PHANTOM

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

Project Number 688146

D4.1 – Available tools for heterogeneous infrastructure setup and software stack development

Version 1.0
9 September 2016
Final

Public Distribution

University of York, Easy Global Market, GMV, Intecs, The Open Group, University of Stuttgart, Unparallel Innovation, Wings ICT Solutions

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	First Draft; Proposed TOC	21st July 2016
0.5	Updated content	8th August 2016
0.8	Ready for Review	15th August 2016
0.9	Final version	31st August 2016
1.0	Final QA updates	9th September 2016

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document constitutes deliverable D4.1 – Available tools for heterogeneous infrastructure setup and software stack development, developed in WP4 of the PHANTOM project. It is the first deliverable of WP4, and presents a set of tools studied and analysed, which can be useful as a baseline for all further developments of WP4.

The content of this document results from the background study of tools that can be used for the development of PHANTOM platform. As such, this deliverable receives valuable input from deliverable D1.2 – First design for Cross-layer Programming, Security and Runtime monitoring, where the PHANTOM toolflow and the functional modules involved in it are described, with the role of each module of the toolflow being well-defined and the main interaction between them identified. This document uses that information to identify functionalities required in each step of the toolflow and seeks to match these with the tools available.

All the tools were analysed against PHANTOM toolflow, and are presented in this document taking into consideration the functional module in which the features of the tools correspond to specific features of the different modules - Systems Modelling, Programming Model, Parallelisation Toolset, Repository, Multi-objective Mapper, Runtime Monitoring, Security Execution Platform, MBT Tools and Deployment Manager - of the PHANTOM toolflow.

1. INTRODUCTION

An ever increasing number of applications start to require more and more computational power mainly due to the increase of the amount of data that must be handled and to the increase in the complexity of the algorithms used to process data. However, processor working frequency and corresponding single core performance have not been able to satisfactorily meet most of their computational needs, leading to the development of multicore processors and multiple processor systems. This situation has lead to an increase of interest on techniques to explore and efficiently take advantage of such systems, resulting on the continuous development of tools and platforms dedicated to address several aspects of the parallelization of applications and processes. New tools are always arising and existing tools are under constant development, where new features are added with regular frequency.

Before the concretisation of any R&D project it is always considered good approach to perform a background study about the techniques and tools that are already developed and available for use. This step assumes even more relevance when the subject of study has a continuous and frequent development as is the case of parallel computing. This document aims to perform this background study by identifying the most relevant parallel computing tools and comparing PHANTOM toolflow defined in deliverable D1.2 with the available tools, trying to identify which tools can, in potential, be used to implement some of the required behaviour/functionalities. At the same time, this document also focuses on the identification of which parts of PHANTOM are not fully/properly covered by the existing tools.

1.1 PHANTOM TOOLFLOW

PHANTOM toolflow is defined and detailed in deliverable D1.2. In this document, a quick overview will be presented to support the analyses of the identified tools. Figure 1 shows an overview of the PHANTOM toolflow, where PHANTOM functional blocks and the information flow between them are presented. The flow starts with developers using the PHANTOM Programming Model to describe their system in terms of parallel interacting components. These components are stored in the Repository, around which PHANTOM development is centred. For example, the parallelisation and optimisation stages take components and associate them with available hardware platforms to create parallelised components and optimised system deployments.

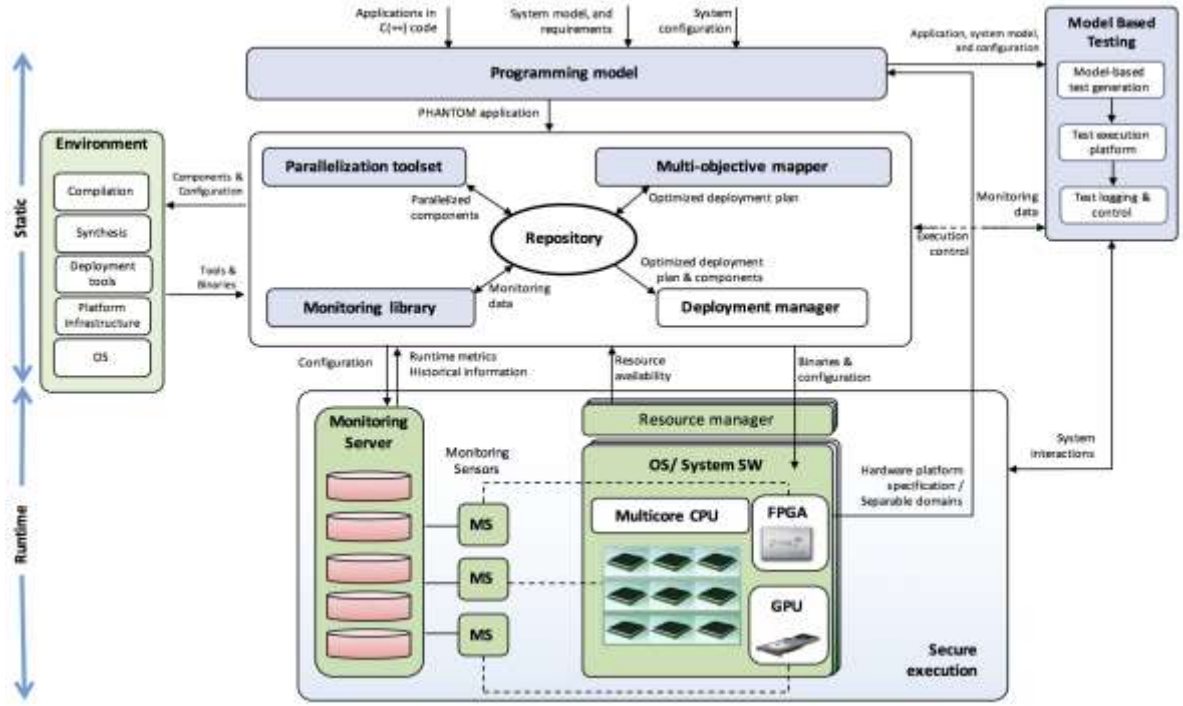


Figure 1: The PHANTOM toolflow

The toolflow handles different types of inputs: metadata describing the software components (System configuration), model describing non-functional requirements (System model) and description of target system platform (Hardware platform specification). Based on those inputs, and along the application specification, the programming model will be used to implement the functionalities of each component as well as express the data and control flows, generating a PHANTOM application.

The PHANTOM application is then stored in the Repository module with the corresponding modelling specification, making it accessible to other PHANTOM modules. From this point, the Multi-objective mapper analyses the application modelling specifications and decides which target platform(s) (CPU, GPU, FPGA, etc.) should be used to run the parallel application as well as which parallelisation techniques should be used (e.g. which data sharing mechanism should be used), compiling all these decisions in an optimized deployment plan that is stored in the Repository. These decisions are then executed by the Parallelisation toolset, which generates the parallelised components specific for each target platform and stores them in the Repository. The Deployment manager will follow the deployment plan defined by the Multi-objective mapper, fetch the parallelised components stored in the repository, generate the corresponding platform specific implementations and perform their deployment on the corresponding target platform.

The execution of the PHANTOM applications will be evaluated in run-time by the Monitoring Framework that performs a set of measurements (both platform default and developer-defined metrics) to assess its runtime performance. This information will be of extreme importance to support the decisions of Multi-objective mapper. The metrics will also be relevant for the Model Based Testing module, designed to perform black box testing for application, assessing both functional and non-functional properties of distributed and parallel computing.

Alongside with these functional modules there are two other modules: Environment and Secure Execution. The Secure execution module will implement the PHANTOM security model on the target platform in order to provide a certain level of security on heterogeneous execution platforms. The environment comprises a set of tools and platforms required for the functioning of other PHANTOM modules. This includes, but is not limited to: hardware specific compilers, FPGA synthesizer, operating systems, etc.

1.2 DOCUMENT OUTLINE

The main aim of this document is to identify the tools that have the potential of providing some of the functionalities needed for the PHANTOM platform. Therefore, more important than the identification of the tools, is the identification of how and when can these tools be used in PHANTOM toolflow. As such, the main body of this document will be focused on the analysis of the tools against PHANTOM toolflow, being the analysis made by functional module. This study is presented in Section 2 - "Analysis of Tools", being divided in nine subsections corresponding to different modules of PHANTOM toolflow:

- Systems Modelling - refers to the requirements to model software and hardware systems, as well as application requirements and specifications;
- Programming Model - relates to the capabilities that PHANTOM Programming model needs to have to be able to express the parallel and distributed computing specificities;
- Parallelisation Toolset - corresponds to the set of tools that will be used to generate the parallelised components;
- Repository - repository of XML files with system descriptions and specifications, as well as, parallelised components and their implementations;
- Multi-objective Mapper - refers to the module responsible for the decision of the most appropriated deployment plan of a given PHANTOM application;
- Runtime Monitoring - framework able to measure several metrics (both predefined and user-defined) during the runtime of the PHANTOM application, running on a heterogeneous execution platform;
- Security Execution Platform - refers to a set of techniques and tools to provide a specific level of security to applications running on heterogeneous execution platforms;
- MBT Tools - set of tools to enable the developers to assess several functional and non-functional requirements by testing PHANTOM applications in a black box approach;
- Deployment Manager - refers to the module responsible for the execution of the deployment plan developed by the Multi-objective mapper, and to the implementation and deployment of the parallelized components in the different target platforms.

In Section 3 are presented the conclusions, where are summarised the conclusions drawn in the analysis of each functional module, being identified which tools can be used, and for what. In this section is also identified the functionalities that the identified

tools fail to cover, and therefore, must be entirely developed by the PHANTOM consortium.

In annex is provided a list with all tools identified during the development of this document that showed some potential related with the implementation of the required by the PHANTOM toolflow. For each tool is provided with a short description and the identification of some of its main characteristics.

2. ANALYSIS OF TOOLS

2.1 MODELLING

PHANTOM toolflow requires as input the description: of the target hardware platform being identified and characterised the resources available and their constraints; of the current system configuration allowing to specify the software component used by the developer and their versions, as well as to define initial deployment restrictions for each application component, defining where in the hardware platform can be, or not, deployed; and of the system functional model and non-functional requirements, required for the proper execution of the testing and monitoring processes.

In PHANTOM, these descriptions will be provided according to one or a set of models designed to provide an accurate and complete information regarding the required hardware, software and application characteristics. In a first approach, these models will be input in PHANTOM in XML format. However, it may also be considered the option to input this information directly into PHANTOM platform using a modelling tool built using Eclipse Modeling Framework (EMF) and its meta model (Ecore) [42].

Three modelling languages were identified as they provided mechanisms to support the modelling of several types of systems, such as real-time and embedded systems, supporting the specification, analysis, design verification and validation of such systems. These tools are SysML [1], UML MARTE [2] and MADES [3].

SysML extends UML 2 standard, which is mainly focused on modelling of the software aspects, adding mechanisms to support the modelling of hardware, software. UML MARTE also extends the UML standard with the aim of providing mechanisms for the modelling of real-time and embedded systems, allowing the specification of several parameters like throughput, bandwidth, delay and memory usage; and supporting a language for the definition of algebraic and time expressions. Moreover, UML MARTE defines concepts related with the allocation of applications on platforms and semantic support for concurrency, communication, and time-constraint aspects. The features introduced by SysML and UML MARTE were added to the UML standard as profiles, both being used by MADES to support several techniques such as code generation and model transformations to support design, validation, simulation and automatic code generation.

Another alternative that may be considered for the system modelling is the usage of Architecture Description Languages. These languages have graphical and/or textual representations to describe and represent system architectures. They are intended to provide a human and machine readable formal representation of the architecture, supporting the execution of system analysis and the automatic generation of system

prototypes. In the context of PHANTOM, the Architecture Analysis and Design Language (AADL) assumes special relevance since it is intended for modelling of embedded and real-time systems, allowing to describe both the software and hardware architecture of a system, supporting the description of a large range of features and components such as processes, threads, data, processors, devices, memory, buses, etc. A large range of tools[1] is currently available to support AADL's toolchain; supporting the design, checking, analysis of systems models as well as code generation. Moreover, AADL supports the XML/XMI interchange format to make easier interoperability with other tools and platforms.

2.2 PHANTOM PROGRAMMING MODEL

Currently, there are several programming environments to handle the task of program parallelisation. Either to tackle specific application areas like large-scale systems and Big Data, where programming models like Hadoop [4], Spark [5], Storm [6] play a major role; or in the case of general purpose programming models like OmpSs [7], OpenMP [8]. Hadoop, Spark and Storm are dataflow programming models that target homogeneous clusters, which impose a very specific execution flow to programmers such as the MapReduce. OmpSs and StarPU are designed to address uniform clusters, being data dependencies set up during runtime. This behaviour allows systems to be dynamic but, at the same time, these program models do not provide mechanisms to pre-define the task graph of the system. This inability results in systems that cannot be easily engineered to meet non-functional requirements like timing, communication costs, or to guarantee a security model.

Moreover, Hadoop, Spark, Storm, OmpSs and OpenMP are all data-oriented programming models which purposely limit the possible inter-task coordination in order to make this task easier for the developer. This behaviour is not compatible with most of the embedded use cases where is required control-oriented programming to allow tasks to run freely and coordinate at set points.

These subjects related with the description and requirement of PHANTOM Programming Model, as well as the limitations of some programming models are presented and discussed in more detail in D1.2 - "First Design for Cross-layer Programming, Security and Runtime monitoring".

2.3 PARALLELISATION

As analysed in deliverable D1.2, the main operations of the Parallelisation toolset are the Code analysis and the Techniques selection. The code analysis operation first creates a structural representation of the source code (e.g. parsing using tools such as ANTLR [9], Clang [10]) and then it simplifies (using tools such as CodeRush [11], AutoRefactor [12]) the source code in order to further facilitate the analysis. The simplified source code is tested for data dependencies (with tools such as COINS [13], CETUS [14]), which could prevent the components parallelisation, while the parallelisable loops-variables attributes are transformed with specific PHANTOM directives. The analysis outcome is the parallelisation annotations/directives which provide the maximum number of possible parallel components to the Multi-objective

[1] https://wiki.sei.cmu.edu/aadl/index.php/AADL_toolshttps://wiki.sei.cmu.edu/aadl/index.php/AADL_tools

mapper. The technique selection receives the deployment indication and further decides on the best parallelis [15]ation API to be used for the parallelisation technique (e.g. OpenMP threads communication, MPI, etc.).

In the context of PHANTOM, a program consists of a set of components that are executed in parallel. PHANTOM parallelisation toolset picks the components provided by developers, and uses them to implement two different parallelisation approaches: component parallelisation, where the different versions of the component are generated to take advantage of the heterogeneous computing elements available in the system (FPGAs, GPUs, SMPs, etc.); and component replication which allows PHANTOM to generate multiple copies of an entire component in order to allow put them running on different "slices" of input data. Therefore, PHANTOM toolset must support and use tools that provide it with the capabilities to generate CPU, GPU and FPGA implementations of components.

2.3.1 SMP

Symmetric MultiProcessing (SMP) systems are characterised by being composed by two or more identical processors, considered at the same functional level by the software. These processors have access to the same resources, memory, and are controlled by the same operating system instance. Many parallelisation tools and programming models have been proposed in recent time. While some of these parallelisation tools are designed to provide general purpose programming models, others aim to tackle specific application use cases, such as Big Data scenarios.

Two programming models are presented by the Khronos Group: OpenCL [16] and SYCL [17]. OpenCL is originally a C programming model, whose support to C++ language was also added to in the most recent version. One of the key point of OpenCL is portability, being able to produce code to run on CPUs, GPUs and FPGAs and supported by the major manufacturers. Portability is achieved through the definition of OpenCL-coded routines, called kernels, that will be used to generate code to run on the different devices (CPUs, GPUs and FPGAs). As such, OpenCL kernel matches to some extent the definition of PHANTOM components, differing mainly in the fact that all kernels must be defined on file, where is also described the host program that executes the dispatchment of kernels. Data can be shared between hosts application and kernels by being set up a Shared Virtual Memory that defines an address space to both host and heterogeneous devices. OpenCL provides mechanisms that support the definition of both data flow and control flow parallelisation as required in the PHANTOM programming model. SYCL is an abstraction layer in compliance with the standard C++ that uses the underlying concepts of OpenCL, being fully compatible with OpenCL and permitting OpenCL code reuse. SYCL follows a single-source approach enabling the definition of hosts and kernels in each function. Both OpenCL and SYCL code is translated to an intermediate language - SPIR-V [18] - that will be interpreted by the specific OpenCL runtime driver deployed in each device. The usage of this intermediate language removes the need to recompile OpenCL code for each target platform, being the translation to the hardware language performed in the device driver.

