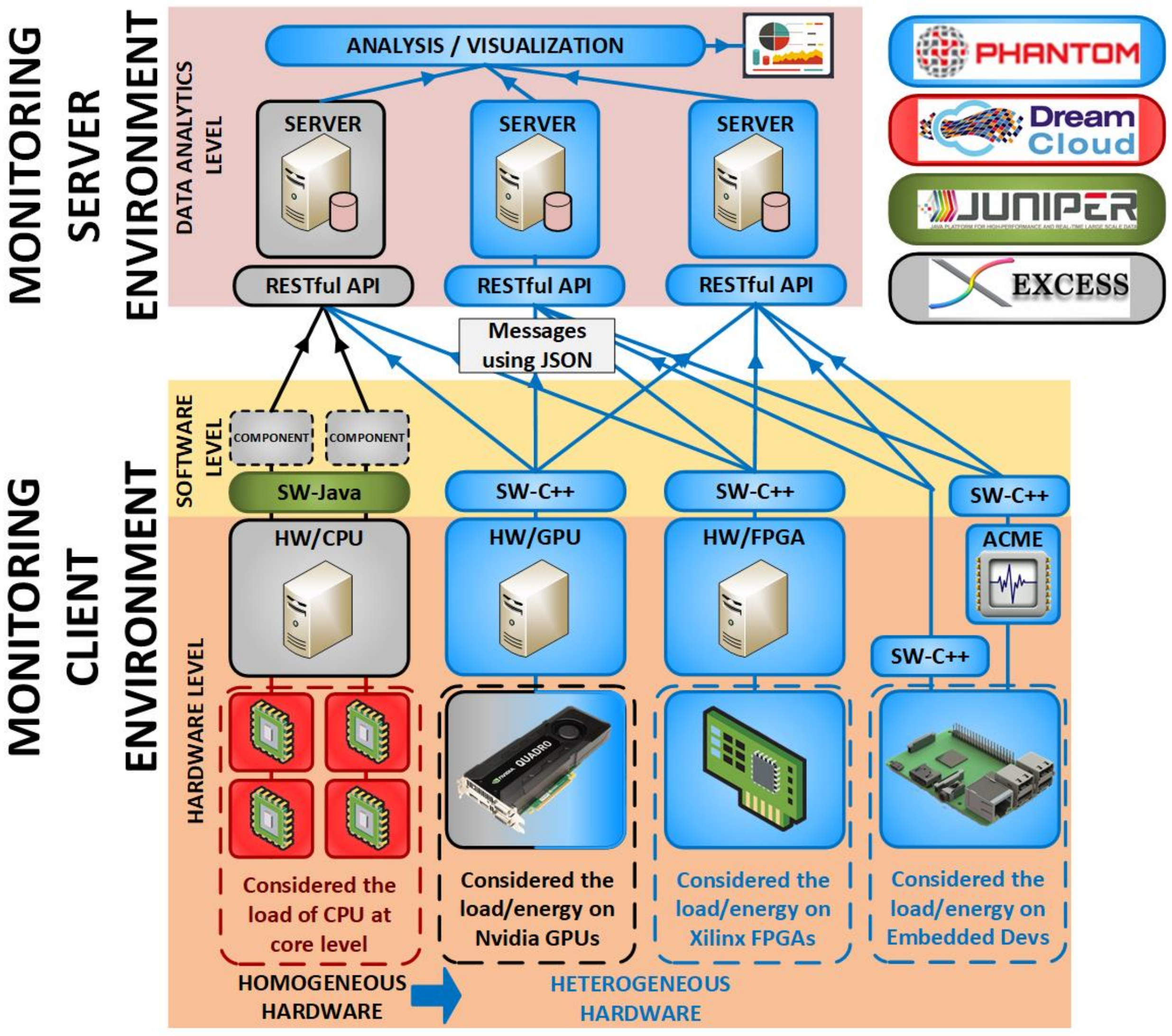


Figure 14: Evolution of the ATOM Monitoring Framework functionality: The grey part represents the activities performed by EXCESS project, green – JUNIPER, red – DreamCloud, blue - PHANTOM

[3] http://www.dreamcloud-project.org/

Table 6: Contributions of previous projects to the Monitoring Client development and PHANTOM extensions

Component	Key Feature	EXCESS	Dream Cloud	JUNIPER	Phantom
Monitoring Client	Core development	✓			
	Monitoring at application level				✓
	Heterogeneous systems support				(✓)
	Highly-accurate time measure-ments				✓
	Support of additional user-defined metrics specific to their applications				✓
	Workflows support		✓		
Plug-ins	hw_power	✓			
	Energy cons. measures of em-bedded devices from external sensor				✓
	Infiniband	✓			
	Meminfo	✓			
	Movidius	✓			
	Xilinx FPGAs				(✓)
	Nvidia	✓			
	Papi	✓			
	Rapl	✓			
	Sensors		✓		
	Vmstat			✓	
	iostat-bash			✓	
	Nvidia-bash	✓			✓✓
	Vmstat-bash			✓	
	Security Audit Monitoring				(✓)

(✓): Planned for the full prototype;   ✓: Already complete for the early prototype

✓✓: To be re-implemented in PHANTOM

8.1.3 Summary of new technologies/extensions developed

Application-level monitoring (contribution to Task 4.3)

PHANTOM use case applications require measurement of the load at the application level. That means that it should be possible to derive the fraction of the resource usage, consumed by a specific application (regardless of the other applications running on the same node/device).

For the implementation of this functionality in the Monitoring Library (see D2.1), the Monitoring Client required an extension of the solution of the ATOM Monitoring Framework (developed in EXCESS and DreamCloud), which can monitor the load of the complete system only (i.e. at the system-level instead of the required application-level).

The innovation supports the measurements of individual applications and their components which are allocated to (different) devices. This information is queried from different sources – such as the OS (knowing the ids of processes and threads that are associated with the specific application run) or hardware counters.

Time granularity of measurements (contribution to Task 4.3)

PHANTOM use case applications require measurement time accuracy of the orders of hundreds of microseconds. However, the existing frameworks (also the EXCESS and DreamCloud monitoring solutions) can only guarantee the accuracy in the order of tens of milliseconds, due to being based on the OS information only. PHANTOM improves the original solution by obtaining the timing information directly from the hardware counters of the CPU, and not from the OS, like before. This provides a time accuracy in the order of few microseconds. Next table shows a typical accuracy of different measuring methods.

Table 7 Typical resolution of each measuring time method[4].

Method	Typical Resolution	Typical Accuracy	Granularity	Difficulty of Use	
stop-watch	0.01 sec	0.5 sec	program	Easy	
Date	0.02 sec	0.2 sec	program	Easy	
Time	0.02 sec	0.2 sec	program	Easy	
prof and gprof	**10 msec**	**20 msec**	**subroutines**	**moderate**	**<----- without the improvements at PHANTOM**
clock()	15-30 msec	15-30 msec	statement	Moderate	
software analyzers	10 µ sec	20 µ sec	subroutine	Moderate	
timer/counter chips	**0.5-4 µ sec**	**1-8 µ sec**	**statement**	**Hard**	**<----- Improved version at PHANTOM**
logic or bus analyzer	50 nsec	half µ sec	statement	Hard	

[4] Measuring Execution Time and Real-Time Performance, David B. Stewart, Pages 1-15, Embedded Systems Conference Boston, Sept. 2006

External power measurement systems support (contribution to Task 4.1)

Added support for ACME and BeagleBone Black boards (to specifically address embedded and small systems due to the maximum supported the load of 6A and 20.5V, in order of 100W). This is a portable device about 8,5 cm x 5,5cm and 39gr. It can run Linux OS, software, and buffer the measurements if the network does not allow sending the data. The previous projects (EXCESS) relied on an integrated PCIE analog to digital converter[5], which cannot be used with embedded devices or any other without a PCIE slot.

Security Auditing advances (minor contribution to Task 4.3 and Task 2.1)

PHANTOM provides a security auditing service both for system use and for use by applications. System auditing gathers selectable security-relevant events from trusted system software into a log in persistent store. The design incorporates decisions to provide special protections to system auditing over other monitoring services, including, a special handler for system auditing, the ability to select events to be audited from a set of auditable events, distinct storage locations, auditing or the starting and stopping of the audit service, auditing the change of handler function, auditing the change of log location, and minimized potential loss of audit records. Flexibility is achieved with the ability to save audit event-specific data.

A client module, included into the Monitoring Client, provides an auditing API for security-enforcing and security-relevant software to invoke the auditing services that are available to both system-level software and applications. The API is used for system-level auditing to report each noted occurrence of events, from a configurable list of security-relevant events, to the audit system for potential recording in the system audit log. Applications are provided with the same framework for application-defined audit logging, which is provided at a best-effort level-of-service to prevent application-level denials of service to guaranteed system-level audit logging. Further details of the preliminary audit specification are provided in Appendix IV.

8.1.4 Early/Full Prototypes functionality

Table 8 summarizes the already developed functionality/features of the PHANTOM Monitoring Client and lists the upcoming actions.

[5] PCI- Express analog I/O board ACPIe-3121: url http://addi-data.com/products/pc-cards/pci-express-boards/pci-express-boards-analog/pci-express-analog-io-board-apcie-3121-apcie-3123/

Table 8: Monitoring Client functionality development roadmap

Feature/Functionality	Target for Release	
	Early Prototype [M18]	Full Prototype [M33]
Heterogeneous HW support		
• CPUs and Embedded ARM boards	✓	
• GPU (NVidia)		✓
• FPGA (Zynq)		✓
Integration of Monitoring Library	✓	
Highly-time accurate measurements	✓	
Support of external power monitoring equipment	✓	
Security Auditing API and implementation		✓

8.2 THE MONITORING SERVER

8.2.1 Short summary of major innovations

Distributed server architecture required for high performance and scalability of monitoring and analytics. The PHANTOM Monitoring Server design targets distributed databases, thus eliminating scalability bottlenecks that are pertained to big systems like HPC, with a big number of Monitoring Clients that need to be simultaneously supported. The ElasticSearch database allows a **decentralized monitoring**, i.e. when the instances of the Monitoring Server are installed on the same hardware as the Monitoring Clients, is also possible in such distributed design.

Support of security auditing. Along with the infrastructure- and application-specific metrics, security audit events can be stored and analyzed in the monitoring database.

Support of push subscription mechanism for retrieval of monitoring events. The PHANTOM Monitoring Server will provide a push service and subscription method to accessing the monitoring database (unlike the traditional approaches based on constant pulling), which among others it will reduce the traffic between the database and the clients and simplifies development of the back-end analysis and visualization tools.

Integration of NodeJS - JavaScript-based analytics on the collected data. The PHANTOM Monitoring Server allows the execution of Java scripts directly at the location of the stored data (collected data from different nodes and/or different tasks) in order to avoid unnecessary traffic and improve the analytics performance of the analytics tools and algorithms. Such analytics was previously performed on the client-side; however, in that case, the data can be considered only from a single experiment and no aggregation across all other experiments was possible.

Support of micro-analytics functions in the user's queries. The PHANTOM Monitoring Server is able to execute some plain analytics functions (such as min/max/average search) on the submitted data right from the user's queries to it.

8.2.2 Background technologies utilized in development

The ATOM Monitoring Server, similarly to the Monitoring Client, is generally based on the solution proposed for the DreamCloud[6] and Excess projects, which were substantially extended to meet the specific requirements of the PHANTOM applications and the PHANTOM platform (see Figure 14 in the previous Monitoring Client section). Table 9 lists the PHANTOM-specific extensions for the Monitoring Server.

Table 9: Contributions of previous projects to the Monitoring Server development and PHANTOM extensions

Component	Key Feature	EXCESS	Dream Cloud	JUNIPER	Phantom
Monitoring Server	Core development	✓			
	Data export	✓			
	Data push service				(✓)
	Security auditing				(✓)
	Distributed server				✓
Web inter-face	Visualization	✓			
	Public hosting	✓			
RESTful API	Core API	✓			
	Data analytics				(✓)
	Support the decision making for heterogeneous systems				(✓)
	Extended API for workflows		✓		
	Energy reports		✓		
	Statistics API	✓	✓		
	Progress tracking		✓		

(✓): Planned for the full prototype; ✓: Already complete for the early prototype

[6] http://www.dreamcloud-project.org/

8.2.3 Summary of new technologies/extensions developed

A push-based subscription mechanism to obtain new events is being implemented for the clients (supervising components of the PHANTOM platform) - a major contribution to task T4.3.

This mechanism was not available in the previous projects. However, this feature is required for the PHANTOM platform, because some tools such as the Multi-Objective Mapper need to constantly poll the monitoring database for new data, which might cause network congestions if the application takes a longer time to complete. It can be realized using frequent requests from them, but this strategy would result in unnecessary bi-directional data traffic (even if there is not new data) and then generate unnecessary load on the data server. Therefore, we are implementing a mechanism based on WebSockets that allows subscribing to the changes on the data whenever needed. Our implementation proposal consists of keeping a register of the previously received data and updating it with newly received records based on subscriptions. Notice that the server only sends *update modifications messages* of those tables to destinations that are subscribed to the event. It means that some processing is required on update messages, which are in JSON format. The subscription will be easy to perform as the following websocket request:

ws://server:ws_port/database/table/_changes

Support of analytics functions on the monitoring data. Monitoring relies on data, that are being dynamically collected by special Monitoring Clients for the (heterogeneous) infrastructure devices and also for the applications running on those devices (facilitated by a special Monitoring Library, see D2.1). Even for relatively simple hardware infrastructure configurations, the amount of the collected data can easily reach the amount of approx. 1GB per day per the simplest monitored device (as for the low-power ARM-based Odroid system, shown in the review). The application-specific monitoring can be much larger (e.g. for the pilot HPC use case prototype approx.. 50MB are collected per run).

All collected data are stored in the Monitoring Database – permanently, in order to allow the further analytics on them. The analytics is required by the supervising components of the PHANTOM platform, such as the Multi-Objective Scheduler, in order to identify the hardware utilization profiles of the specific components of a PHANTOM application (note that the components of the same application can be executed on different, heterogeneous devices). A typical example of the analytical operation is the calculation of the maximum, minimum, and the average utilization of the heterogeneous hardware during the execution of the application, such as the CPU, memory, I/O, network load or even cache misses etc.

Given the large size of data and the complexity of the data structures in the Monitoring Database as well as aiming to reduce the amount of communication overhead between the PHANTOM platform components, such analytic operations have to be performed on the Monitoring Database side. In order to facilitate the data analytics, the Monitoring Server provides means for both Macro- and Micro-Level analytics: the **Macro-level** analytics can be implemented directly on premises of the Monitoring Server with **JavaScript (NodeJS scripts)**, and the **Micro-level** can be implemented by means of injection of Groovy, JavaScript or Python scripts into the queries to the Monitoring Server.

Micro-level analytics will provide the advantage of the easy extension of the already implemented analytics by the users who have no access to the Monitoring Server (similarly to JavaScript on the HTTP pages). The micro-level analytics will allow to embed the user-defined functions into the queries to the Monitoring Database. As an example of an embedded script, see Listing below.

Listing 14: Example of micro-analytics query

```
curl -XGET 'http://server/database/table/_ search?&pretty=true&size=3' -d '{
        "query": {  "match_all": {}  },
        "sort": {
                  "script" : "ctx.table.field1 += param1",
                  "script_lang" : "groovy",
                  "script_params" : {
                  "param1" : 1
                  },"order" : "asc"
        }
}'
```

Support of security auditing. A server plug-in that permits generation and storing of security audit events in a distinct security audit log is a novel aspect of the design. The security audit module is realized through the extensibility of the Monitoring Framework, and provides security auditing services both to system software with guarantees, and to potential application-specific auditing with best effort. For system-level auditing, a set of auditable events are defined, and may be selected by an administrative API for audit capture by the server, which may also generate alarms based on the occurrence of event patterns. The unification of security auditing services with the Monitoring Framework is the principal innovation of this approach. By extending and leveraging the monitoring infrastructure, security auditing features can be provided with an economy of mechanism. Further details of the preliminary audit specification are provided in Appendix IV.