StarPU consists of runtime system design to provide support to heterogeneous multicore architectures (CPUs and GPUs) by taking care of the mapping and execution of tasks on heterogeneous devices while transparently addressing some low-level issues (e.g. data

transfers). StarPU implements the concept of kernel in a data structure called codelet, describing a computational task that can be implemented for multiple architectures. StarPu also allows to define and run tasks that are composed by a codelet, the data set and the description of how data will be used by that codelet. Tasks are asynchronous, being the operation of task submission non-blocking. This feature may be exploited to implement the concept of PHANTOM components. StarPU provides a high-level data management library, enforcing memory coherence over the machine. By default, StarPU gives priority to data replication instead of data transfer for the sake of performance. STARPU can be integrated with MPI support to clusters combining it with intra-node data transfer and computation.

SMP involves by nature cost-effective ways to support parallel programming. Also, not only can the uniprocessor system be used in a SMP, but it can be scaled for data sharing. However, the scalability of SMP due to cache coherence and shared objects still remains under specific limits. PHANTOM can benefit from the several existing programming models. As anticipated, OpenCL kernel matches to some extent the definition of PHANTOM components, while provides mechanisms that support both data flow and control flow parallelisation. On the other hand, StarPu can be exploited for PHANTOM components implementation due to the capability to support asynchronous tasks.

2.3.2 GPU

Graphics Processing Units (GPUs) can be used to accelerate typical sequential programs by accelerating specific functions. However, for a real improvement of an application execution speed, these functions must be able to take advantage of the GPU main features: large number of small, specialised, and energy efficient cores; and higher memory access speed than CPUs but with less memory amount available.

Several parallel computing tools support the development of parallel applications targeting GPU platforms, while most of them support simultaneously the CPU and GPU parallelisation. Probably the most used is OpenCL from Khronos Group, supporting not only cross-platform applications but also all of the major GPU vendors. This feature is shared by other tools from this group - SYCL and Vulkan [19] - since all of them are converted to the intermediate language SPIR-V that is supported by both CPUs and GPUs runtime drivers. Vulkan consists of a graphics and compute API in C, whose approach is to provide more control to the developer in order to make drivers simpler and with lower bug potential. As such, developers must control many low-level aspects of the application (like memory allocation, thread management, etc.), which allows applications to have more predictable behaviours and better performance.

NVIDIA proposes CUDA [20] as a general-purpose processing platforms for GPUs. CUDA supports several programming languages (like C, C++ and Fortran), and provides a compiler for OpenACC API [15]. However, CUDA supported devices is limited to NVIDIA GPU cards, and not to every card since not all models and versions support CUDA.

The use of GPUs can offer a real improvement of an application execution speed and higher memory access speed than CPUs for some specific application functions when designed appropriately. OpenCL supports not only cross-platform applications but also all of the major GPU vendors, while may be combined with other tools like Vulkan

enabling applications to have more predictable behaviours and better performance. CUDA supports general-purpose computation on GPUs (GPGPU) using graphics APIs, while offering a number of advantages compared to traditional approaches, like scattered reads from arbitrary addresses in memory, unified memory and fast shared memory among threads enabling higher bandwidth, faster downloads and readbacks to and from the GPU, full support for integer and bitwise operations, and others; however, it is limited to NVIDIA GPU cards.

2.3.3 FPGA

Reconfigurable computing devices can increase the performance of compute intensive algorithms by implementing application specific co-processor architectures. The power cost for this performance gain is often an order of magnitude less than that of modern CPUs and GPUs. Exploiting the potential of reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs), is typically a complex and tedious hardware engineering task. Traditional methods for development on FPGAs require a hardware architecture to be designed using a Hardware Description Language (HDL). This requires the developer to design data paths, control logic, interface with Intellectual Property (IP) hardware cores, and deal with timing constraints. Alternatively, high level design tools attempt to remove some of the tedium in FPGA development. Using these, a developer may instead specify an architecture's behaviour at the algorithmic level in some high level programming language. The Compilation process will then extract data paths, state-machines, process schedules etc., and automatically generate an HDL architecture specification. Recently the major FPGA vendors (Altera, and Xilinx) have released their own high-level design tools, which have great potential for rapid development of FPGA based custom accelerators.

Vivado High Level Synthesis (HLS), released by Xilinx, is the most popular commercial HLS tool in market. This tool enables the development of IP cores using standard C code, which is subsequently mapped to an RTL architecture (either VHDL or Verilog). Internal C function code is synthesized into RTL statements which operated over a number of clock cycles. The function signature is mapped to RTL ports using standard protocols such as Buses, FIFOs, and RAM. The use case for the Vivado HLS is the rapid development of IP cores that typically need to be integrated into a larger system design. As a result, the Vivado HLS is primarily targeted at improving development abilities of hardware engineers who have sufficient expertise in both software and hardware design.

On the other hand, Altera has released an OpenCL Software Development Kit (SDK) for certain supported Stratix V PCIe cards. The Open Computing Language (OpenCL) is an open standard for parallel programming on heterogeneous compute systems comprising of a device side parallel programming language, and a host side C Application Programming Interface (API). While OpenCL implementations primarily exist for CPU and GPU architectures, the Altera OpenCL SDK is an implementation of the OpenCL embedded profile for FPGAs. The device side OpenCL compute kernel code is translated to a Verilog IP core which is then used to generate a FPGA configuration image. The Host side OpenCL API can then be used to configure and control the FPGA for data processing. In contrast with the Vivado HLS the OpenCL SDK permits the development of FPGA-based applications by software engineers with minimal hardware experience.

OpenCL-based high-level design, which employs OpenCL as the programming language, is a relatively new methodology for application design on FPGA platform. OpenCL is an open standard for parallel programming of heterogeneous systems which can consist of CPU, GPU, DSP, FPGA, etc. For a typical SoC-FPGA-based embedded system, the system is usually composed of ARM core, memory and accelerators that are generated by programming logics. If we consider the ARM core being the host and accelerators being the devices, the SoC system can be deemed to a heterogeneous system. Therefore, utilizing OpenCL as the high-level design language for SoC-FPGA is also promising.

Both Vivado HLS and Altera SDK for OpenCL can generate efficient hardware architecture with much shorter developing time than non high-level design methods. But the tools themselves and development experiences with them are actually quite different. Vivado HLS is a tool that used to speedup hardware accelerator (IP) design and creation on FPGA. The generated file is IP core, which can be connected to other IP or ARM in Vivado system design. Therefore, some work should be done in Vivado as well and hardware knowledge is necessary for the user. Besides, more directives, such as memory type, interface, loop control, etc., are supported in Vivado HLS. By contrast, Altera SDK for OpenCL is a tool provided for software programmers, which means little hardware knowledge is required for user and much easier workflow for the whole process. However, comparing to Vivado HLS, less directives are supported and is more difficult to debug applications. Of course, OpenCL knowledge is required for the programmer.

Additionally, there is OpenSPL [21] an open specification for a Spatial Programming Language. The operations of a spatial program exist in space rather than as a sequence of operations over time. This means that all of the operations of program can happen at once and that the notion of what it means to execute a program is more about getting the data in and out of the system as opposed to the sequence of events that take place in typical procedural languages. Conventional programs execute in 1 dimension, where time progresses forward following the instruction sequence. Spatial programming is programming in 2 dimensions, where data progresses forward in parallel across the fabric of an array or chip (e.g. FPGA). The major objective of Spatial Programming and OpenSPL is to maximize the amount of computation per cubic foot of datacenter space. For suitable applications Spatial Programming can achieve one to two orders of magnitude improvement in computational density, thereby providing the same advantage in power consumption.

The massively parallel architecture of the FPGA provides a platform for the development of low power and power efficient (i.e., FLOPS/Watt) application specific accelerators allowing the development of High Performance Computing (HPC) with a low power footprint.

PHANTOM can explore FPGAs advantages by creating a repository for specific algorithms that are commonly used and are known to have better performance running on FPGAs, such as FFT, FIR filters, etc. These accelerators can then be used to do the complex computations in parallel instead of the corresponding sequential CPU code. Furthermore, Vivado HLS tools can be used to synthesize parts of the application code, which are identified by a set of metrics, to be parallelizable and run faster on FPGAs.

So not only the precompiled functions but also other parts of the application code can be implemented for FPGAs and exploit application specific co-processor architectures.

2.4 PHANTOM REPOSITORY

PHANTOM supports iterative development based around a central Component Repository. The repository stores the user's components (including alternate versions of those components generated by the parallelization toolset), and metadata describing the target platform, the mapping of components to the platform, runtime monitoring data, etc.

The MOM will read the Platform Description in order to deploy the component network over the hardware. The platform description carries high-level information about the hardware, such as the CPUs, memory spaces, and communications channels in the system, and some simple metrics about each one. The goal of this description is to allow PHANTOM to more easily handle parameterisable architectures. The platform description uses XML.

The Component Network consists of a set of deterministic components communicating asynchronously or synchronously through communication objects. These objects have defined protocols of queues, shared memories and signals. This is represented internally in the Component Repository using an XML interchange format, from which a high level network specification can be extracted.

When the developer invokes the Multi-objective mapper, it attempts to map the component network over the platform description. The result of this mapping process is a deployment. Deployments are represented in the repository in the same XML-based format.

The PHANTOM repository will be organized as follows:

- C/C++ files (description and components' code)
- XML files (description of platforms, high level description of components, initial setup, etc.);
- Relational Databases (Runtime monitoring data);
- Versioning control systems (Source code files and binaries).

2.4.1 Databases

2.4.1.1 Document-Oriented

A document-oriented database, or document store, is a computer program designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of NoSQL databases. XML databases are a subclass of document-oriented databases that are optimized to work with XML documents. Graph databases are similar, but add another layer, the relationship, which allows them to link documents for rapid traversal.

Document-oriented databases are inherently a subclass of the key-value store, another NoSQL database concept. The difference lies in the way the data is processed; in a key-value store the data is considered to be inherently opaque to the database, whereas a document-oriented system relies on internal structure in the document in order to extract metadata that the database engine uses for further optimization.

Document databases contrast strongly with the traditional relational database (RDB). Relational databases generally store data in separate tables that are defined by the programmer, and a single object may be spread across several tables. Document databases store all information for a given object in a single instance in the database, and every stored object can be different from every other. This makes mapping objects into the database a simple task, normally eliminating anything similar to an object-relational mapping.

XML databases can be used in PHANTOM to store the XML files that describe the platforms, components, relations between components employing the defined protocols of queues, shared memories and signals, the initial setup parameters and configuration of the runtime monitoring.

BaseX, EXist and Berkley DB XML are some of the most known and used XML databases. In contrast to other document-oriented databases, XML databases provide support for standardized query languages such as XPath and XQuery. Both are mature, well documented, well understood query languages.

BaseX is a NoSQL light-weight XML database management system and XQuery processor. It is specialized in storing, querying, and visualizing large XML documents and collections. It's a schemaless database so as long as the documents are well formed, it can store any XML document, regardless of the XML schema. BaseX is highly conformant to World Wide Web Consortium (W3C) specifications and the official Update and Full Text extensions. The included GUI enables users to interactively search, explore and analyze their data, as well as evaluate XPath/XQuery expressions in realtime, and the clients offered across most common and current languages make working with BaseX straightforward. BaseX can be extended via modules too, adding extra features to the database itself as well as enhancing XQuery itself.

eXist is an open source NoSQL database and application platform built on XML technology. It is classified as both a NoSQL Document database and a native XML database (and it provides support for XML, JSON, HTML and Binary documents). Unlike most RDBMS and NoSQL databases, eXist provides XQuery and XSLT as its query and application programming languages. eXist allows software developers to persist XML/JSON/Binary documents without writing extensive middleware. eXist follows and extends many W3C XML standards such as XQuery. eXist also supports REST interfaces for interfacing with AJAX-type web forms. Applications such as XForms may save their data by using just a few lines of code. The WebDAV interface to eXist allows users to "drag and drop" XML files directly into the eXist database. eXist automatically indexes documents using a keyword indexing system.

Finally, the Berkeley DB XML database specializes in the storage of XML documents, supporting XQuery-based access. It is built on top of Oracle Berkeley DB and inherits its rich features and attributes. The Berkeley DB XML provides a document parser,

XML indexer, and XQuery engine on top of Oracle Berkeley DB to enable the fast and efficient retrieval of data. Applications perform database administration, eliminating the need for a DBA and allowing continuous, unattended operation.

In PHANTOM, the usage of XML databases may improve the performance of PHANTOM during the process of data extraction from the XML files. XML databases have optimised internal query engines that may be used to read the descriptions inside XML files, providing better speeds than the methods using file parsing. The real existence of such improvement is something that must investigated during the project, since it will depend on the number and size of the XML files used. Only then will be possible to identify if the use of XML databases is really a useful asset for the PHANTOM repository.

2.4.1.2 Relational

A relational database is a digital database whose organization is based on the relational model of data. Virtually all relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database. This model organizes data into one or more tables (or "relations") of columns and rows, with a unique key identifying each row. Relational databases can be used in PHANTOM to store runtime monitoring data in a relation based schema. The sample from runtime monitoring data would be stored with a specific relation to the platform, component and sensor or parameter that it derived from, simplifying the storage and access of runtime monitoring data.

SQLite [22] is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program. SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity. SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems.

Unlike client–server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. The library can also be called dynamically. The application program uses SQLite's functionality through simple function calls, which reduce latency in database access, since function calls within a single process are more efficient than inter-process communication. SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing. SQLite read operations can be multitasked, though writes can only be performed sequentially.

Due to the server-less design, SQLite applications require less configuration than client-server databases. SQLite is called zero-conf because it does not require service management (such as startup scripts) or access control based on GRANT and passwords. Access control is handled by means of File system permissions given to the database file itself. Databases in client-server systems use file system permissions which give access to the database files only to the daemon process.

Another implication of the serverless design is that several processes may need to be able to write to the database file. In server-based databases, several writers will all connect to the same daemon, which is able to handle its locks internally. SQLite on the other hand has to rely on file-system locks. It has less knowledge of the other processes that are accessing the database at the same time. This DBMS allows only one single write operating to take place at any given time, hence allowing a limited throughput.

SQLite can be used in PHANTOM to build the server to store data from the Runtime Monitor. Since the database will only be accessed by the PHANTOM platform, it can manage the concurrent write requests, overcoming the single write operation limitation of the SQLite. This solution can be ideal for small platforms and embedded systems, but may prove to be insufficient in the case of computer clusters. With the increase in size and complexity of the platform, there will be an increase in data that could outpace the limited throughput that SQLite has. Therefore, SQLite may not be the preferred choice for write-intensive deployments. However, for simple queries with little concurrency, SQLite performance profits from avoiding the overhead of passing its data to another process.

2.4.2 Versioning Control Systems

PHANTOM will have to manage many files with different purposes and origins, such as source files, binary files, etc. Versioning Control Systems are a good option managing the storage of such files, since they provide a proven storage mechanism, supporting a distributed access and edition mechanism. Moreover these systems keep record of version history of the files, providing an added value to PHANTOM platform.

Git [23] is a version control system that is used for software development and other version control tasks. As a distributed revision control system, it is aimed at speed, data integrity, and support for distributed, non-linear workflows. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers. Branches in Git are very lightweight: A branch in Git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.

Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.

The PHANTOM source code files and binaries can be stored in Git, over a heterogeneous platform with several remote nodes, since it allows having a copy of the entire development history locally and makes it easy to guarantee that every node has the latest source code and binaries.

Subversion [24] is a software versioning and revision control system. Subversion can be used to maintain current and historical versions of files such as source code, web pages,

and documentation. The system maintains versioning for directories, renames, and file metadata. Users can move and/or copy entire directory-trees very quickly, while retaining full revision history. Subversion has native support for binary files, with space-efficient binary-diff storage.

Each revision in a Subversion filesystem has its own root, which is used to access contents at that revision. Files are stored as links to the most recent change; thus a Subversion repository is quite compact. The system consumes storage space proportional to the number of changes made, not to the number of revisions.

Subversion uses an inter-file branching model to implement branches and tagging. A branch is a separate line of development. Tagging refers to labelling the repository at a certain point in time so that it can be easily found in the future. All the versions in each branch maintain the history of the file up to the point of the copy, plus any changes made since. One can "merge" changes back into the trunk or between branches.

Subversion does not store the modification times of files. As such, a file checked out of a Subversion repository will have the 'current' date (instead of the modification time in the repository), and a file checked into the repository will have the date of the check-in (instead of the modification time of the file being checked in). This might not always be what is wanted. To mitigate this, third-party tools exist that allow for preserving modification time and other filesystem metadata. However, giving checked out files a current date is important as well, since this is how tools like 'make' will take notice of a changed file for rebuilding it.

The key difference between Git and Subversion is that SVN is centralized and Git isn't. With Subversion, the repository is kept in a server, which means that without internet, developers cannot commit. If they want to make a copy of the code, they have to literally copy/paste it. With Git, this problem doesn't exist. Each local copy is a repository, and they can commit to it and get all benefits of source control, even while offline. When they regain connectivity to the main repository, they can commit against it. Git has the advantage that it's much better suited if some developers are not always connected to the master repository. Also, branching and merging support is a lot better, after all these are the core reasons it was written. And it's much faster than SVN. A benefit of Subversion is fine-grained permissions. SVN lets you limit access to directories and files by user. Git can do permissions, but it's not a native concept.

Both SVN and Git can be used in PHANTOM, for storing source code and binaries, despite the two having advantages and disadvantages, neither is better since they are quite different in their core structure. What can also be done are Git-SVN Bridges: The central repository is a Subversion repo, but developers locally work with Git and the bridge then pushes their changes to SVN. PHANTOM platform could use this method to exploit both SVN and Git features.

2.5 MULTI OBJECTIVE MAPPER

The Multi-Objective Mapper (MOM) is entrusted with the task of defining an optimal deployment plan for a PHANTOM application. This implies that MOM will need to take decisions on the placement of the parallelised components and on the shared data communications throughout the target architectures. Decisions will be based on the specification of the PHANTOM application, on the target hardware platform

specification, the Model Based Testing results and the results obtained by the Runtime Monitoring. Parameters whose values are not specified in the system specifications and models, will be explored in order to identify the configuration that provides better results. Decisions can be supported by data obtained from the results of the monitoring library and the application testing performed by the Model Based Testing. Moreover, MOM can also perform a dynamic optimisation of the component deployment based on the runtime information collected by the Runtime Monitoring module.

To be able to perform the mapping, MOM must be able to identify the target platform where the components achieve the best performance based on the application specification. This means that MOM must be able to have a certain level of understanding of the parallelised component code in order to be able to decide if it should run in a CPU, GPU or if it can achieve even better performance when implemented in a FPGA. MOM must decide which parallelisation techniques that will be used to in each case, identify the components that must share data and define the appropriate mechanisms for data sharing (will depend on the target platforms and on the communication that can be performed between these platforms).

JUNIPER [25] consists of a platform designed to be able to apply real-time technologies to Big Data processing. Juniper applications are distributed concurrent applications that can be implemented and deployed on a Juniper platform infrastructure in different ways, on different target platforms (clusters, networked nodes, FPGAs, etc.). JUNIPER defines a module to aid in the decision of how and where components from JUNIPER application should be deployed in order to optimise the performance and ensure that the application requirements are met. This module is called Scheduling Advisor and is divided in two components: the Heterogeneity Aware Scheduler and the Advisor Component. The Heterogeneity Aware Scheduler performs a benchmarking of each application component by executing it on each platform and collecting runtime parameters to assess the performance. Then, using the benchmarking results of each application component, the Heterogeneity Aware Scheduler compiles a plan that provides the optimal overall performance of the JUNIPER application. The Advisor component analyses the monitoring data corresponding to the execution of a JUNIPER application and attempts to identify potential issues related with the application performance and with the utilisation of resources. A detailed report is compiled for each issue found, being all issues grouped on a list and described in a XML file.

Therefore, JUNIPER provides a functional component able to take decisions regarding the selection of an optimal placement of JUNIPER application components on a heterogeneous infrastructure based on the execution of benchmarking evaluation of each component on the available target platforms. This evaluation is performed before the execution of the application, since the plan is defined offline and JUNIPER is unable to change the deployment plan during the execution of the application. PHANTOM will study JUNIPER approach, trying to reuse the tools and methodologies of the Scheduling Advisor and extending them to support online refinement of the deployment plan.

No tools or methodology were found to make decisions on the parallelisation and data sharing aspects of the component. Therefore, this area will be a main target of PHANTOM research. Multi-objective mapping optimisations are usually tackled by evolutionary and genetic algorithms associated with specific heuristics. Therefore, the

development of such custom heuristic tool will be a main target of PHANTOM research in order to address appropriately the PHANTOM workflow dynamics and the target use cases.

2.6 RUNTIME MONITORING

The performance of the monitoring system has two main aspects: The impact on system domain and the ability to monitor efficiently. Any element of the monitoring system that prevents the main system functionality from working is inappropriate. Ideally, the monitoring is a tiny fraction of each application's footprint, requiring simplicity. The monitoring function must be highly tuneable to allow for issues such as network performance, improvements to applications in the development life-cycle, appropriate levels of detail, etc. Impact on the system's primary function must be considered. Furthermore, monitoring must be efficient and capable of handling all monitoring goals in a timely manner, within the desired period. This is most related to scalability.

The system monitor may be writing data directly into a database, allowing other processes to access the database outside the context of the system monitor. This is dangerous however, as the table design for the database will dictate the potential for data-sharing. Ideally, the system monitor is a wrapper for whatever persistence mechanism is used, providing a consistent and 'safe' access interface for others to access the data.

Nagios [26] and Zabbix [27] are some of the main monitoring systems for networks, systems and applications that many corporative installations nowadays use. Zabbix is a full-blown development forked from Nagios, and its main characteristic is that it has a very aggregated view on monitoring. It covers performance, not only statuses, which is one of the most significant lacks in Nagios; apart from having a WEB management system that allows central management, without the troublesome configuration files, like the ones Nagios needs. Zabbix is a customizable open source tool that adapts well to a wide variety of applications, allowing centralized monitoring of a complete infrastructure. It's easy to build custom plugins for applications that may not be part of the existing community. It's a very visual tool and useful for trending and visualizing your data. It allows the creation of custom graphs, charts and alerts.

Nagios needs less resources than Zabbix, since it does not use a database to store information. The difficulty in management is not the Nagios instalation, but it is the need of an entire team dedicated to managing Nagios, meaning that the software on its own, without a team of people, cannot be exploited correctly.

Zabbix and Nagios both need installing a lot of plugins in order for them to be efficient and offer a series of complete features. Zabbix, on the other hand, does not have an “official” plugin library for the community, although it does have a list of OIDs for SNMP queries. Furthermore, it does not offer the possibility to work with Enterprise tools such as Oracle, Exchange, Active Directory, and others in the core. Nagios has a huge library, but it is low on maintenance since all the plugins are 100% open source and there is not a company to back it up or take care of them. Zabbix has a powerful template and trigger definition system based on regular expressions. It is quite powerful, yet at the same time complex in use: only meant for people who are capable of understanding regular expressions and in Nagios there is nothing of the sort.

Aside from the architecture, monitoring frameworks can be classified via the following nine key properties: scalability, non-intrusiveness, timeliness, granularity, extensibility, data storage, visualization, adaptability, and predictability. The requirements over these properties are not easy to be met by existing monitoring frameworks such as Nagios or Zabbix.

ATOM [28] introduces a new monitoring framework for both HPC and embedded systems to tackle the challenge of analysing the system's run-time context and overcoming the drawbacks of existing solutions. ATOM excels in being low-intrusive, flexible, and allowing to collect, visualize, and analyse relevant application and infrastructure data in near real-time.

We believe that ATOM delivers the features that are crucial for optimizing the energy-consumption and performance of applications at run-time, as well as improving energy-awareness in the PHANTOM platform, by monitoring applications at run-time, collecting large amounts of performance and energy-related data at high frequencies, and providing a user-friendly interface for data analysis.

2.7 SECURITY EXECUTION PLATFORM

Multiple Independent Levels of Security/Safety (MILS [29]) is a high-assurance security architecture based on the concepts of separation and controlled information flow, implemented by separation mechanisms that support both untrusted and trustworthy components, ensuring that the total security solution is Nonbypassable, Evaluatable, Always invoked and Tamperproof (NEAT). MILS architecture separates security mechanisms into manageable components. Processes are isolated into partitions that comprise a collection of data objects, code and system resources. These individual partitions can be evaluated separately. This approach substantially reduces the proof effort for secure systems. To support these partitions, the MILS architecture is divided into three layers: separation kernel; middleware, including the Partitioning Communications System (PCS); and applications.

The MILS separation kernel divides the computer into separate address spaces and scheduling intervals, guarantees isolation of the partitions and supports carefully controlled communications among them. The Separation Kernel Protection Profile (SKPP [30]) defines separation kernel as hardware and/or firmware and/or software mechanisms whose primary function is to establish, isolate and separate multiple partitions and control information flow between the subjects and exported resources allocated to those partitions, guarantying the protection of all resources (including CPU, memory and devices) from unauthorized access. The separation kernel requires the highest level of authentication, and is the only piece of software that runs in privileged mode. Therefore, no other code, not even device drivers, has the ability to affect the processor's protection mechanisms. Everything else, including all middleware, runs in user mode.

In the MILS architecture, middleware has a broader meaning than just traditional middleware. Most of the traditional operating system functions have been moved from the operating system to "middleware services," e.g., file systems, device drivers, trusted path, etc. Middleware services include a Partitioning Communications System (PCS) to extend the scope of the separation kernel to intersystem communication. Middleware

resides in the same kind of partition as the application that it supports, either co-resident with the application or in a partition by itself. Middleware runs in unprivileged (user) mode, making these services subject to separation kernel policy enforcement. The services that previously ran in privileged mode as part of the operating system, such as memory allocation, device drivers, I/O primitives, file systems and network stacks, now run in user mode in the MILS middleware layer.

Finally, the application level entities manage, control and enforce their own application-level security policies, such as firewalls, crypto services and guards. Instead of the "fail first, patch later" approach, trusted components are mathematically verified so that they are: Nonbypassable, Evaluatable, Always invoked and Tamperproof. In order to be effective, all system protection must be NEAT.

The separation kernel is microprocessor-centric. On this microprocessor, one can build a firewall that separates applications—top-secret from not, safety-critical from not—and guarantee that those applications won't talk to each other without an application-centric firewall. The separation kernel makes decisions about what goes on at the microprocessor level, but it knows nothing of the network. It just secures this one node. When a distributed system configuration is created, we would like it to be as safe or secure as if it were just a single processor. That's accomplished by implementing end-to-end enforcement of the basic MILS separation kernel policies. The Partitioning Communications System (PCS) is the enforcement mechanism. The collection of MILS nodes in a distributed system is called an enclave, and the PCS is present in each node in the enclave. The PCS takes this secure environment in the separation kernel and extends it to an enclave of computers—2, 100 or 10,000 computers. There will still be an application-centric firewall that separates applications, but it must be NEAT. Partitions are no longer restricted to being on the same processor. There could be hundreds of microprocessors, but one can still guarantee the firewall is tamperproof and nonbypassable. The MILS architecture makes it possible to secure tens of thousands of computers in a global information grid.

The MILS architecture is being applied today and will continue to be important in the most demanding applications where failure is unthinkable. Because it is both secure and affordable, it will be practical to use this architecture in PHANTOM to ensure that security is met throughout the entire PHANTOM platform, from the small embedded systems to the large and complex HPC platforms.

2.8 MODEL-BASED TESTING

The objective of Model-based Testing (MBT) in PHANTOM is to carry out black-box testing for use case applications and components on the PHANTOM platform with a focus on functional and non-functional properties of distributed and parallel computing systems. As described in D1.2, MBT activities can be divided into four stages as: 1) Creation of MBT models - the MBT models are created from requirement specifications of the application; 2) Generation of test cases - test cases are automatically generated by applying test selection criteria; 3) Execution of test cases - the generated test cases are executed either manually or within an automated test execution environment; 4) Results, evaluation, and feedback - the evaluation ends with a decision indicating the possible defect points and providing the feedback to the correspondent stakeholder.

Correspondingly the tools supporting MBT activities can be categorized as:

1. Modelling tools to support the creation of MBT models
2. Test generation tools to automatically generate test cases
3. Execution tools to execute test cases and analyse testing results

In this section, we briefly introduce the representative tools for each category and the details of tools can be found in the Annex. In particular, some tools fall into one or more categories since their functions cover more than one aspect.

Modelling tools provide textual or graphic interface to support the creation of MBT models which are used later for test case generation. IBM RSAD [31] is a commercial tool for software design and modelling based on UML. It is built on the Eclipse software framework and is extensible with a variety of Eclipse plugins. Alternatively Papyrus [32] is an open-source model-based engineering tool for UML and SysML. In addition to modelling, Papyrus also supports the execution of behaviour models using an animation and simulation framework. SIRIUS [33] is a modelling workbench created by composing of a set of Eclipse which allows the users to create, edit and visualize models. The models supported by SIRIUS are not limited to UML but include a set of diagrams, tables and trees.

Test generation tools are the key to support MBT activities, as compared to traditional tests, and to that respect, MBT is able to automatically generates test cases from MBT models. SmartTesting CertifyIT [34] is a commercial tool to generate test cases from models defined by IBM RSAD. The current supported input formats include UML state machine and class diagram, while the test cases can be exported in a test environment such as IBM Rational Quality Manager/Functional Tester but also as HTML, XML, or any other applicable format (including Perl/Python Script, or Java classes for Junit). MISTA [35] is an open-source and independent tool to build MBT models based on state machine and functional net, and then correspondingly generate test cases. The output formats of test cases cover a number of common programming/markup languages, such as C++, C#, Java and HTML. Diversity [36] is an open-source tool developed and integrated in Eclipse. Originally it takes models defined in xLIA (eXecutable Language for Interaction & Assemblage) [36] as input and generates test cases in TTCN-3 [37]. The recent integration of DIVERSITY with Papyrus allows users to define MBT models by uses of UML/sysML diagrams and then directly generate TTCN-3 based test cases from UML/sysML diagrams. In PHANTOM, MBT uses TTCN-3 as its primary format of test case due to its test performance, multi-purpose support and global standards [37].

Execution tools support the execution of test cases in a manual or automatic environment, and provide functions of result reporting and analysing. In PHANTOM we mainly investigate the automated execution environments supporting TTCN-3 test case execution by platforms. TITAN [38] is a TTCN-3 compilation and execution environment with an Eclipse-based IDE supporting both functional and performance tests. Together with Titan, a tool-box containing support for numerous protocols and several adapters are also released to open source.

In addition, PragmaDev Studio [39] is a commercial tool with complete support of all MBT stages and provides functions for requirement specification, test modelling, test case generation and test execution. This tool takes models in SDL format as input, and the output format of test cases is TTCN-3.

In PHANTOM, MBT is introduced for three aspects:

1. Functional Testing: By comparing real output with expected output, when a list of inputs is predefined.

2. Non Functional Testing: MBT can automatically generate testing data in a reasonable range (requires specifications), and by using the generated data, it is able to run applications with different input and get performance information such as energy consumption under different conditions (this is an advantage of MBT).

3. Model validation: Before/After implementation of applications, MBT is able to do model validation to check if the intended implementations contain any deadlock or some functions that will never be used (this is another advantage of MBT but requiring specification of applications).