8.2.4 Early/Full Prototypes functionality

Table 10 summarizes the already developed functionality/features of the PHANTOM Monitoring Server and lists the upcoming actions.

Table 10: Monitoring Server functionality development roadmap

Feature/Functionality	Target for Release	
	Early Prototype [M18]	Full Prototype [M33]
Push-based subscription to new events notification		✓
Data analytics with JavaScript	✓	
Micro-analytics function execution for the use queries		✓
Security Auditing		✓
Integration with the other components of PHANTOM platform • Basic • Advanced	✓ (Basic)	✓ (Advanced)

8.3 The Resource Management Service (Supervisor)

8.3.1 Short summary of major innovations

Unification of the monitoring and the resource management by implementing them with common mechanisms. In PHANTOM, a common technological platform, served by the Monitoring Framework, is used for both monitoring and resource management. For the resource management, such a unification enables a more light-weight architecture, a better performance (due to immediate availability of real data from the infrastructure), a more service-oriented design with numerous benefits for the users (in terms of simplified integration of external tools with the Resource Manager) as well as for the other tools of the PHANTOM platform (in terms of quick accessing to the resource configuration via a common data layer).

Status tracking of all logical units of the heterogeneous hardware. The PHANTOM Resource Management Service is designed to enhance any native RM software (like SLURM for HPC or OpenStack for Cloud or any RTOS for embedded) with the useful insight into all components of the heterogeneous hardware (for a plain CPU – at the core-level, or with the hosted GPU or/and FPGA). The resource manager relies on the monitoring capabilities and leverages the concept of dynamic resource management – i.e. there is no static schema with the information about the resource status; on the contrary, all information on utilization and energy consumption of the hardware can be retrieved in real-time and used to control the heterogeneous infrastructure.

Application management. The PHANTOM Resource Management keeps track of all applications, running in the supervised heterogeneous hardware environment and makes it available to all other tools of the PHANTOM platform via the Monitoring Server services.

Ahead-of-time resource reservation support. The PHANTOM Resource Manager is designed as a reactive component – it can not only respond to queries with requests about the resource status, but also accept the reservation requests from the applications for future allocations (e.g. in order to enforce the locality of data processing).

GPU Monitoring instrumentation at the application level. At the Excess project was used a multiple channel ADC card for measuring the energy consumption of the Nvidia GPUs. Instead of that, at the current project, we consider using the Nvidia Management Library (NVML) for monitoring at the application level. The instrumentation of the GPU monitoring (based on the GPU internal power management) will allow registering the availability and ahead-of-time resource reservation of the GPUs.

Open interfaces that facilitate the integration with native RM software. Whether SLURM or OpenStack, the PHANTOM Resource Manager will provide extensions to get access to the smallest details of the managed hardware, leveraging the monitoring capabilities.

8.3.2 Background technologies utilized in development

The PHANTOM Resource Management software is delivered as a part of the Monitoring Framework due to a number of overlapping requirements to the underlying software, such as the data layer, the service layer, etc.

It is a development started from scratch and leveraging the service-oriented design approach of the monitoring framework. The previous projects (like aforementioned DreamCloud and EXCESS) although offered some resource management capabilities but they were pretty rudimental (e.g., only a static resource allocation schema was provided) and have no use for a heterogeneous environment.

8.3.3 Summary of new technologies/extensions developed

- Started the implementation status tracker for heterogeneous infrastructure testbed (major contribution to task T4.4)
- Developed templates for registering a new device in the infrastructure (major contribution to task T4.4)
- Started the implementation of the application manager for PHANTOM applications (major contribution to task T4.4)

- Developed RESTful interfaces to access resources/applications status (major contribution to task T4.4)
- Started the implementation resource management library for controlling advanced reservation programmatically from the application or a management script (major contribution to task T4.3)

8.3.4 Early/Full Prototypes functionality

Table 11 summarizes the already developed functionality/features of the PHANTOM Resource Manager and lists the upcoming actions.

Table 11: Resource Manager functionality development roadmap

Feature/Functionality	Target for Release	
	Early Prototype [M18]	Full Prototype [M33]
Resource Tracker		
• CPUs	✓	
• GPUs		✓
• FPGAs		✓
• SoCs		✓
Application Management		
• Basic version (simple status tracking)	✓	
• Enhanced version (with submission interface)		✓
Management Scripts		
• Elaboration of regulation setting for all HW types	✓	
• Implementation of scripts for applying the regulation settings		✓

8.4 THE PHANTOM FPGA LINUX ENVIRONMENT

8.4.1 Short summary of major innovations

The FPGA Linux environment was developed by the University of York. Work in this area aims to allow efficient exploitation of FPGA platforms by non-experts. The following details are entirely automated by the environment.

- Creation of kernels, bootloaders, and root file systems.
- Libraries for interaction between Linux software and IP cores.
- The FPGA design containing requested IP cores.
- Monitoring sensors for the platform.
- PHANTOM Communication Libraries to allow the board to communicate with the rest of the PHANTOM application.

8.4.2 Background technologies utilized in development

The hardware generation makes use of FPGA vendor tools to compile the design.

8.4.3 Summary of new technologies/extensions developed

Definition of the PHANTOM IP core hardware interface which allows standardized IP cores from the marketplace to be automatically formed into an FPGA design, based on mappings from the Multi-Objective Mapper.

Definition and implementation of the PHANTOM IP core software interface, which allows userland software to interact with PHANTOM IP cores, including programming and querying the FPGA logic.

Implementation of the PHANTOM Communications API for the embedded Linux target. Based on MPI, this allows FPGA boards to be integrated as part of the PHANTOM platform.

Implementation of scripts and programs which automatically create FPGA designs from PHANTOM IP cores. The platform can, in response to a decision from the MOM, automatically create an FPGA design for a range of target FPGAs that includes the appropriate cores. Also generated is the required metadata for the userland software libraries.

Implementation of scripts and programs to build a reference Linux distribution, including required kernels and bootloaders, to allow deployment of the PHANTOM platform across a range of FPGA targets.

8.4.4 Early/Full Prototypes functionality

The early prototype environment already performs all features listed in section 8.4.1. The next prototype will extend functionality in the following areas:

Investigation and implementation of further monitoring capabilities. Aim to provide information on the bandwidth usage of various busses on the FPGA, specifically the memory contention, and the links between CPU and FPGA logic.

Investigation and implementation of further security capabilities. A range of potential security improvements can be investigated, such as providing better support for clearing memory, and providing physical isolation on the FPGA logic fabric.

Implement support for UltraScale+ boards. Currently, only Zynq-based boards are supported. Support for UltraScale+ boards would add a large number of additional supported targets.

Online operation. The full prototype will be extended to dynamic mapping to work online in response to runtime monitoring where possible. Dynamic migration will not be attempted, as this is focused on very long running tasks, due to the long run time of FPGA compilation. Instead, application components will be redeployed between system nodes.

8.5 The PHANTOM IP Core Marketplace

8.5.1 Short summary of major innovations

The IP Core Marketplace exposes the IP cores as components to the end user. The user can check, in the Marketplace, what algorithms are available as components, to be used in the FPGA without effort, and choose if one or more IP cores can be applied to his/her specific application. The user only needs to annotate their source code with a pragma informing the PHANTOM Platform to use a FPGA version of the component instead of the normal software version.

Although the IP cores are developed in a FPGA-based design flow, they are generic implementations of common algorithms. This means that from the end user perspective the FPGA IP cores are seen as components that implement a specific algorithm and can be reused in different applications. Each IP core corresponds to a component and several different IP cores can be deployed in a single FPGA, making the system very versatile.