2.9 DEPLOYMENT MANAGER

The Deployment Manager (DM) is expected to execute the deployment plan defined by the Multi-Objective Mapper. This role implies that DM will need to implement a code generation and refactoring stage, including the communication and consistency code required by the target platform where each component will be deployed. When a component is expected to run on a CPU, DM most gets the version of the component meant to be parallelized on CPUs and compiles it for the target CPU architecture. If, however, MOM decides that this component should be implemented on a FPGA, then DM must get the corresponding version of the component from the Repository and synthesize it to the target FPGA.

The deployment will also require DM to adjust the data sharing mechanism between components depending on the deployment of components that need to communicate. For instance, if two components are deployed on the same machine then the data sharing functions may be implemented using calls from the operating system. If these components are deployed on different machines, and connected by a network, then the APIs must be implemented using a protocol that supports the data sharing over the network. This set may require the refactoring of the components' code to include data synchronization calls.

2.9.1 Platform Specific Code Generation

2.9.1.1 Compilers

The deployment manager needs to have access to a set of compilers and be able to generate code for both CPUs and GPUs. Here, GCC [40] and Clang [10] are the most used compilers for the supporting compilation to multiple CPU platforms and to some GPUs.

GCC is a compile produced by the GNU Project that support various programming languages, namely C and C++, being the official compiler of the GNU operating system and the standard compiler in Linux and BSD operating systems. GCC is currently supporting OpenMP 4.0 for C/C++ and a limited version of OpenACC 2.0a specification. When using the OpenMP, GCC also supports code offloading to Intel Many Integrated Core (MIC) targets, and allows the generation of NVIDIA PTX - the low-level parallel thread execution virtual machine and instruction set used by CUDA - when using the OpenACC flag.

Clang consists of the front-end compiler front used by the LLVM project, generating LLVM intermediate representation, used by the LLVM toolchain, based on high-level programming languages. The original Clang project supports C, C++ and Objective-C. Due to the fact that Clang is built with a library-based design, Clang is to be integrated into other applications, fomenting the development of other compilers based on Clang. Therefore, Clang is on the base of most of the toolset of parallel computing frameworks. This is the case for NVIDIA CUDA, which SDK uses a CUDA LLVM Compiler; and OpenCL as its toolchain uses a modified version of Clang to compile OpenCL code into SPIR-V. OpenMP is supported by default by the original Clang Project. NVIDIA also provides a toolset to develop CUDA applications using the OpenACC standard, however it is not clear if this toolset uses Clang compiler.

Both GCC and Clang are used for compilation of parallel computing applications. Both support OpenMP, but only a modified version Clang supports OpenCL. GCC is able to generate NVIDIA low-level instruction set, but the official toolset for NVIDIA CUDA is based on Clang. Since such important toolsets (OpenCL and CUDA) use Clang and its usage give assess to a huge amount of tools under the LLVM umbrella, it seems that Clang is the best choice for parallel code compiler in PHANTOM.

2.9.1.2 Synthesizers

The generation of code specific for hardware accelerators depends on internal architecture of FPGAs. The number and type of logical resources and their organisation inside the FPGA varies not only from manufacturer to manufacturer, but also between the product family of a manufacturer. Therefore, the tools used for the synthetization of low-level hardware specific code will be the tools provided by FPGA manufacturers.

2.9.2 Operating System

An operating system (OS) is a set of software that handles computer hardware. Basically it acts as an interface between user program and computer hardware. It provides various services like communications, error detection, program execution, I/O operations, user interface, file manipulation, accounting, protection, security, resource allocation etc. These services are common for all but there are certain applications with specific requirement like processing time. For such applications, special OS is designed known as Real Time Operating System (RTOS [41]). The motive behind RTOS development is to process data as it comes in without or minimum buffering delay. Generally all RTOSes use POSIX [42] as the standard for application programming interface (API) calls.

A RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more

frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency. A real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

FreeRTOS [43] is a free real-time operating system kernel. It is aimed at small embedded real-time systems. Since most of the code is written in the c programming language, it is highly portable and has been ported to many different platforms. Its strength is its small size, making it possible to run where most (or all) other real-time operating systems won't fit. A typical use of the FreeRTOS kernel is when one needs to have some kind of (time critical) control algorithm, while also providing user-control and sensor monitoring.

RTLinux [44] strives to integrate a non real-time OS with a real-time OS. Since it is hard to get a non real-time OS to work nicely together with real-time tasks, RTLinux instead separates the non-real-time system from the hardware and acts as a layer in between. This makes it possible for RTLinux to run real-time tasks with very low latency and very high predictability. Linux itself is run as the idle-process in RTLinux and is therefore only run when there is no real-time task that needs to be run. RTLinux being a bigger system than FreeRTOS would have a more common use where more CPU heavy processing is required. One can also see the good use of already available tools in Linux.

Real-Time Executive for Multiprocessor Systems (RTEMS [45]), formerly Real-Time Executive for Missile Systems, is a real-time operating system (RTOS) designed for real-time, embedded systems and to support various open API standards including POSIX and µITRON [46]. RTEMS includes a port of the FreeBSD [47] TCP/IP stack as well as support for various filesystems including NFS and the FAT filesystem. RTEMS does not provide any form of memory management or processes. In POSIX terminology, it implements a single process, multithreaded environment. This is reflected in the fact that RTEMS provides nearly all POSIX services other than those which are related to memory mapping, process forking, or shared memory. RTEMS is bigger than FreeRTOS but has more flexibility and offers better support, though it is still a lot smaller than RTLinux. RTEMS is very popular for space uses since it supports multiple microprocessors developed for use in space including SPARC, ERC32 and LEON, MIPS Mongoose-V, Coldfire, and PowerPC architectures, which are available in space hardened models.

These RTOSes can be used in PHANTOM as the target platform OS since they support a vast number of architectures raging from embedded devices, to FPGAs and nodes in HPC centres. The minimal size and latency of RTOSes lead to minimal overhead in the target platforms. Furthermore, multitasking RTOS-based applications are interrupt-driven, so it is possible to largely eliminate polling from the application, freeing up processor resources for useful work and enabling power-saving modes to be invoked during idle periods, leading to more efficient use of CPU resources. Additionally, RTOSes provide a standardized set of stack and driver APIs that abstracts the specifics of the underlying hardware, making applications much more portable.

3. CONCLUSIONS

Three modelling languages, namely SysML, UML MARTE and MADES, are identified as more appropriate for PHANTOM to support the modelling of several types of systems, such as real-time and embedded systems, supporting the specification, analysis, design verification and validation of such systems. Another alternative is the usage of the Architecture Analysis and Design Language (AADL), which is intended for modelling of embedded and real-time systems, allowing to describe both the software and hardware architecture of a system, supporting the description of a large range of features and components, and enabling the design, checking, analysis of systems models as well as code generation.

A number of Compilers/Synthesisers (Clang) and Code Generators/Checkers (ANTLR, CodeRush, AutoRefactor, COINS, CETUS) can be used for the first operation of the Parallelisation toolset, namely the code analysis dealing with structural representation, simplification and testing for data dependencies of the source code. Then, the second operation of technique selection decides on the best parallelisation API to be used for the parallelisation technique (e.g. OpenMP threads communication, MPI, etc.).

SMP involves by nature cost-effective ways to support parallel programming. OpenCL kernel matches to some extent the definition of PHANTOM components, while provides mechanisms that support both data flow and control flow parallelisation. On the other hand, StarPu can be exploited for PHANTOM components implementation due to the capability to support asynchronous tasks.

The use of GPUs can offer a real improvement of an application execution speed and higher memory access speed than CPUs for some specific application functions when designed appropriately. OpenCL supports not only cross-platform applications but also all of the major GPU vendors, while may be combined with tools like Vulkan enabling applications to have more predictable behaviours and better performance. CUDA supports general-purpose computation on GPUs, while offering a number of advantages compared to traditional approaches for GPUs, but it is limited to NVIDIA GPU cards.

PHANTOM can explore FPGAs advantages by creating a repository for specific algorithms that are commonly used and are known to have better performance running on FPGAs. Furthermore, HLS tools can be used to synthetize parts of the application code, that are identified by a set of metrics, to be parallelisable and run faster on FPGAs, so not only the precompiled functions but also other parts of the application code can be implemented for FPGAs and exploit application specific co-processor architectures.

Both SVN and Git can be used in PHANTOM, for storing source code and binaries, either separately or in union with Git-SVN Bridges. In this configuration the central repository is a Subversion repo, but developers locally work with Git. The bridge then pushes their changes to SVN.

SQLite can be used in PHANTOM to build the server to store data from the Runtime Monitor. This solution can be ideal for small platforms and embedded systems, but may prove to be insufficient in the case of HPC due to the single write operation limitation of the SQLite, which limits the database throughput.

The usage of XML databases to store and manage descriptions and specifications represented in XML files. Such databases have the potential to improve the access to information in these files due to the fact that they provide built-in query engines specific and optimised for XML files. The usage of these databases may be studied during the development of PHANTOM platform in order to determine if they introduce any practical and significant improvement on PHANTOM overall performance.

JUNIPER Scheduling Advisor can be used as a first basis for supporting the Multi-objective mapper to derive the optimized deployment plan; however, extensions are required for supporting online refinement of the deployment plan, as well as to make decisions on the parallelisation and data sharing aspects of the component. Therefore, this area will be a main target of PHANTOM research, more probably with the development of evolutionary algorithms associated with specific heuristics, being able to address appropriately the PHANTOM workflow dynamics and the target use cases.

We believe that ATOM delivers the features that are crucial for PHANTOM's runtime monitor, collecting large amounts of performance and energy-related data at high frequencies, monitoring applications at run-time, and providing a user-friendly interface for data analysis.

The MILS architecture is both secure and affordable, making it ideal to implement in PHANTOM to ensure that security is met throughout the entire PHANTOM platform.

A number of tools (IBM RSAD, Papyrus, SIRIUS) can be used as modelling tools to support the creation of MBT models. SmartTesting CertifyIT, MISTA, DIVERSITY are candidates for test generation tools; however, DIVERSITY has a clear advantage, since it is an open-source tool developed and integrated in Eclipse that generates test cases in TTCN-3, which is the primary format of test case considered in PHANTOM due to its test performance, multi-purpose support and global standards. Automated execution environments supporting TTCN-3 test case execution by platforms will be considered, such as TITAN.

RTOSes can be used in PHANTOM as the target platform OS due to their support over a vast number of architectures. The minimal size and latency of RTOSes, and the interrupt-driven architecture, lead to minimal overhead in the target platforms and more efficient use of CPU resources. Furthermore, the standardized set of stack and driver APIs that abstracts the underlying hardware, greatly simplifies the process of making applications portable.

4. BIBLIOGRAPHY

[1] T. Weilkiens, Systems Engineering with SysML/UML: Modeling, Analysis, Design, San Francisco, CA: Morgan Kaufmann Publishers Inc, 2008.

[2] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," [Online]. Available: http://www.omg.org/spec/MARTE/. [Accessed 22 August 2016].

[3] I. Gray, N. Matragkas, N. C. Audsley, L. S. Indrusiak, D. Kolovos and R. Paige, Model-based hardware generation and programming-the MADES approach. In Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 14th IEEE International Symposium, 2011, pp. 88-96.

[4] Apache Software Foundation, "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/. [Accessed 16 August 2016].

[5] Apache Software Foundation, "Apache Spark - Lightning-Fast Cluster Computing," [Online]. Available: http://spark.apache.org/. [Accessed 9 August 2016].

[6] Apache Software Foundation, "Apache Storm - Distributed and fault-tolerant realtime computation," [Online]. Available: http://storm.apache.org/. [Accessed 9 August 2016].

[7] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell and J. Planas, OmpSs: a proposal for programming heterogeneous multi-core architectures., Parallel Processing Letters, 2011, pp. 173-193.

[8] OpenMP, "The OpenMP API Specification for Parallel Programming," [Online]. Available: http://openmp.org/wp/. [Accessed 3 August 2016].

[9] T. Parr, The definitive ANTLR 4 reference, 2013.

[10] ""clang" a C language family frontend for LLVM," [Online]. Available: http://clang.llvm.org. [Accessed 09 08 2016].

[11] Developer Express Inc, "CodeRush for Visual Studio," [Online]. Available: https://www.devexpress.com/products/coderush/. [Accessed 10 8 2016].

[12] J.-N. Rouvignac, "The AutoRefactor project," [Online]. Available: http://autorefactor.org/. [Accessed 8 8 2016].

[13] Association of COINS Compiler Infrastructure, "COINS Compiler Infrastructure," [Online]. Available: http://coinscompiler. osdn.jp/international/. [Accessed 9 8 2016].

[14] C. Dave, H. Bae, S.-J. Min, S. Lee, R. Eigenmann and S. Midkiff, "Cetus: A source-to-source compiler infrastructure for multicores," *Computer,* vol. 42, no. 12, pp. 36-42, 2009.

[15] "OpenACC Home," [Online]. Available: http://www.openacc.org. [Accessed 11 08 2016].

[16] The Khronos Group, Inc, "The open standard for parallel programming of heterogeneous systems," [Online]. Available: https://www.khronos.org/opencl/. [Accessed 3 August 2016].

[17] The Khronos Group, Inc, "C++ Single-source Heterogeneous Programming for OpenCL," [Online]. Available: https://www.khronos.org/sycl. [Accessed 9 August 2016].

[18] The Khronos Group, Inc, "The first open standard intermediate language for parallel compute and graphics," [Online]. Available: https://www.khronos.org/spir/. [Accessed 5 August 2016].

[19] The Khronos Group, Inc, "New generation graphics and compute API that provides high-efficiency, cross-platform access to modern GPUs," [Online]. Available: https://www.khronos.org/vulkan/. [Accessed 11 August 2016].

[20] Nvidia, "Parallel computing platform and programming model," [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed 11 August 2016].

[21] OpenSPL, [Online]. Available: http://www.openspl.org. [Accessed 29 August 2016].

[22] SQLite, "Self-contained, serverless, zero-configuration, transactional SQL database engine," [Online]. Available: https://www.sqlite.org/index.html. [Accessed 10 August 2016].

[23] Git, "Free and open source distributed version control system," [Online]. Available: https://git-scm.com/. [Accessed 17 August 2016].

[24] Apache Software Foundation, "Open source version control system," [Online]. Available: https://subversion.apache.org/. [Accessed 12 August 2016].

[25] JUNIPER, "Java Platform for high-performance and real-time large scale data," [Online]. Available: http://www.juniper-project.org/. [Accessed 9 August 2016].

[26] Nagios, "The industry standard in IT infrastructure monitoring," [Online]. Available: https://www.nagios.org/. [Accessed 2 August 2016].

[27] Zabbix LLC, “The Enterprise-class Monitoring Solution for Everyone,” [Online]. Available: http://www.zabbix.com/about.php. [Accessed 5 August 2016].

[28] ATOM, “Airport detection and Tracking of Dangerous Materials by Passive and Active Sensors Arrays,” [Online]. Available: http://atom-project.eu/. [Accessed 16 August 2016].

[29] Wikipedia contributors, “Multiple Independent Levels of Security,” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Multiple_Independent_Levels_of_Security&oldid=702651787. [Accessed 26 August 2016].

[30] Lynx software technologies, “Separation Kernel Protection Profile,” [Online]. Available: http://www.lynx.com/industry-solutions/industry-standards/skpp-separation-kernel-protection-profile/. [Accessed 3 August 2016].

[31] IBM, “Rational Software Architect Designer,” [Online]. Available: http://www-03.ibm.com/software/products/en/ratsadesigner. [Accessed 18 August 2016].

[32] The Eclipse Foundation, “Papyrus Modeling environment,” [Online]. Available: https://eclipse.org/papyrus/. [Accessed 8 August 2016].

[33] The Eclipse Foundation, “Sirius - The easiest way to get your own Modeling Tool,” [Online]. Available: https://eclipse.org/sirius/. [Accessed 4 August 2016].

[34] Smartesting Solution & Services, “CertifyIt,” [Online]. Available: http://www.smartesting.com/en/certifyit/. [Accessed 9 August 2016].

[35] MISTA, “Model-based Integration and System Test Automation,” [Online]. Available: http://cs.boisestate.edu/~dxu/research/MBT.html. [Accessed 3 August 2016].

[36] The Eclipse Foundation, “Eclipse Formal Modeling Project,” [Online]. Available: https://projects.eclipse.org/proposals/eclipse-formal-modeling-project. [Accessed 9 August 2016].

[37] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles and C. Willcock, An intrudoction to the Testing and Test Control Notation (TTCN-3), vol. 3, Computer Networks, 2003, pp. 375-403.

[38] The Eclipse Foundation, “Eclipse Titan,” [Online]. Available: https://projects.eclipse.org/projects/tools.titan. [Accessed 9 August 2016].

[39] PragmaDev, “Modeling and testing tools,” [Online]. Available: http://www.pragmadev.com. [Accessed 11 August 2016].

[40] Free Software Foundation, Inc., “GCC, the GNU Compiler Collection,” [Online]. Available: https://gcc.gnu.org. [Accessed 09 08 2016].

[41] Wikipedia contributors, “Real-time operating system,” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Real-time_operating_system&oldid=738540170. [Accessed 19 August 2016].

[42] IEEE Standards Association, “POSIX - Austin Joint Working Group,” [Online]. Available: http://standards.ieee.org/develop/wg/POSIX.html. [Accessed 19 August 2016].

[43] Real Time Engineers ltd, “FreeRTOS,” [Online]. Available: http://www.freertos.org/. [Accessed 18 August 2016].

[44] The linux fundation, “RTLinux,” [Online]. Available: https://wiki.linuxfoundation.org/realtime/start. [Accessed 10 August 2016].

[45] RTEMS Project, “RTEMS Real Time Operating System (RTOS),” [Online]. Available: https://www.rtems.org/. [Accessed 19 August 2016].

[46] ITRON Project, “ITRON Project Archive,” [Online]. Available: http://www.ertl.jp/ITRON/. [Accessed 2 August 2016].

[47] FreeBSD, “The FreeBSD Project,” [Online]. Available: https://www.freebsd.org/. [Accessed 1 August 2016].

[48] The Eclipse Foundation, “Eclipse Modeling Framework (EMF),” [Online]. Available: https://eclipse.org/modeling/emf/. [Accessed 09 08 2016].

Identified Tools

A.1. System Modelling

SysML

Description:

General-purpose modelling language designed for systems engineering applications, supporting the specification, analysis, design verification and validation of systems and systems of systems, assuming the role of the de facto standard for Model-Based Engineering. Extends UML standard, which tends to be software-centric, to easily model hardware, software, information, processes, personnel, and facilities.

Characteristics:

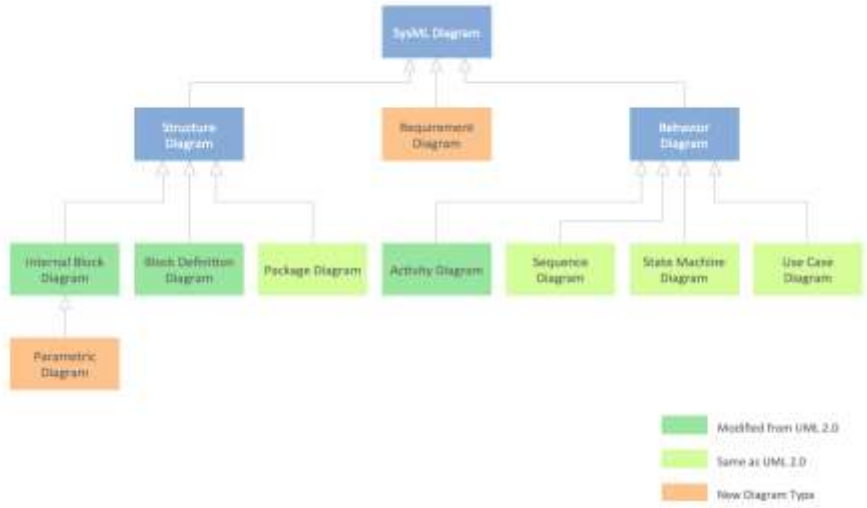
⇒ Requirements-driven design;

⇒ Based on OMG UML 2 standard, reusing UML concepts whenever possible, trying to make as few changes as possible to the standard, which allows an easy implementation by vendors of UML tools;

⇒ Extensions are made to the UML by using the profile mechanism of UML 2;

⇒ Allows the partitioning of the model elements into packages, forming logical groupings that minimise the circular dependences among them;

⇒ SysML supports the XMI interchange capability from UML, being also designed to be supported by the ISO 10303-233 – "Standard for the Exchange of Product model data" (STEP) in order to support interoperability between engineering tools.



UML MARTEs

Description:

MARTE (Modeling and Analysis of Real-Time and Embedded systems) is a UML profile for model-based development of real-time and embedded systems, providing support for specification, design, and verification/validation stages. MARTE intentions is not to define new techniques for analysing real-time and embedded systems, but to provide support to them.

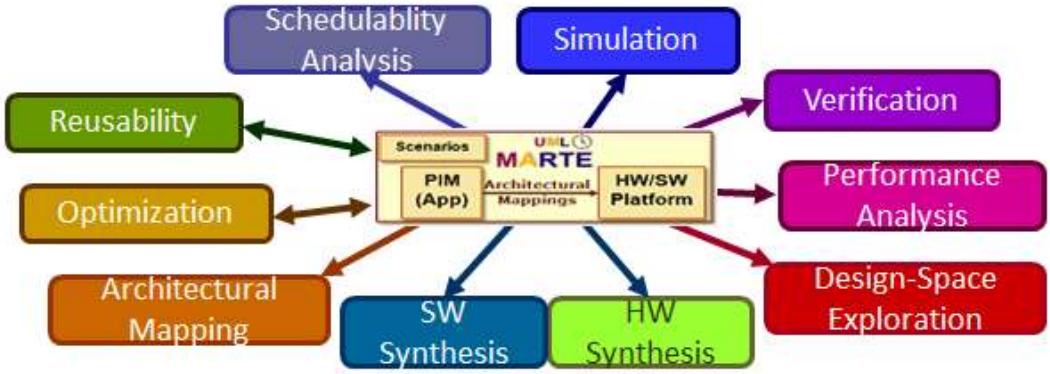
Characteristics:

⇒ Defines concepts for software and hardware platform modelling, supporting the description of resources and the definition of their properties.

⇒ Provides support the specification of semantically well-formed non-functional property modelling (e.g., throughputs, bandwidths, delays, memory usage), supported by a language to formulate algebraic and time expressions [9];

⇒ Defines concepts for allocation of applications on platforms and provides semantic support for real-time object paradigm to specific concurrency, communication and time-constraint aspects;

⇒ Provides support for quantitative analysis (e.g. scheduling, performance).



MADES

Description:

MADES aims to develop model-driven techniques to improve existing practices in development of real-time and embedded systems for avionics and surveillance embedded systems industries. For that, MADES defines a modelling language based on two existing UML profiles for embedded systems - SysML and MARTE – and provides tools and technologies to support the design, validation, simulation and automatic code generation.

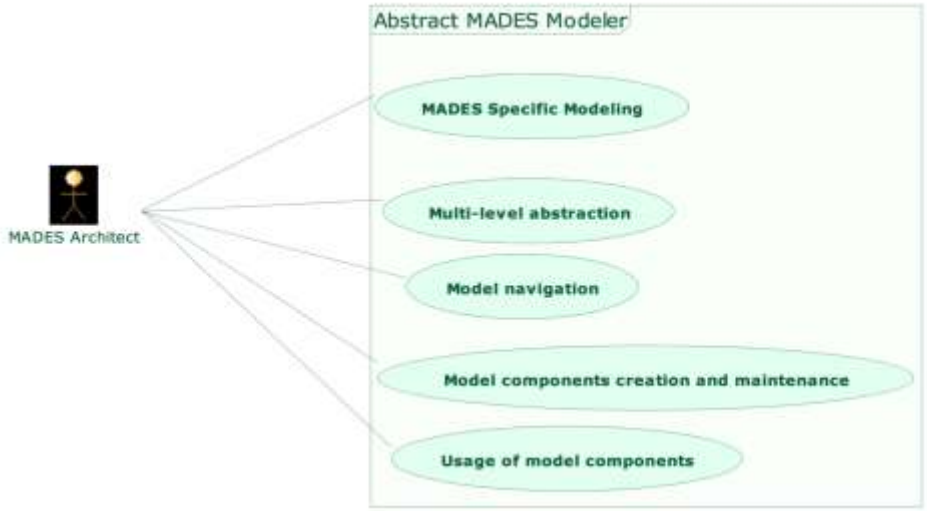
Characteristics:

⇒ Supports automate code generation to conventional programming languages, such as C, and to hardware description languages (e.g. VHDL);

⇒ Resorts to Compile-Time Virtualisation to reduce the impact of the diverse elements of modern hardware architectures and cope with their increasing complexity, allowing developers to write normal architecturally-neutral code with distributed operating system features such as threading, shared memory, cache coherency etc. and to automatically distribute that program over a complex target architecture;

⇒ MADES diagrams are able to model different viewpoints and different levels of abstraction, being able to describe both the structure and the behaviour of the system, as well as the time-constraints associated;

⇒ MADES toolset resorts to model transformation engines, being information required by these tools provided as model annotations or as user input.



AADL

Description:

AADL (Architecture Analysis and Design Language) consists in an architecture description language designed to model both software and hardware architectures of embedded and real-time systems. It is used to conduct analyses of critical real-time factors such as performance, dependability, security, and data integrity.

Characteristics:

⇒ Supports both a textual and graphical representation;

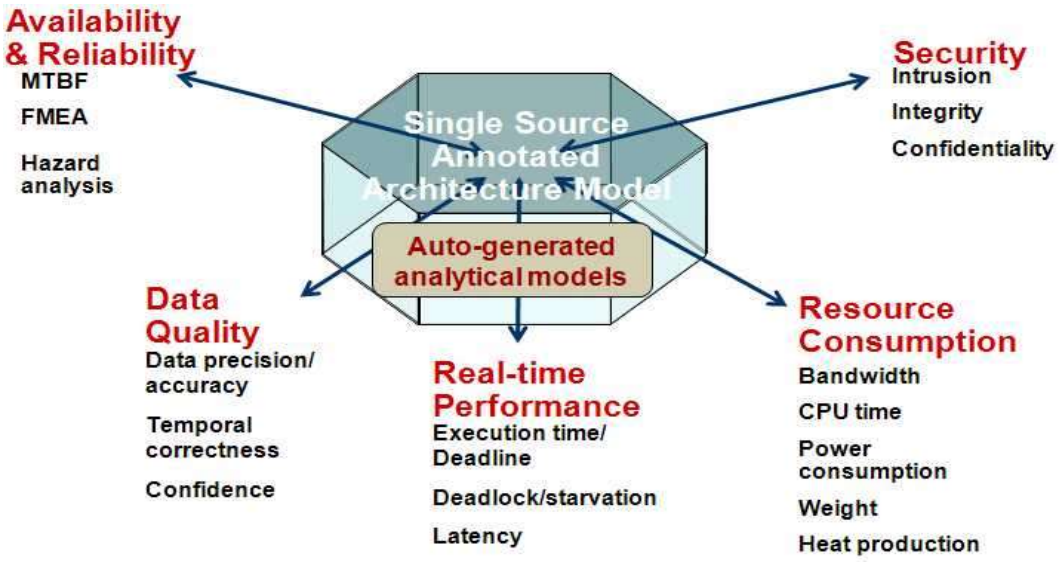
⇒ Provides modelling concepts to describe the runtime architecture of application systems in terms of concurrent tasks, their interactions, and their mapping onto an execution platform;

⇒ Supports an XML/XMI interchange format to support AADL model exchange between contractors and interoperability with commercial and in-house tools;

⇒ Provides analysis of system structure and runtime behaviour rather than functional behaviour;

⇒ Extendable by using user-defined properties and language annexes to the core language, enabling the addition of new components behaviour, definition of fault and propagation concerns, definition of modelling patterns, and description of specific data constraints;

⇒ Able to model embedded systems as component-based system architecture, component interactions, task execution and communication with precise timing semantics, execution platform and specify application binding, and operational modes and fault tolerant configurations.



A.2. Parallelisation tools

OpenCL

Description:

OpenCL consists in an open, royalty-free standard for cross-platform, parallel programming of diverse processors (CPUs, GPUs and FPGAs)[2].

Characteristics:

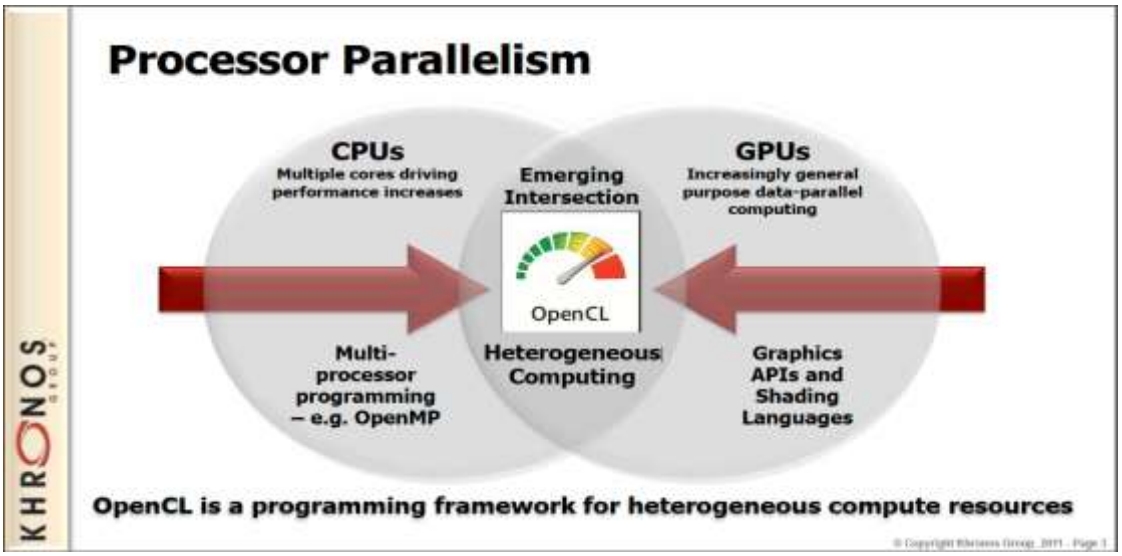
⇒ Programming Language supported: C and C++ (introduced in OpenCL 2.2);

⇒ Utilizes a subset of ISO C99 with extensions for parallelism;

⇒ Highly focused on application portability and execution in heterogeneous systems;

⇒ Supports Shared Virtual Memory (An address space exposed to both the host and the devices within a context);

⇒ OpenCL compilers are able to directly generate binary files (OpenCL C) or to translate code to an intermediate representation - Standard Portable Intermediate Representation (SPIR-V).



Vulkan

Description:

Vulkan is a new generation graphics and compute API that provides high-efficiency, cross-platform access to modern GPUs used in a wide variety of devices from PCs and consoles to mobile phones and embedded platforms.