The integration and deployment of IP Cores in the FPGA is done automatically by the scripts included in the PHANTOM Linux Software Distribution, which also includes the various security and monitoring requirements of the PHANTOM platform. The MOM chooses, for a specific application, which components should run in the FPGA and invokes the proper scripts, from the PHANTOM Linux Software Distribution, to create a FPGA implementation, that has the needed components/IP cores. The PHANTOM Linux Software Distribution scripts create a design and the proper interfaces to use these IP cores, synthesizes and implements the FPGA design to generate a bitstream for configuring the FPGA. It also takes care of exposing the IP cores, as Userspace I/O (UIO) devices on the Linux side, that are then used by the application, as components that communicate via MPI, just like the software components, from a user's perspective.

At first, this integration was done manually, as it is state-of-the-art and required a significant effort to correctly expose the IP Cores on the Linux side, available to use by any application. After this was successfully achieved it allowed the creation of scripts to automatize the integration process. Now, all the integration, from IP core, to FPGA design and synthesis, to Linux integration is done without the user intervention/knowledge.

8.5.2 Background technologies utilized in development

For the development of IP Cores, the Xilinx tools were used: Vivado, Vivado HLS, and Xilinx SDK.

For the Linux integration, the Petalinux tools were used.

For the first set of optimized IP Cores, a generic implementation of DWT (Discrete Wavelet Transform) was used as a starting point.

8.5.3 Summary of new technologies/extensions developed

- FPGA IP cores integration with Linux via Userspace I/O, Normally the FPGA/Linux integration is done using AMBA (Advanced Microcontroller Bus Architecture).

- DWT IP Core with performance enhancements vs Vivado HLS generation. Improvement went from 1118 ns per pixel to 5 ns per pixel of processing time.

8.5.4 Early/Full Prototypes functionality

In the Early Prototype the main focus was on GMV use case, and on the usage of SIMONS application on an embedded environment (e.g. a Drone), where the execution time of the SIMONS on ARM went from 70 minutes to 40 minutes using ARM plus FPGA.

This was using DWT IP Core, which in the original implementation is invoked 10 times. The next step on GMV case is the addition of a highly optimized IP Core for the Inverse DWT, which is expected to give even a bigger boost in time because it is invoked 20 times in a single SIMONS run.

The next main functionality for the IP Core Marketplace will be the automatic generation of PHANTOM Compatible IP Cores. The Marketplace will have an API that will allow for arbitrary C code be injected and IP Cores with the proper interfaces will be generated and make available in the Marketplace. This will be an exploratory work, since some limitations are expected, either in the interfaces and in the performance.

9. CONCLUSIONS

The deliverable presented the released version of the PHANTOM monitoring platform, runtime environment for reconfigurable (FPGA) components, and resource management framework. These components play an essential role for the PHANTOM software stack and are used by the other PHANTOM components, e.g. by the Multi-Objective Mapper (MOM). In particular, the Monitoring Framework provides useful insights in the application performance and hardware utilization, which is essential for scheduling. The FPGA Linux distribution enables using reconfigurable platforms in the same way as the standard CPUs and GPUs require.

The purpose of the release was to provide preliminary versions of the components that conform to the integration protocols of the PHANTOM platform and are stable enough to enable further development of the platform. The basic description of the components that are included in the preliminary release is supplied with the minimal necessary documentation needed for their installation and testing. For the next releases, we aim to considerably improve this documentation by creating handbooks and manuals with more detailed and better-structured manuals.

In the remaining lifetime of the PHANTOM project, the components of the preliminary release will be gradually improved according to the identified actions and provided in the upcoming enhanced (D4.3 in M27) and final (D4.4 in M32) releases.

APPENDIX 1. MONITORED METRICS

Table 12: List of metrics for standard CPU-based devices

category		metrics	methodology	unit	remarks
Application-level	performance	execution time	int clock_gettime(CLOCK_REALTIME, struct timespec *tp)	ns	
		CPU execution time	int clock_gettime(CLOCK_PROCESS_CPUTIME_ID, struct timespec *tp)	ns	
	resources utilization	CPU utilization	(process cpu time (/proc/[pid]/stat)) / (global cpu time (/proc/stat))	%	http://stackoverflow.com/questions/1420426/calculating-cpu-usage-of-a-process-in-linux
		RAM utilization	VmRSS (/proc/[pid]/status) / MemTotal (/proc/meminfo)	%	
		swap utilization	VmSwap (/proc/[pid]/status) / SwapTotal (/proc/meminfo)	%	
		virtual memory size	VmSize (/proc/[pid]/status)	KB	
	IO	disk IO throughput	(read_bytes + write_bytes) / seconds (/proc/[pid]/io)	Bytes/s	since kernel 2.6.20
	power	power (with given pid)	ptop (include CPU, memory, disk, wireless network)	milli-watt	
Infrastructure-level	performance	floating point instructions per second	PAPIF_flips	Mflip/s	PAPI
		floating point operations per second	PAPIF_flops	Mflop/s	
	resources utilization	CPU utilization	total_cpu_time – idle_time / total_cpu_time (proc/stat)	%	https://github.com/Leo-G/DevopsWiki/wiki/How-Linux-CPU-Usage-Time-and-Percentage-is-calculated
		RAM utilization	(MemTotal – MemFree) / MemTotal (proc/meminfo)	%	http://www.computerworld.com/article/2722141/it-management/making-sense-of-memory-usage-on-linux.html
		swap utilization	(SwapTotal – SwapFree) / SwapTotal (proc/meminfo)	%	
	IO	disk IO utilization	(Field 10 (# of milliseconds spent doing I/Os) in /proc/diskstat) / (total time)	%	https://www.kernel.org/doc/Documentation/iostats.txt
			(Field 11 (weighted # of milliseconds spent doing I/Os) in /proc/diskstat) / (total time)	%	
	temperature	temp per core	libsensors	°c	
	network	network throughput	(bytes_recv + bytes_trans) / seconds (/proc/net/dev)	Bytes/s	
	power	power	ptop (include CPU, memory, disk, wireless network)	milli-watt	
		power	external power measurement hardware	milli-watt	https://www.tindie.com/products/BayLibre/acme-power-measurement-kit/

Table 13: List of metrics for accelerated GPU-based devices

category		metrics	methodology	unit	remarks
Application-level	performance	execution time	int clock_gettime(CLOCK_REALTIME, struct timespec *tp)	ns	
Infrastructure-level	resources utilization	GPU utilization	utilization.gpu / 100 (nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization))	%	Percent of time **over the past sampling period** during which one or more kernels was executing on the GPU.
		GPU mem accessing rate	utilization.memory / 100 (nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization))	%	Percent of time **over the past sampling period** during which global (device) memory was being read or written.
		GPU memory total	Memory.total (nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *Memory))	bytes	
		GPU memory used	Memory.used (nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *Memory))	bytes	
	IO	PCIe transmit throughput	value / 0.020 (nvmlDeviceGetPcieThroughput (nvmlDevice_t device, NVML_PCIE_UTIL_TX_BYTES, unsigned int *value))	bytes/s	return bytes transmitted in 20 ms; transform to bytes/s
		PCIe receive throughput	value / 0.020 (nvmlDeviceGetPcieThroughput (nvmlDevice_t device, NVML_PCIE_UTIL_RX_BYTES, unsigned int *value))	bytes/s	return bytes received in 20 ms; transform to bytes/s
	temperature	GPU temp	nvmlDeviceGetTemperature (nvmlDevice_t device, NVML_TEMPERATURE_GPU, unsigned int *temp)	°c	
	power	GPU power (for entire board)	nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)	milliwatt	power for entire GPU board
		total power	ptop (power for CPU board) + GPU power	milliwatt	