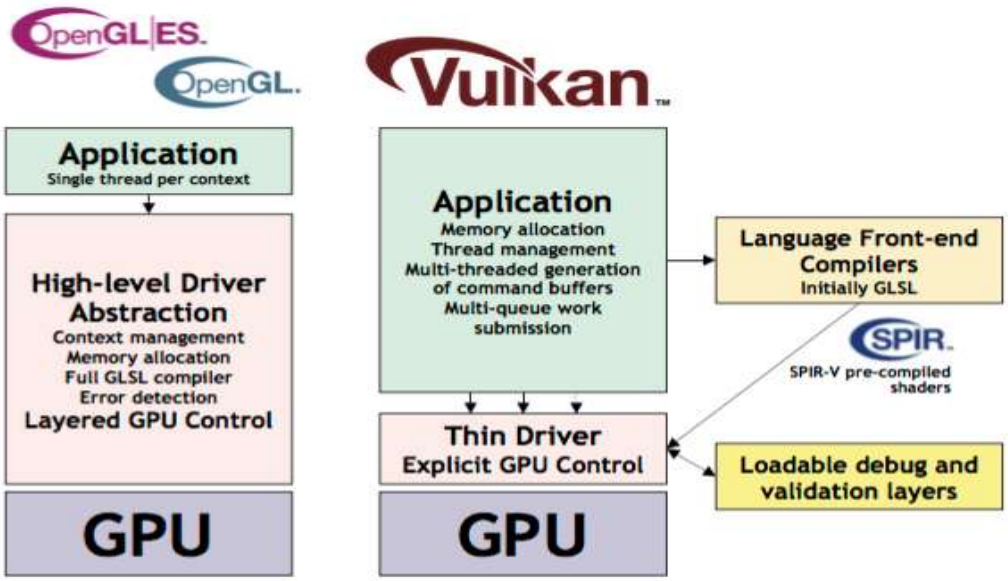
[2] A list with the vendors and devices that support OpenCL can be found here: https://www.khronos.org/conformance/adopters/conformant-products#opencl

Characteristics:

⇒ Provides a C-based graphics and compute API, which defines procedures and functions to specify shader programs, compute kernels, objects, and operations involved in producing high-quality graphical images, specifically colour images of three-dimensional objects;

⇒ Graphics and compute shaders for Vulkan are defined using an intermediate representation called SPIR-V;

⇒ Cross-vendor and Cross-Platform support: (Intel, NVidia, AMD, ARM)[3].



SYCL

Description:

SYCL is a cross-platform abstraction layer that builds on the underlying concepts, portability and efficiency of OpenCL that enables code for heterogeneous processors to be written in a "single-source" style using completely standard C++.

Characteristics:

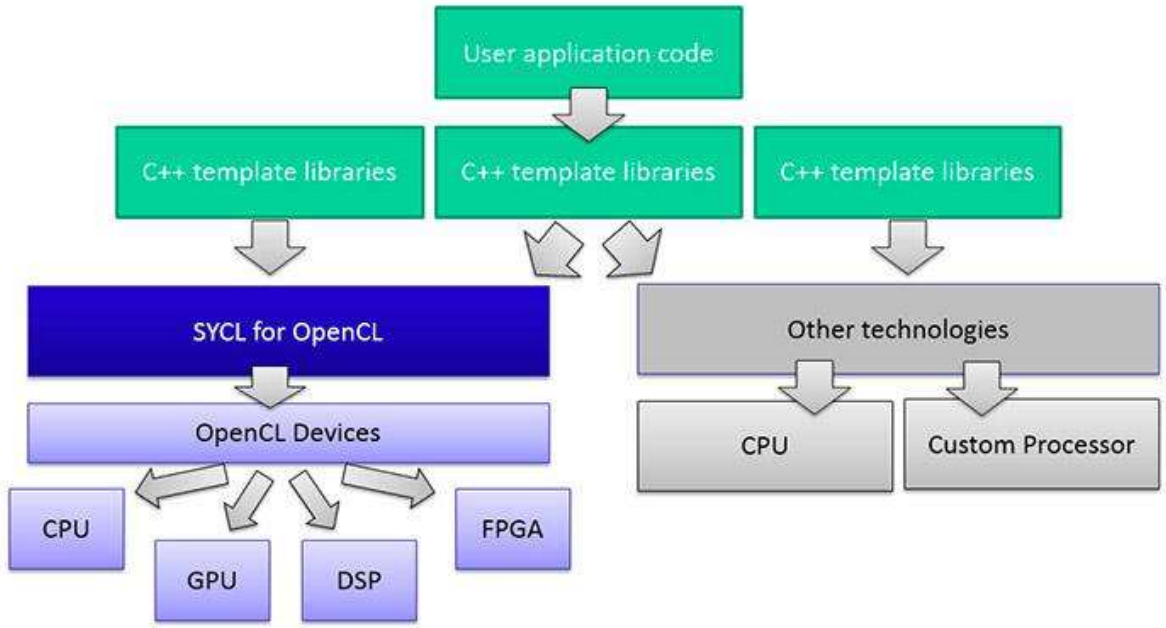
⇒ SYCL enables single source development where C++ template functions can contain both host and device code to construct complex algorithms that use OpenCL acceleration;

⇒ The open-source C++ 17 Parallel STL for SYCL, hosted by Khronos, enables the upcoming C++ standard to support OpenCL 2.2 features such as shared virtual memory, generic pointers and device-side enqueuer;

⇒ Device compilers enable SYCL running on OpenCL devices;

⇒ Developers have the ability to perform low level optimizations as needed.

[3] All venders and devices supporting Vulkan can be found here:
https://www.khronos.org/conformance/adopters/conformant-products#vulkan



SPIR-V

Description:

SPIR-V is the first open standard intermediate language which is intended to be used on parallel computing and graphic processing.

Characteristics

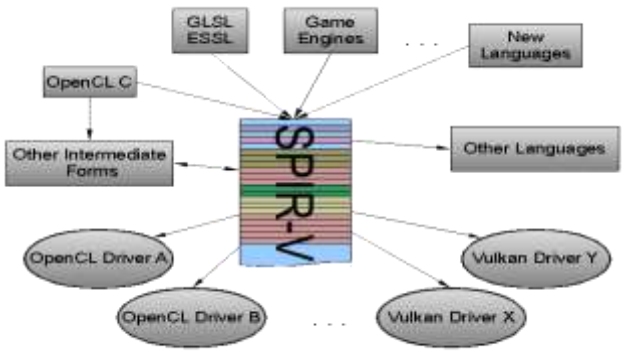
⇒ Part of OpenCL and Vulkan specification

⇒ Map easily to other intermediate languages

⇒ Be the form passed by an API into a driver to set shaders/kernels

⇒ Be low-level enough to require a reverse-engineering step to reconstruct source code.

⇒ Improve portability by enabling shared tools to generate or operate on it



OpenACC

Description

OpenACC allows the acceleration of applications by inserting simple compiler directives while maintaining a single source code for multiple CPUs and accelerators. The directives and programming model defined in the OpenACC API document allow programmers to create high-level host + accelerator programs without requiring the programmer to explicitly manage the accelerator (startup, configuration and shutdown), or manage data or program transfers between the host and accelerator.

Characteristics

⇒ Supports C and C++ where and parallelization directives are specified using pragmas, and supports Fortran where directives are expressed using a specially formatted comment statements;

⇒ Directives allow the programmer to override the compiler's mapping and data movement decisions, when necessary;

⇒ OpenACC compilers automatically map compute-intensive loops to parallel or vector execution units;

⇒ Can be used along with OpenMP – although both cannot be used on the same loop simultaneously;

⇒ OpenACC is interoperable with CUDA and accelerated libraries, and internode communication libraries like MPI and OpenSHMEM;

⇒ Portable a variety of hosts and accelerators /coprocessors.

OpenMP

Description

OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in Fortran and C/C++ programs. OpenMP was initially designed to make easier application of the parallelization paradigm on programs running in multi-CPU platforms, however it has been extended so that also DSP systems and accelerators could also be programmed.

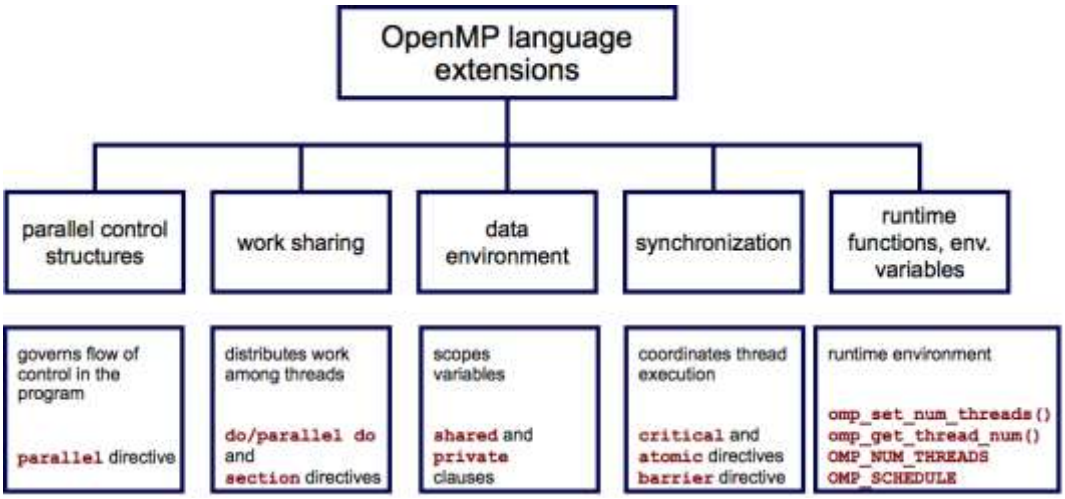
Characteristics

⇒ Is widely used for Symmetric Multiprocessing (SMP) systems, supporting 3 different programming languages (Fortran, C, C++);

⇒ Establish a simple and limited set of directives for programming shared memory machines;

⇒ OpenMP is mainly designed to multi-processor platforms, however some compilers also support GPUs and DSPs as target;

⇒ OpenMP 4.0 (currently at version 4.5) marks the beginning of the merge between OpenMP and OpenACC specifications;



POSIX Threads (Pthreads)

Description

Standardized C language threads programming interface that can be used to implement parallelism.

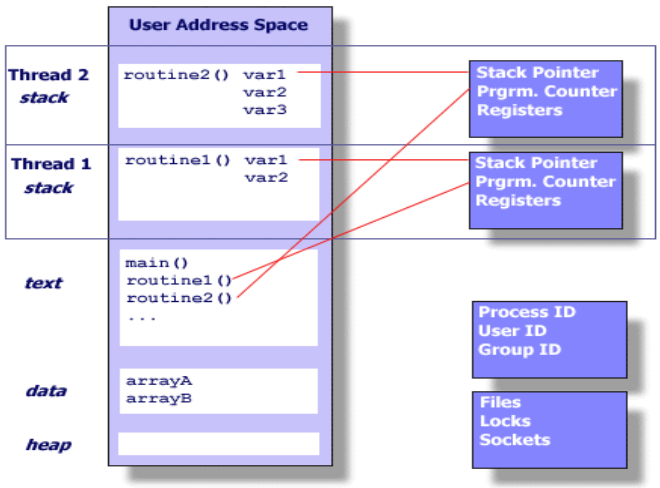
Characteristics

⇒ Uses a shared memory model:

⇒ All threads have access to the same global, shared memory

⇒ Threads also have their own private data

⇒ Programmers are responsible for synchronizing access (protecting) globally shared data.



CUDA

Description

Parallel computing platform and application programing interface model to use CUDA-enabled graphics processing unit (GPU) for general purpose processing.

Characteristics

⇒ Language: C, FORTRAN, DirectCompute, OpenAAC

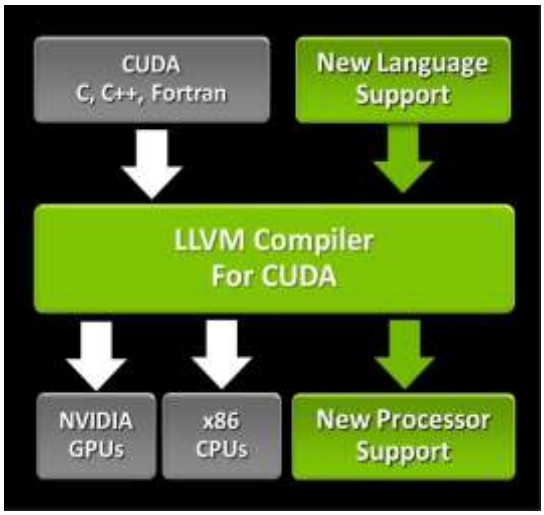
⇒ Requires CUDA-enabled GPUs (only NVidia GPUs)

⇒ Parallelism concepts: Hierarchy of thread groups, shared memories, and barrier synchronization -> exposed language extensions

⇒ The CUDA programming model also assumes that both the host and the device maintain their own separate memory spaces in DRAM.

⇒ Unified Memory Programming

⇒ Interoperability with OpenGL, Direct3D, SLI.



StarPU

Description

Is a library used for task programming that can be used on hybrid architectures (CPU & GPU)

Characteristics

⇒ CPU/GPU implementation of tasks

⇒ Portability: functions can be encapsulated in abstract tasks (codelets) where is possible to specify a set of functions for each architecture

⇒ Data transfer: provides a high-level data management library which enforces memory coherence over the machine, managing data allocation, fetching and transfer to achieve the most of the architecture (e.g. GPU-GPU direct transfers)

⇒ Dependencies between tasks can be explicitly and implicitly by the programmer

⇒ Heterogeneous scheduling is used to efficiently uses all computing resource available

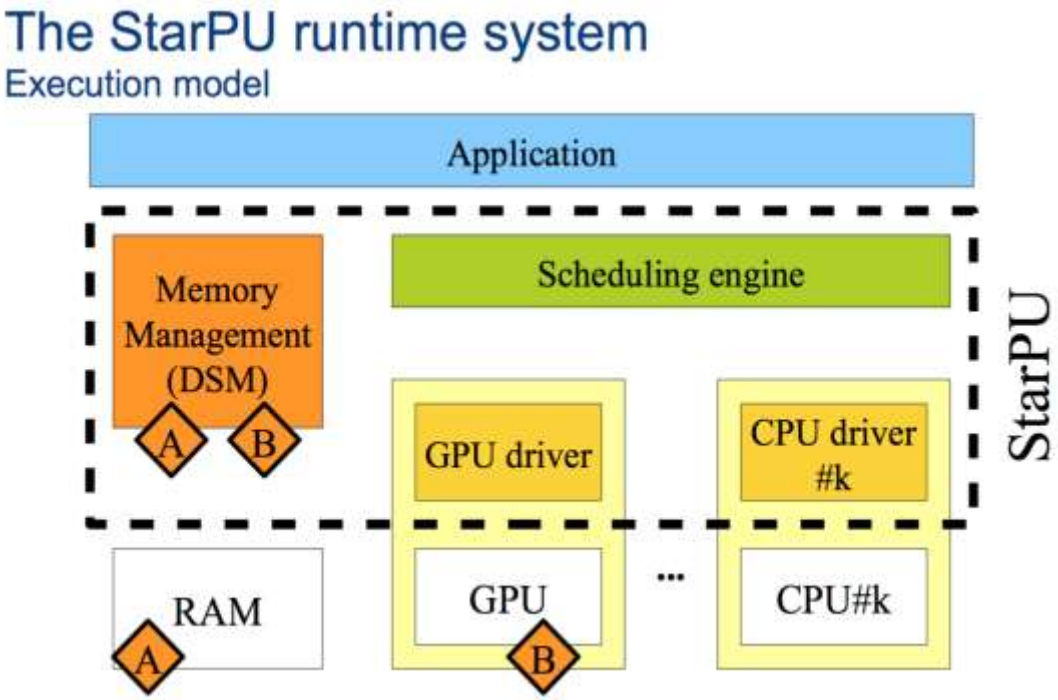
⇒ The several strategies are available to schedule tasks execution with the platforms that provides greater performance

⇒ Clusters can be supported by integrating StarPU with MPI, which will then be automatically combined and overlapped with the intra-node data transfers and computation. The application can also just provide the whole task graph, a data distribution over MPI nodes, and StarPU will automatically determine which MPI node should execute which task, and generate all required MPI communications accordingly.

⇒ Extensions to the C Language with pragmas and attributes that make it easy to annotate a sequential C program to turn it into a parallel StarPU program

⇒ OpenMP4 compatible allows to just rebuild OpenMP applications with K'Star source-to-source compiler, then build it with the usual compiler, and the result will use the StarPU runtime

⇒ OpenCL-compatible interfaces allow to simply run OpenCL applications on top of StarPU



Apache Spark

Description

Open source cluster computing framework providing an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

Characteristics

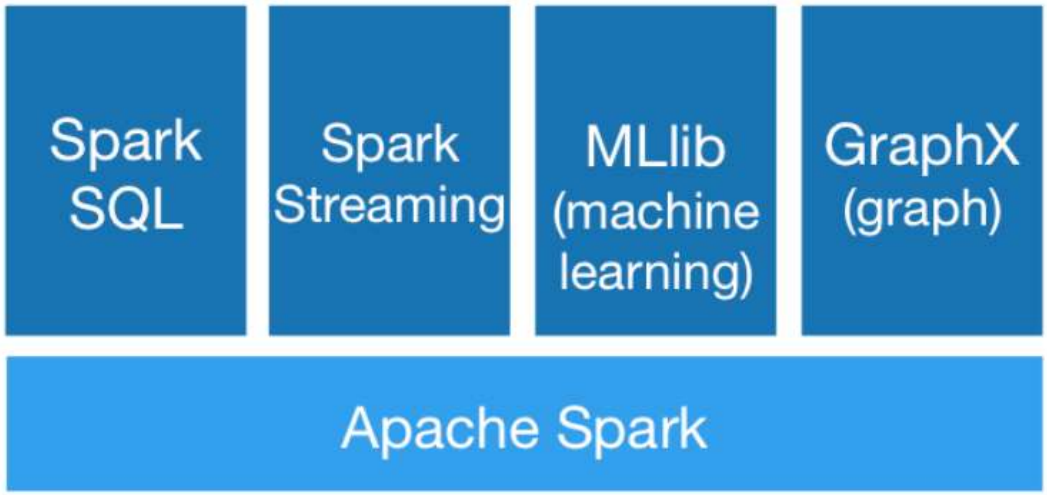
⇒ Addresses MapReduce limitation of providing a single, linear dataflow structure.

⇒ Supports Java, Scala, Python, SQL and R languages

⇒ Focused on the execution of machine learning, data analytic and data mining algorithms on big-data context.

⇒ Provides libraries for Spark SQL, Data Streaming analytics, Machine learning and graph computation

⇒ Apache Spark requires a cluster manager and a distributed storage system.



Apache Storm

Description

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime. Storm is simple, can be used with any programming language.

Characteristics

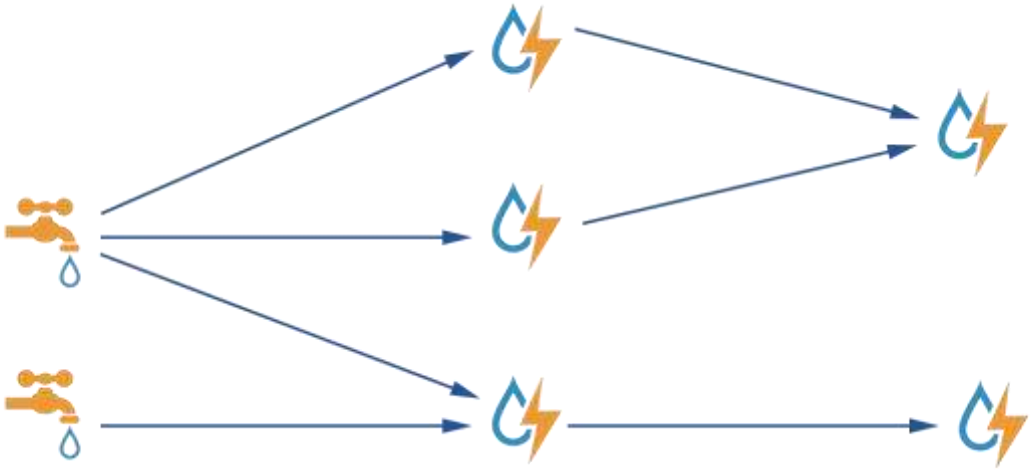
⇒ Has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL and more

⇒ Is fast: a benchmark clocked it at over a million tuples processed per second per node

⇒ It is scalable, fault-tolerant, guarantees your data will be processed and is easy to set up and operate.

⇒ Integrates with well-known queueing and database technologies

⇒ Consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.



Apache Hadoop

Description

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each one offering local computation and storage

Characteristics

- ⇒ Supports POSIX-style filesystem extended attributes
- ⇒ Clients can browser an fsimage via the WebHDFS API
- ⇒ Users can submit and kill applications through the YARN's REST API
- ⇒ Supports authentication through kerberos
- ⇒ Supports dynamic hierarchical user queues which are created dynamically at runtime under any specified parent-queue.

OpenMPI

Description

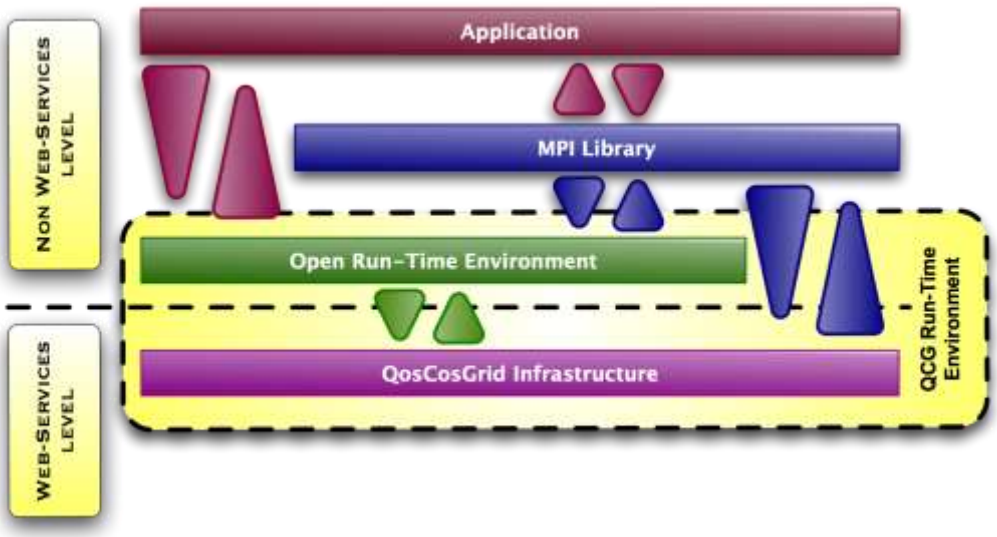
This library is an open source, freely available implementation of the MPI (Message Passing Interface) specifications.

Characteristics

- ⇒ Supports multithreading by enabling thread safety and concurrency
- ⇒ Supports dynamic process spawning
- ⇒ Fault tolerance available for network and process
- ⇒ Supports network heterogeneity
- ⇒ Supports run-time instrumentation

⇒ Supports several job schedulers

⇒ Support for GPUs (and CUDA)



MPICH

Description

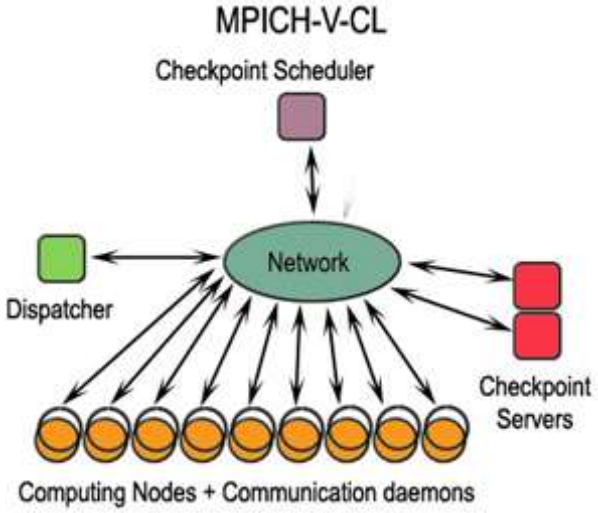
MPICH is a high-performance and widely portable implementation of the Message Passing Interface (MPI) standard, designed to implement all of MPI-1, MPI-2 and MPI-3 (including dynamic process management, one-sided operations, parallel I/O and other extensions).

Characteristics

⇒ MPICH provides a separation of process management and communication

⇒ The default runtime environment in MPICH is called Hydra

⇒ MPICH and its derivatives form the most widely used implementation of MPI in the world. They are used exclusively on nine of the top 10 supercomputers.



MVAPICH

Description

Is an open-source MPI software to exploit the novel features and mechanisms of high-performance networking technologies (InfiniBand, 10GigE/iWARP and 10/40GigE RDMA over Converged Enhanced Ethernet) and deliver best performance and scalability to MPI applications.

Characteristics

⇒ Supports several underlying transport interfaces (OFA-IB-CH3, OFA-IB-Nemesis, etc)

⇒ MPI-3 standard compliance

⇒ CH3-level design for scaling to multi-thousands cores with highest performance and reduced memory usage

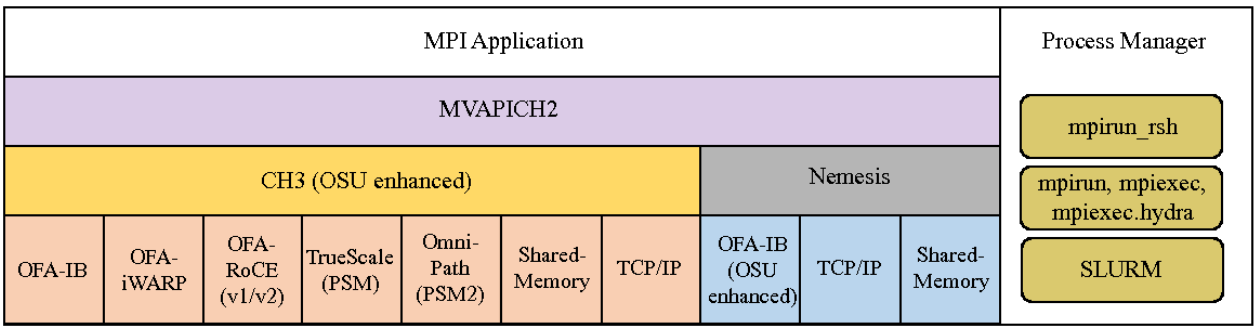
⇒ Support for MPI communication from NVIDIA GPU device memory

⇒ OFA-IB-Nemesis interface design

⇒ Flexible process manager support

⇒ Support for various job launchers and job schedulers

⇒ Configuration file support, provides a convenient method for handling all runtime variables through a configuration file.



MCAPI

Description

The purpose of MCAPI, which is a message-passing API, is to capture basic elements of communication and synchronization that are required for closely distributed (multiples cores on a chip and/or chips on a board) embedded systems.

Characteristics

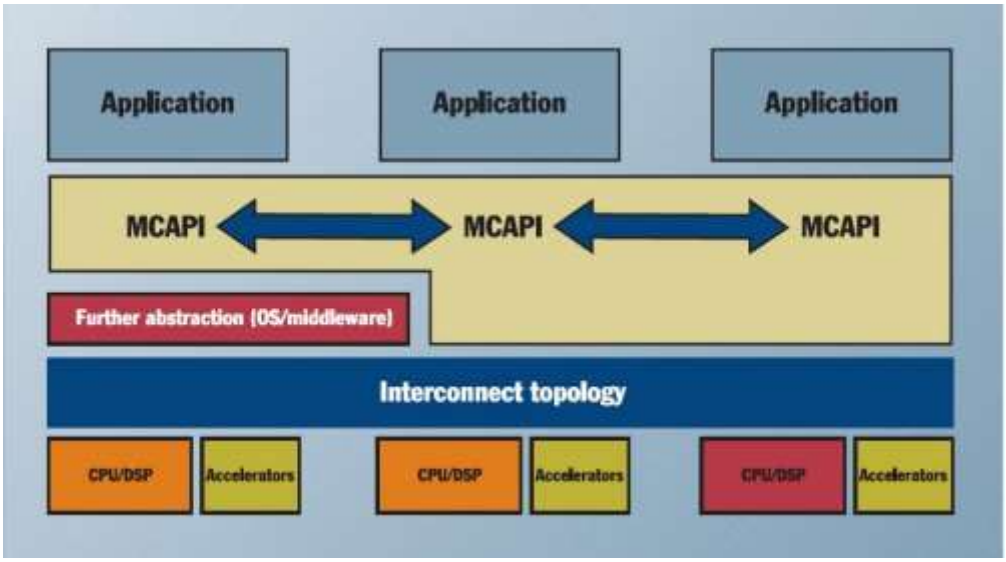
⇒ Message-passing API for inter-core communication, providing data sharing and synchronization between cores.

⇒ Applicable to multiple cores on a chip and multiple processors on a board

⇒ Focused in efficiency, small implementation

⇒ Supports the specification of different priorities between

⇒ Open source implementation: OpenMCAPI - For linux, android, Nucleus RTS, Drivers/porting to specific hardware may be needed



OpenSHMEM

Description

It consists in an effort to create an API specification that defines a standardized interface for writing parallel applications.

Characteristics

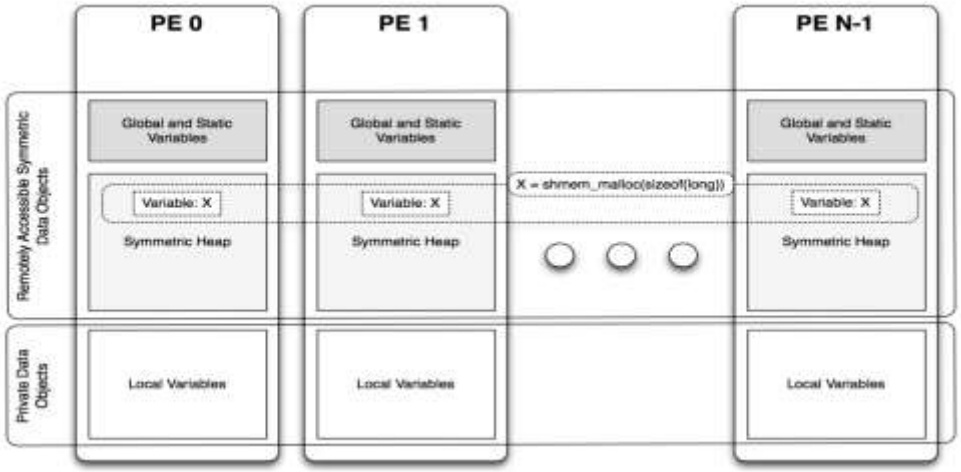
⇒ Consists on a Partitioned Global Address Space (PGAS) library interface specification

⇒ Library available both on C, C++ and Fortran

⇒ Provides asynchronous one-sided communication without software assistance

⇒ Supports several atomic operations (Swap, Increment, Add, etc.)

⇒ Supports collective operations which are used to handle resources available on the OpenSHMEM library



OmPSs

Description

Data-flow programming model to ease application porting to the heterogeneous architectures. It exploits task level parallelism and supports asynchronicity, heterogeneity and data movement.

Characteristics

⇒ In OmpSs the task construct allows the annotation of function declarations, when a function is annotated with a task construct each invocation of that function becomes a task creation point

⇒ In this library pragmas are available allowing to give indications on the data input/output for each task

⇒ User can specify one or multiple hardware devices where a given task should be executed and whether data needs to be copied from/to those devices

⇒ Various versions of the tasks can exist to target different architectures

⇒ Support Fortran, C and C++ languages

OpenSPL

Description

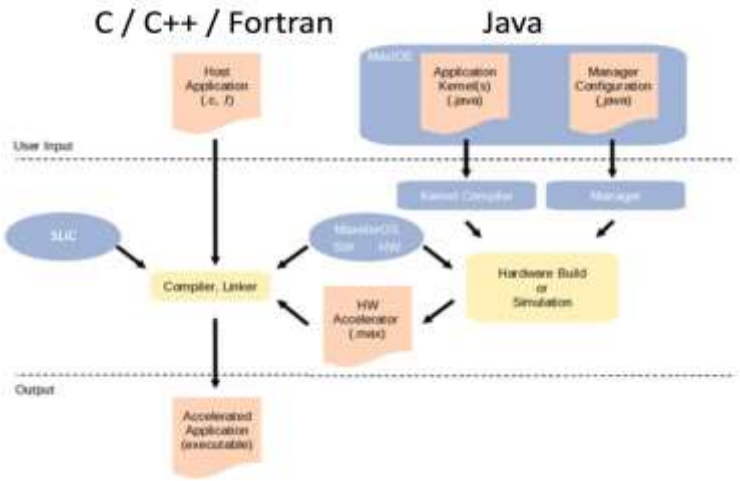
Programming model, based on Java language (MaxJ), that defines a Spatial Programming Language, which splits the control flow from the dataflow.