Table 14: List of metrics for reconfigurable FPGA-based devices

category			metrics	methodology	unit	reference
PS (Processing System)	Application-level	performance	execution time	int clock_gettime (CLOCK_REALTIME, struct timespec *tp)	ns	
			CPU execution time	int clock_gettime (CLOCK_PROCESS_CPUTIME_ID, struct timespec *tp)	ns	
		resources utilization	CPU utilization	(process cpu time (/proc/[pid]/stat)) / (global cpu time (/proc/stat))	%	http://stackoverflow.com/questions/1420426/calculating-cpu-usage-of-a-process-in-linux
			RAM utilization	VmRSS (/proc/[pid]/status) / Mem Total (/proc/meminfo)	%	
			swap utilization	VmSwap (/proc/[pid]/status) /Swap Total (/proc/meminfo)	%	
			virtual memory size	VmSize (/proc/[pid]/status)	KB	
		IO	disk IO throughput	(read_bytes + write_bytes) / seconds (/proc/[pid]/io)	Bytes/s	since kernel 2.6.20
		power	power (with given pid)	ptop (include CPU, memory, disk, wireless network)	milli-watt	
	Infrastructure-level	performance	floating point instructions per second	PAPIF_flips	Mflip/s	PAPI high-level API
			floating point operations per second	PAPIF_flops	Mflop/s	
		resources utilization	CPU utilization	total_cpu_time – idle_time / total_cpu_time (proc/stat)	%	https://github.com/Leo-G/DevopsWiki/wiki/How-Linux-CPU-Usage-Time-and-Percentage-is-calculated
			RAM utilization	(MemTotal – MemFree) / MemTotal (proc/meminfo)	%	http://www.computerworld.com/article/2722141/it-management/making-sense-of-memory-usage-on-linux.html
			swap utilization	(SwapTotal – SwapFree) / SwapTotal (proc/meminfo)	%	
		IO	disk IO utilization	(Field 10 (# of milliseconds spent doing I/Os) in /proc/diskstat) / (total time)	%	https://www.kernel.org/doc/Documentation/iostats.txt
				(Field 11 (weighted # of ms spent doing I/Os) in /proc/diskstat) / (total time)	%	
		temperature	temp per core	libsensors	°c	
		network	network throughput	(bytes_recv + bytes_trans) / seconds (/proc/net/dev)	Bytes/s	
		power	power	ptop (include CPU, memory, disk, wireless network)	milli-watt	
PL (Programmable Logic)	Infrastructure-level	IO	read/ write transactions	AXI performance monitor	number of requests	
			read/ write latency (avg)	AXI performance monitor		
			read/ write latency (std. Dev)	AXI performance monitor		
			read/ write throughput	AXI performance monitor	MB/s	
PS + PL	infrastructure-level	power	total power	via UCD9248	milli-watt	http://www.wiki.xilinx.com/Zynq+Power+Management

APPENDIX 2. MONITORING CLIENT'S PLUG-INS

Currently, the Monitoring Client supports 7 plug-ins, whose implementation and design details are collected in the directory src/plugins. The monitoring client is designed to be pluggable. Loading a plug-in means starting a thread for the specific plug-in based on the users' configuration at run-time. The folder src/agent plays a role as the main control unit, as managing various plug-ins with the help of pthreads. Folders like src/core, src/parser, and src/publisher are used by the main controller for accessorial support, including parsing input configuration file (src/mf_config.ini), publishing metrics via HTTP, and so on.

Board_power plug-in

This plug-in is dedicated to collect power metrics sampled by an external ACME power measurement kit for a target platform. The ACME power measurement kit can be seen as a compact solution composed of both the hardware and software (http://baylibre.com/acme/). From hardware aspect, there are two key components: a BeagleBone Black board as the main processing unit and an ACME cape as the extension for multi-channel power measurements. The ACME software suite contains an iio-daemon, which reads power measurements from the IIO (Industrial I/O) devices and sends the data out continuously via Ethernet (http://wiki.baylibre.com/doku.php?id=acme:start).

As shown in Figure 15, the Board_power plug-in runs on a hosting platform with the ACME power measurement kit being accessible through given IP address. The target platform, whose power consumptions are interested, is connected with the ACME power measurement kit by the supported power probe. As the ACME power probe is standard, the plug-in and ACME power measurement kit could be used as a generic solution for all possible platforms.

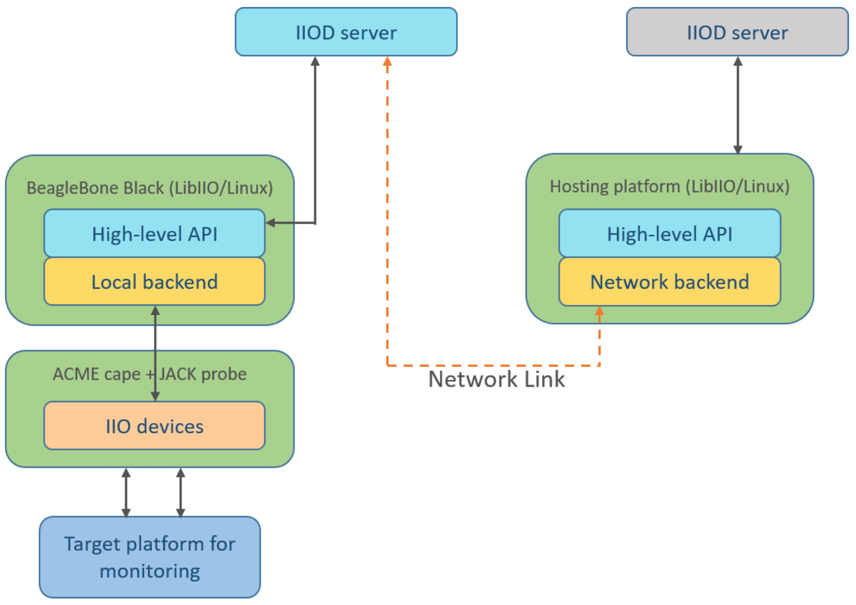


Figure 15: Connection between ACME power measurement kit and the monitoring platform

We use Libiio library for the plug-in's implementation (https://wiki.analog.com/resources/tools-software/linux-software/libiio). It is a library that has been developed by Analog Devices to ease the development of software interfacing Linux IIO devices. Since the iio-daemon running on the ACME kit supports network backends, we use the Libiio APIs firstly to create a network context, before filling IIO buffers and filtering the associated metrics.

CPU_perf plug-in

This plug-in measures performance-related metrics of CPUs in a fine granularity – per CPU core. For each CPU core, the floating-point operations per second, floating-point instructions per second, and total instructions per second are sampled and collected.

The implementation of the plug-in depends on essentially the PAPI hardware counters and PAPI library (http://icl.utk.edu/papi/) (http://icl.cs.utk.edu/projects/papi/wiki/Introduction_to_PAPI-C). It is advisory to check at first if the required PAPI events are supported on the target platform, which are PAPI_FP_OPS, PAPI_FP_INS, and PAPI_TOT_INS respectively.

CPU_temperature plug-in

The target of this plug-in is CPU's temperature per CPU core. To achieve this, the lm_sensors (Linux monitoring sensors) library is used (https://wiki.archlinux.org/index.php/lm_sensors). It is a free and open-source tool which provides interfaces for monitoring CPU's temperature and thermal properties.

Linux_resources plug-in

This plug-in is implemented to retrieve run-time information about Linux kernel and processes by using Linux provided /proc file system. The metrics supported include the CPU utilization percentage, memory utilization percentage, disk I/O statistics and network statistics.

We calculates the CPU utilization rate by dividing the CPU usage time by the total CPU time, both of which can be derived from the file /proc/stat (https://github.com/Leo-G/DevopsWiki/wiki/How-Linux-CPU-Usage-Time-and-Percentage-is-calculated). Because that the values read directly from /proc/stat are time in cycles since system boot, it is thus necessary to calculate at first the time passed by during an interval before calculating the CPU utilization rate.

For memory monitoring, we read from /proc/meminfo the current total available physical memory size, the unused physical memory size, the total amount of swap available, and the total amount of swap free, in order to calculate the RAM usage rate and the swap usage rate at the current time point (http://www.computerworld.com/article/2722141/it-management/making-sense-of-memory-usage-on-linux.html).

The total system disk I/O read and write bytes are derived by adding all read/write bytes per process during the sampling interval, which can be read from the /proc/[pid]/io files (https://www.kernel.org/doc/Documentation/iostats.txt). Then the system I/O throughput can be retrieved by dividing the total amount of read/write by the time interval. Similarly, we calculate the network throughput based on statistics read from file /proc/net/dev (http://man7.org/linux/man-pages/man5/proc.5.html).

Linux_sys_power plug-in

With the help of Linux kernel and /proc file system, we implement this plug-in, which is capable of power monitoring of various system components, including CPU, memory, disk I/O, and wireless network.

The CPU power consumption is estimated by using a Linux module named as cpufreq-stats (http://lxr.free-electrons.com/source/Documentation/cpu-freq/cpufreq-stats.txt) (https://www.kernel.org/doc/Documentation/cpu-freq/cpufreq-stats.txt). It is a driver that provides CPU frequency statistics for each CPU through its interface, which appears normally in the directory /sysfs/devices/system/cpu/cpuX/cpufreq/stats. By reading the values kept in the file time_in_state, we retrieve the amount of time spent in each of the frequencies supported by the dedicated CPU. We assume that the relationship between CPU frequency and power consumption is a linear correlation, consequently it is feasible to estimate the average CPU power during an interval with given minimum and maximum CPU power specifications.

For memory power estimation, we uses the system call "__NR_perf_event_open" to stat the hardware cache misses (http://man7.org/linux/man-pages/man2/perf_event_open.2.html). Together with reading the disk I/O read/write statistics, we calculate the memory power consumption with the following formula. The L2 cache miss latency and L2 cache line size can be obtained via some known calibrator (http://homepages.cwi.nl/~manegold/Calibrator/calibrator.shtml).

$$\frac{\left(\frac{IO_{read} + IO_{write}}{L2CacheLineSize} + MemAccess\right) \times L2CacheMissLatency \times MemPower}{Sample\ Interval}$$

Disk and wireless network power consumptions are calculated based on their activities, like read/write and receive/send bytes during the sampling interval. As long as the energy specifications of the disk and wireless network card are given, we could compute the constants, like energy cost per disk read/write and energy cost per wireless network receive/send, and get finally the energy consumed during a specific period.

Our implementation is based on the methodology proposed by the pTop project. Please refer to the project web page for more details and information. (http://mist.cs.wayne.edu/ptop.html) (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.360.7151)

NVML plug-in

The target of this plug-in is the Nvidia GPU, which is normally hosted by a connected CPU. The plug-in, which runs on the hosting platform, uses the NVML (Nvidia management library) provided C-based APIs to monitor the associated GPU devices' properties (https://developer.nvidia.com/nvidia-management-library-nvml). The metrics supported by this plug-in, including GPU usage rate, GPU memory usage rate, PCIe throughput, GPU temperature and GPU power consumption, depend on the models of the GPU devices. For example, PCIe statistics are available only for Maxwell or newer architectures (https://docs.nvidia.com/deploy/pdf/NVML_API_Reference_Guide.pdf).

RAPL_power plug-in

For Intel CPU an alternative method for power monitoring is by using this plug-in, which is implemented with the help of RAPL (Running Average Power Limit) provided energy and power information. RAPL, as clarified in the reference, (https://01.org/zh/blogs/2014/running-average-power-limit-%E2%80%93-rapl?langredirect=1) is not an analog power meter, but rather uses a software power model to estimate energy usage by using hardware performance counters and I/O models.

In general cases, RAPL domains cover both the CPU package (including core and uncore devices) and the DRAM, as can be seen from Figure 16. However, the specific RAPL domains available in a platform vary across product segments.

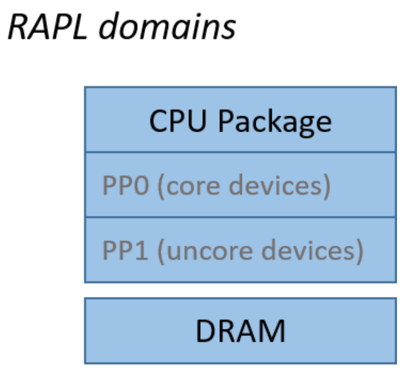


Figure 16: RAPL power measurements domains

APPENDIX 3. MOVIDIUS BOARD INTEGRATION IN MONITORING FRAMEWORK

Myriad2 general information

A high-level diagram of the Myriad2 platform architecture is shown in Figure 17. On the Myriad2 platform, there are twelve 128-bit vector-processors units (SHAVE processors), SIPP Hardware Accelerators, 2MByte on-chip SRAM (CMX), 64-bit interface to DDR2/3 RAM running at up to 1033MHz and a range of other peripherals. Besides, the two Leon4 RISC processors are used to manage execution: the RISC1 core is designed to be real-time controller for scheduling the activity of Myriad2 while RISC2 core is designed to run an operating system such as Linux or RTEMS and to manage all the control interfaces. Each of the LEON4 cores also includes a fully IEEE754 compliant FPU with fp64 support which allows the platform to provide native fp64 support either standalone, or accelerated by the 12 on-board SHAVE processors which natively support fp16 or fp32 but not fp64. Each SHAVE is connected to the 256kB 2-way L2 Cache and has 2 x 64-bit data ports to CMX memory as well as a 128-bit instruction port. The SIPP Hardware Accelerators also have 16 x 2 x 64-bit data connections into the CMX at the nominal system clock frequency of up to 800MHz, delivering an aggregate 400GBytes/s of total bandwidth, necessary for sustained high performance for many numerical applications.

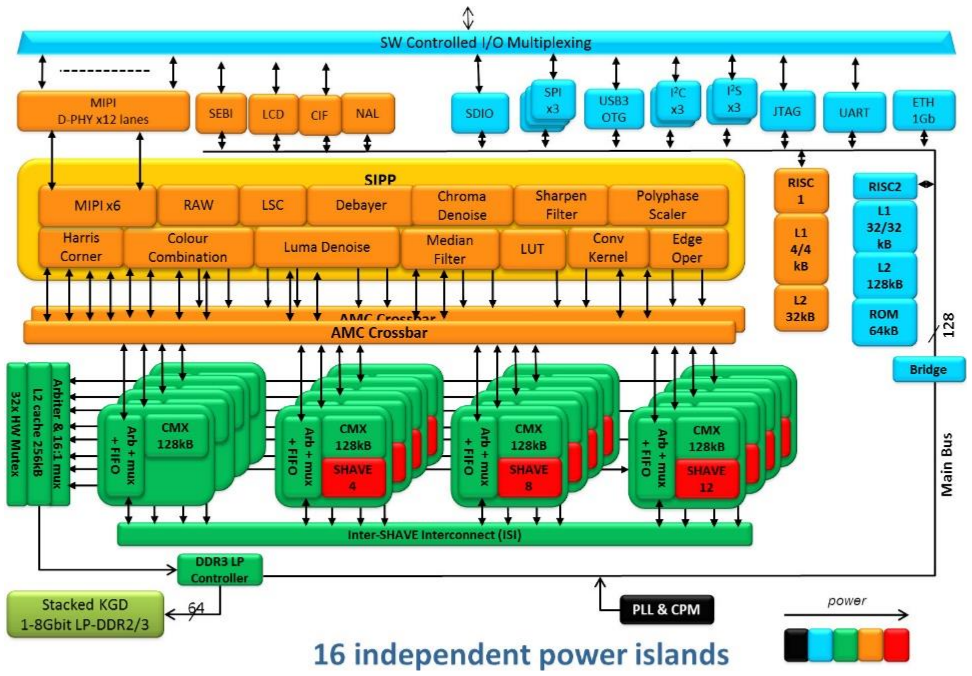


Figure 17: Hardware architecture and components of Myriad2 platform

Myriad2 software development

Software development tools

The Myriad2 software development environment is shown in Figure 18. Included in the MoviTools are its compiler, debugger, simulator, and so on, all developed by Movidius, as supporting Myriad2 various processors. The Movidius compiler now supports fully the C/C++ language with various libraries included. The operating system on Myriad2 is RTEMS, which provides a multi-threaded, multi-tasking environment for application's threads sharing the same memory space. Within RTEMS each subsystem is implemented as an independent manager (task, interrupt, clock, semaphore …). In the system, required managers and applications are linked to one binary image. The system includes also TCP/IP networking and local file-systems support.

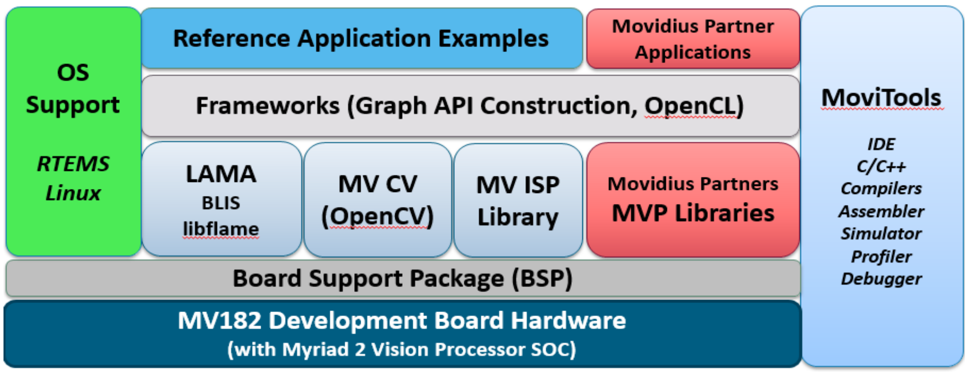

OS Support
RTEMS
Linux
Reference Application Examples
Movidius Partner Applications
Frameworks (Graph API Construction, OpenCL)
LAMA
BLIS
libflame
MV CV (OpenCV)
MV ISP Library
Movidius Partners MVP Libraries
Board Support Package (BSP)
MV182 Development Board Hardware
(with Myriad 2 Vision Processor SOC)
MoviTools
IDE
C/C++
Compilers
Assembler
Simulator
Profiler
Debugger

Figure 18: Myriad2 software development tools and environment

Programming paradigms

According to different hardware and operating system's availability, the programming paradigms of Myriad2 platform can be classified into three types, shown in Figure xx-xx respectively as follows.

Standard programming paradigm

The standard programming paradigm for Myriad2 involves using RTEMS running on LeonOS and the SIPP scheduler on LeonRT. The advantage of this paradigm is that it provides parallelization in an easy to use environment. The SIPP scheduler itself is able to ensure parallel pipeline configurations for managing the HW filters and exterior interfaces with a low footprint so as to ensure LeonRT optimized utilization. The SIPP used number of SHAVEs is configurable, so any extra number of SHAVEs not used for line based pipelines will remain free to be used by the RTEMS operating system running on LeonOS for various other purposes including (but not limited to) computer vision algorithms.

One Leon programming paradigm

Some applications might not require heavy line based processing. Such applications might choose to completely switch OFF the LeonRT processor and instead only use LeonOS with (or without) RTEMS. HW filters may still be used. Using this programming paradigm, as shown in Figure 19, LeonOS would control all of the applications running on the 12 SHAVE cores.

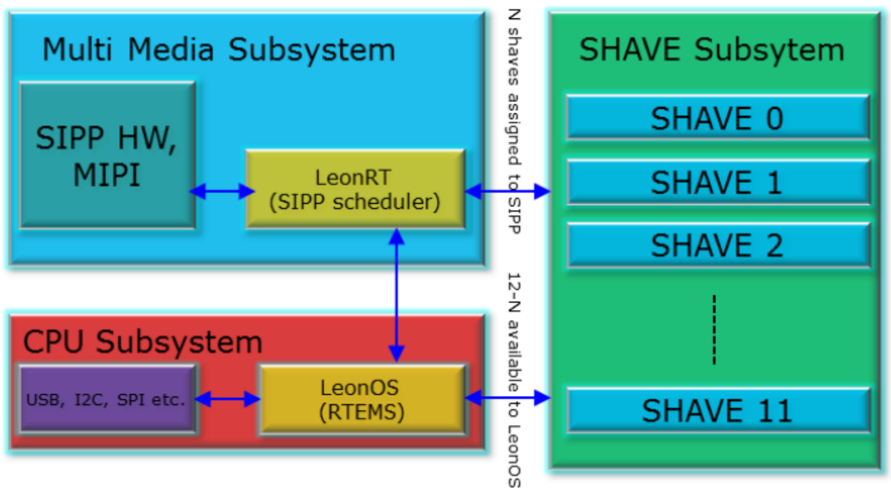


Figure 19: Myriad2 standard programming paradigm

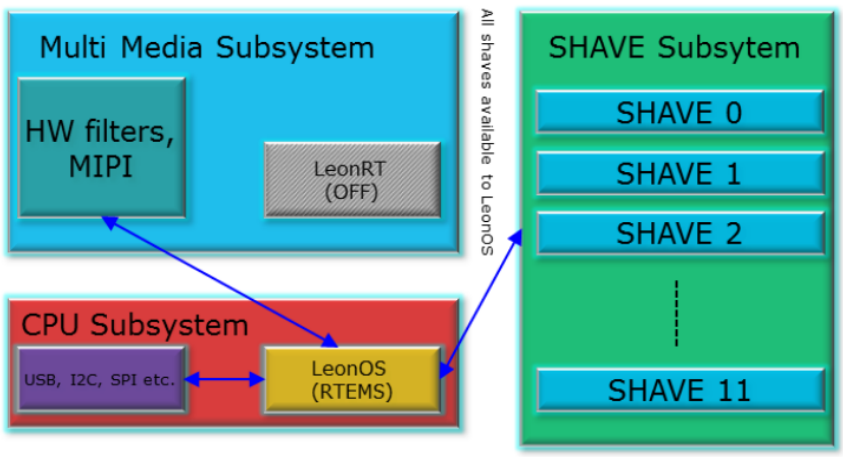


Figure 20: Myriad2 one Leon programming paradigm

Bare metal programming paradigm

A bare metal programming paradigm (cf. Figure 21) will also be supported by the MDK build system. This will allow the developer to use both LEON cores without any operating system, only minimal schedulers running to control the pipelines application. This paradigm requires more integration efforts but allows developers to write applications which will not be affected by any operating system overhead.

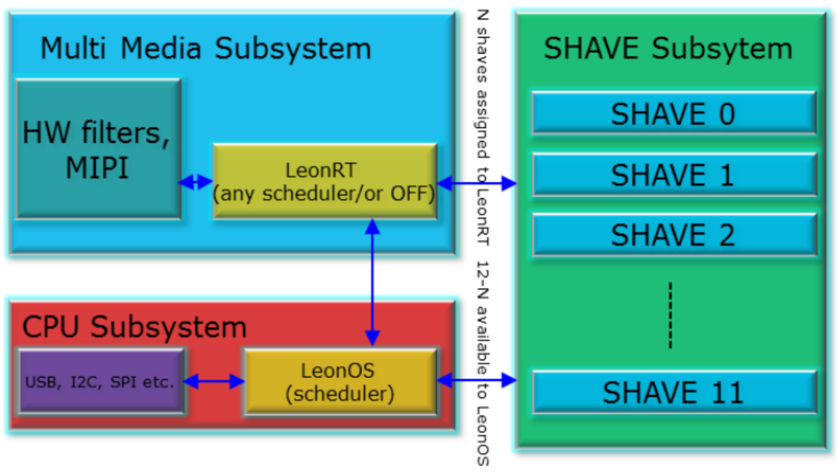


Figure 21: Myriad2 bare metal programming paradigm

According to users' requirements and sensors availability on Myriad2, we implement several functions for temperature and power monitoring and provide APIs for code instrumentation of a generic Myriad2 application.

The board-specific metrics that are supported by the Monitoring Framework are listed in Table 15.

Table 15: Metrics supported for Myriad2 platform

Type	Metrics	Units	Description
Power	power_core	mW	Power consumption of the cores and processors
	power_ddr	mW	Power consumption of DDR memory
Temperature	temperature_CSS	°c	Temperature of the CPU SubSystem (CSS), equals to the Leon RSIC2 processor
	temperature_MSS	°c	Temperature of the Media Sub System (MSS), equals to the Leon RSIC1 processor
	temperature_UPA0	°c	Average temperature of half of the Microprocessor Array (6 VLIW SHAVE vector processors)
	temperature_UPA1	°c	Average temperature of the other half of the Microprocessor Array (the other 6 VLIW SHAVE vector processors)

The power monitoring is achieved based on the MV0198 power measurement daughter-card, which is composed of 4 ADCs and samples 13 power rails and 2 voltage rails of the Myriad2 motherboard MV0182. With the enabled I2C interface to the motherboard and drivers provided APIs for reading the measurements, we implement functions in C to initialize the MV0198 driver and gather measurements periodically.

For temperature monitoring, we use the temperature sensor library provided in the Leon RSIC2 operating system. As can be seen from **Table 2**, temperature metrics are divided into four parts, which can be mapped to accordingly different hardware components.

For users convenience, we design and implement the application-level APIs for Myriad2 platform same as these for other platforms. Listing below gives an example shows how to use these APIs in a Myriad2 application.

Listing : Example of a Myriad2 application with integrated monitoring APIs

```
void POSIX_Init(void *args)
{
    UNUSED(args);
/* NEED FOR USING ETHERNET */
    initClocksAndMemory();
    EthPHYHWReset();
    InitGpioEth(INVERT_GTX_CLK_CFG);
    initGrethAndNet();
/* SETUP METRICS */
    metrics m_resources;
    m_resources.num_metrics = 2;
    m_resources.local_data_storage = 0;           // TOD: local data storage
    m_resources.sampling_interval[0] = 1000;    // 1s
    strcpy(m_resources.metrics_names[0], "power_monitor");
    m_resources.sampling_interval[1] = 1500;    // 1.5s
    strcpy(m_resources.metrics_names[1], "temp_monitor");
/* START MONITORING */
    mf_start("141.58.0.8", "movidius", &m_resources);
/* DO THE WORK */
    sleep(15);
/* STOP MONITORING */
    mf_end();
    exit(0);
}
```

Function “mf_start” creates threads for monitoring power and/or temperature, which are configured by setting the metrics’ names and sampling intervals. Each thread runs in a loop, samples the specialized metrics, and uses the board’s network stack to send the sampled metrics to the server. After execution of the target code block, “mf_end” terminates the running threads and ends accordingly the metrics sampling process. It is noted that the monitoring data are sent to the server via HTTP and hardware sockets, therefore programmers should initialize and configure the system’s Ethernet module properly before using the provided APIs.

APPENDIX 4. SECURITY AUDITING SPECIFICATION

In this appendix is described the functions and auditable events of the security auditing API. The Flexibility of the functions is achieved with the ability to save audit specific data for different types of events, for both system-level software and application level.

The Security of the audit records and logs generated by the next functions and auditable events environment are assumed to be protected from unauthorized access and tampering by the operating environment. Concerns about the log that are justified in a particular deployment may be addressed with one or more measures, such as disk or file system encryption, transmitting the log data to a different host over a one-way (except for handshaking) communication path or recording in a blockchain ledger.

The channels used to communicate audit data are assumed to be free from ease dropping and tampering. In a particular deployment if it is determined that the channels cannot be trusted then confidentiality and integrity controls can be implemented at the communication level.

System Auditable Events (this is the minimal list and will be added to)

The System Auditable Events can be generated by both system-level software and application auditing levels The auditable events listed below, are the minimal list to be supported and others may be added during the implementation.

sys_audit_start
sys_audit_stop
sys_audit_log_assign
sys_audit_monitor_assign
sys_authentication_attempt

Functions of the security auditing API for the Application level:

The functions for supporting the security auditing instrumentation and their usage syntax are brief described below.

- Audit record of an app event metric. It can be related to an auditable system event or an app defined event.

  monitor_server_call (<monitor type>, <monitor channel>, <monitor data>)

  <monitor type> is an enumerated or atomic value: metric, sys_audit, app_audit, …

  <monitor channel> is a differentiator within each <monitor type> such as
  <metric name> for monitor type metric
  <auditable event name> for monitor type sys_audit
  <app defined identifier> for monitor type app_audit

  <monitor data> is a numeric value or a string (or a numeric value coded as a string) depending on <monitor type> and <monitor channel>.

that in turn call monitor_server_call:
monitor_server_call(sys_audit, <auditable event name>, <event-specific data>)
or
monitor_server_call(app_audit, <app-defined event>, <event-specific data>)
respectively.

- Definition of the location of the storage of the records. This classification of the stored data facilitates its future management. Different analysis or users will have interest in different sets of data.

  monitor_log_assign(<monitor type>, <monitor channel>, <log location>)

  It assigns <log location> to be the new destination for monitor records for the particular <monitor type> and <monitor channel>.

  <monitor channel> can be assign to "all" when it is wished to send all channels for the given monitor type to the same location.

  This function generates a sys_audit_log_assign event to old (if active) and new locations.

- Assignment of a function to process the calls for a give monitor type.

  monitor_handler_assign(<monitor type>, <handler>)

  registers <handler> to be invoked by the monitor server to process received call for the given monitor type.

  This function generates a sys_audit_monitor_assign event to the log.

- Convenience functions, I NOT UNDERSTAND ITS PURPOSE

  sys_audit_gen(<auditable event name>, <event-specific data>)

  app_audit_gen(<app-defined event>, <event-specific data>)

Functions of the security auditing API for the System level:

The functions defined above audit security only during the execution of applications.

Additionally, in the PHANTOM project, a system-level security auditing system is also defined, which also allows the security record of the system to be maintained when no audited application is being executed. The security record at system level can be active during the entire operating time of the system.

The System audit-specific interfaces (some may apply also to application audit) and their usage syntax are brief described below.

- Request system audition for a specific event list

  audit_start(<audit selection>, <log location>)

It causes auditing of the auditable events in <audit selection> to be started to <log location>. Logging appends to the log location if it already exists. Generates a sys_audit_start event to the log.

- Request end of any running system audition

  audit_stop( )

  It causes auditing to stop. Generates a final sys_audit_stop event to the log.

- Request replacing the list of events being audited.

  audit_select(<new audit selection>, <add, remove, replace>)

  It changes the current audit selection with the <new audit selection> by adding, removing, or replacing according to <add, remove, replace> argument.

- Setting an alarm configuration for a specified pattern.

  audit_alarm(<pattern>, <alarm handler>)

  The audit generation facility will test for the specified <pattern> and will invoke the <alarm handler> if the pattern is recognized. The <pattern> is, at a minimum, a set of auditable events in the same form as an audit selection. The intersection of the set of events in the pattern and those in the current audit selection determines the events actually considered to generate an alarm. The function may be used to reset the remembered pattern and handler to the default values (typically "none").

- Audit generation.

  Audit generation – see sys_audit_gen interface defined above.
  The sys_audit_gen function only accepts requests from security-relevant system software. This prevents untrusted software from overwhelming the audit service and thereby causing a denial of service. The sys_audit_gen function qualifies whether the first argument <auditable event name> is in the list of events currently selected for auditing. If so, a time stamp is generated, and an audit record is generated with the following information and placed into the audit buffer:
    Audit event name
    Source of the event
    Timestamp
    Event-specific data
  Then the audit generation function tests whether the current qualified event completes the current alarm pattern currently being monitored and if so invokes the current alarm handler.

- Audit Daemon.

  Audit Daemon – empties the audit buffer in the monitoring server when it is full or after a configurable interval of time since the last time the buffer was emptied.