Characteristics

⇒ Code to be executed sequentially can be written in C language and will run on a CPU

⇒ Spatial code is written in variation of Java and will be executed on a FPGA using Data Flow Engine (DFE)

⇒ With spatial programing is possible to optimize a process, resulting in less power consumption and faster results



A.3. Management/Deployment/Computing Tools

Kubernetes

Description

Is an open-source system used for automating deployment, scaling and management of containerized applications.

Characteristics

⇒ It is a portable library supporting public, private, hybrid and multi-cloud deployment

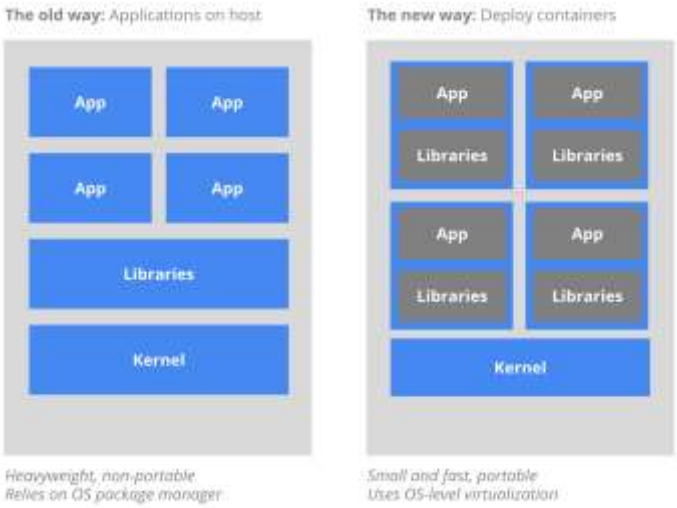
⇒ It is extensible in form of modular, pluggable, hookable and composable

⇒ Self-healing capabilities: auto-placement, auto-restart, auto-replication and auto-scaling

⇒ Applications are splitted into smaller, independent pieces that can be deployed and managed dynamically

⇒ Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.

⇒ A task before being executed must placed in a container



Docker Swarm

Description

Docker Swarm is a native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

Characteristics

⇒ Cluster management integrated with Docker Engine

⇒ Decentralized design

⇒ Declarative service model

⇒ Scaling

⇒ Desired state reconcilation

⇒ Multi-host networking

⇒ Service discovery

⇒ Load balancing

⇒ Secure by default

⇒ Rolling updates

Apache Mesos

Description

Apache Mesos abstracts CPU, memory, storage and other compute resources away from machines (physical or virtual) enabling fault-tolerant and elastic distributed systems to easily be built and run effectively

Characteristics

⇒ Linear scalability (tested up to 10 000 nodes)

⇒ Fault tolerant replicated master and agents using ZooKeeper Non-disruptive upgrades

⇒ Native support for launching containers with Docker and AppC images

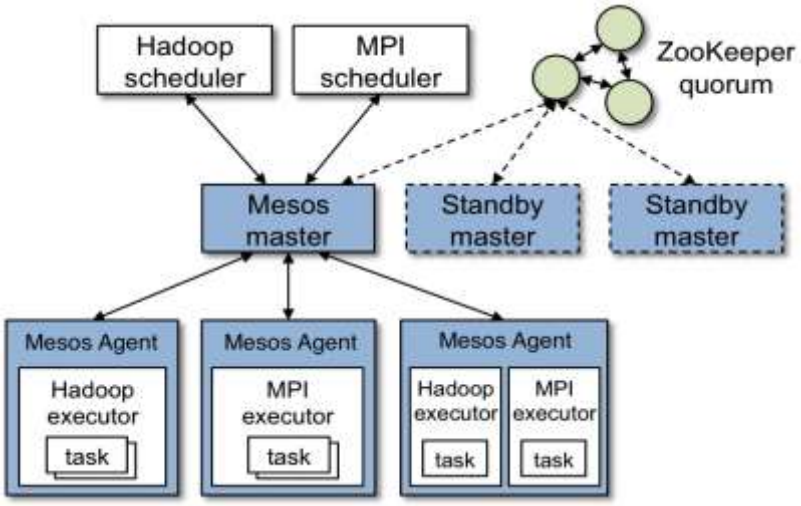
⇒ First class isolation support for CPU, memory, disk, ports, GP and modules for custom resource isolation

⇒ Supports for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

⇒ HTTP APIs for developing new distributed applications, for operating the cluster and for monitoring

⇒ Built-in Web UI for viewing cluster state and navigating container sandboxes

⇒ Runs on Linux, OSC and windows. Cloud provider agnostic



Fleet

Description

A low-level cluster management tool, designed as a foundation for higher order orchestration.

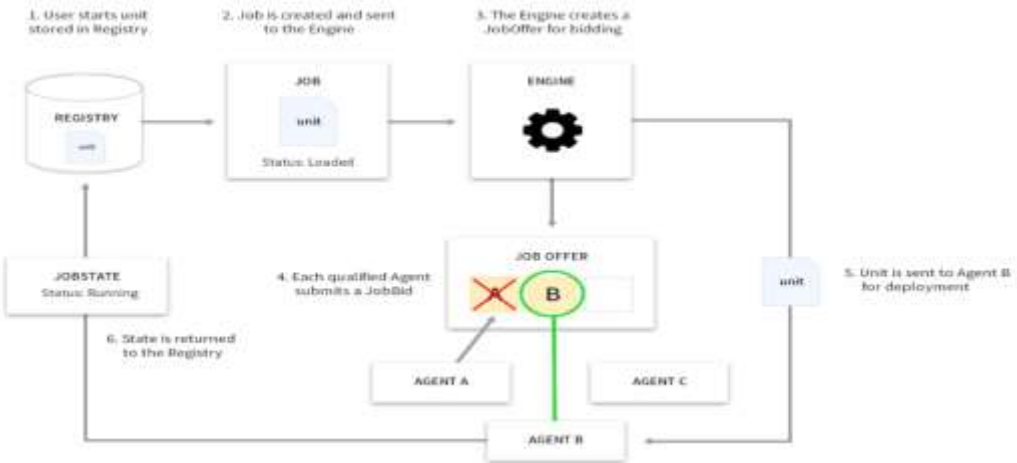
Characteristics

⇒ Supports various scheduling hints and constraints

⇒ It have some scalability limitations, is not recommended to run fleet clusters larger than 100 nodes or with more than 1000 services.

⇒ Supports different deployment patterns for an unit (deployment in a single/multiple location, automatic rescheduling, etc)

⇒ Intended to be build on a Linux machine



Ansible

Description

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration and many other IT needs.

Characteristics

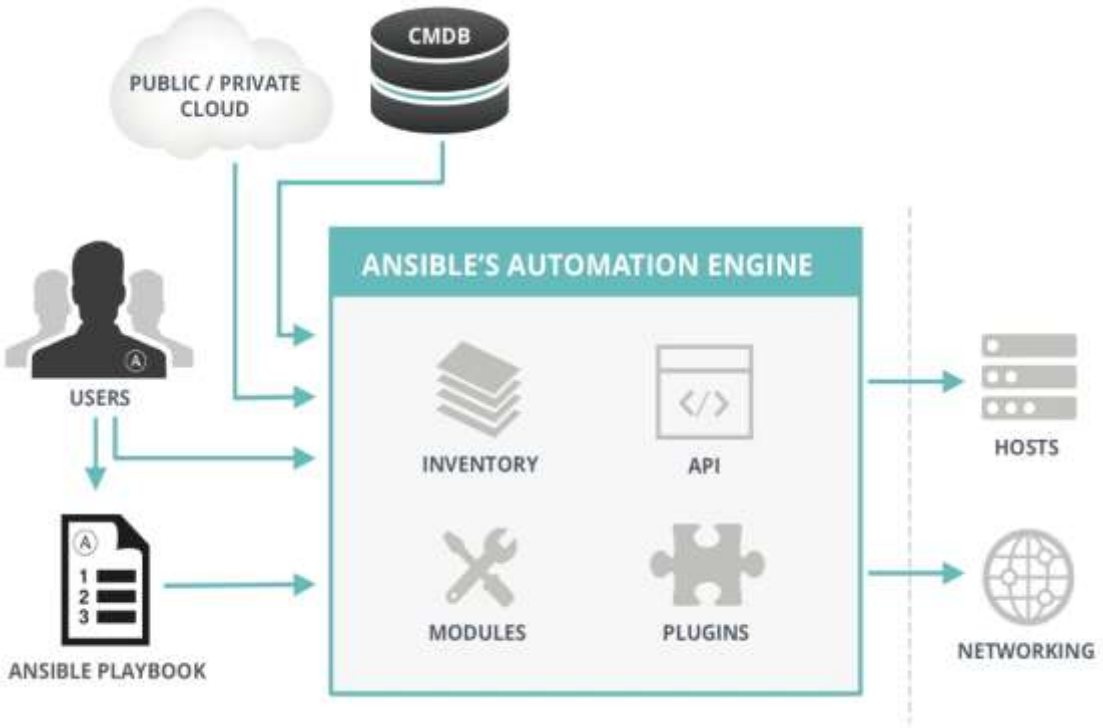
⇒ Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them.

⇒ Passwords are supported, but SSH keys with ssh-agent are one of the best ways to use Ansible, Kerberos is also supported

⇒ It is possible to manage the inventory by using a simple text file.

⇒ It is possible to use Ansible for ad hoc parallel task execution

⇒ Playbooks can finely orchestrate multiple slices of your infrastructure topology.



Apache Spark

Apache Spark can be both considered a parallelisation tool and Deployment/Computing tool, since it provides mechanisms to define parallel applications and to setup the computing infrastructure to implement applications. A Description of Apache Spark is provided in section A.2.

Apache Hadoop

Apache Hadoop can be both considered a parallelisation tool and Deployment/Computing tool, since it provides mechanisms to define parallel applications and to setup the computing infrastructure to implement applications. A Description of Apache Hadoop is provided in section A.2.

Apache Storm

Apache Storm can be both considered a parallelisation tool and Deployment/Computing tool, since it provides mechanisms to define parallel applications and to setup the computing infrastructure to implement applications. A Description of Apache Storm is provided in section A.2.

A.4. Infrastructures for heterogeneous platforms

JUNIPER

Description

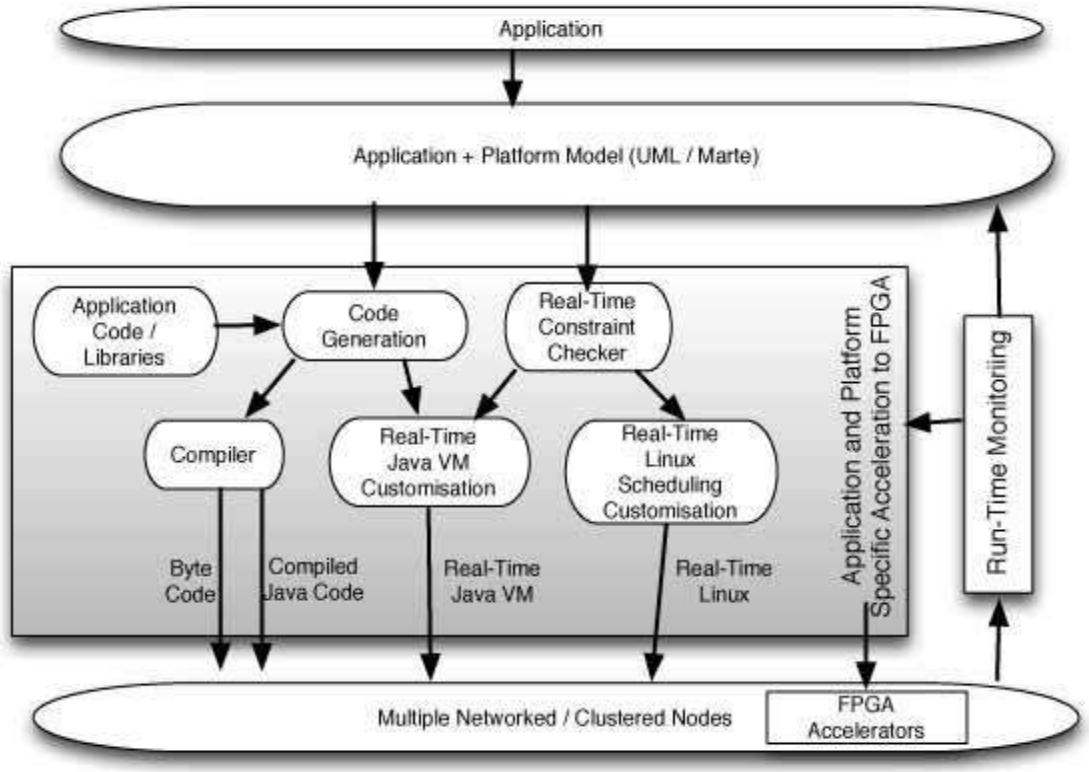
Platform from real-time technologies, using real-time principles, so that appropriate guarantees can then be given with respect to Big Data processing

Characteristics

⇒ This platform is based on a modular architecture, where the data is going to be handled across different stages.

⇒ The efficient and real-time exploitation of large streaming of data is based on several questions like performance, guarantees and scalability.

⇒ The project vision is to create a java platform that can support a range of high performance intelligent information management application domain.



OpenAMP

Description

Open source framework that allows operating systems to interact within a broad range of complex homogeneous and heterogeneous architectures and allows asymmetric multiprocessing applications to leverage parallelism offered by the multicore configuration.

Characteristics

⇒ Allows to configure, deploy, and manage multiple OS's across homogeneous and heterogeneous cores

⇒ Is compatible with Android OS

⇒ Allows the communication between different processors and OS's

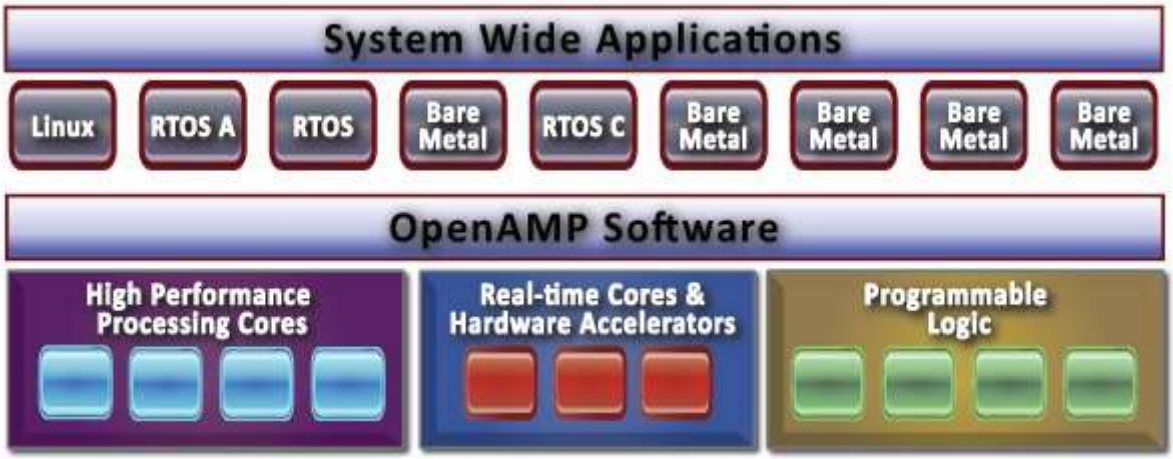
⇒ Shared memory protocol - Virtio/rpmsg

⇒ Lifecycle APIs - remoteproc

⇒ Proxy technologies emulate Linux processes

⇒ Compatibility with Multicore Communications API (MCAPI)

⇒ Pre-ported OS support by Express Logic, FreeRTOS, Mentor, Micrium, NXP, Xilinx



A.5. Compilers/Synthesisers

GCC

Description

The GNU Compiler includes front ends for C, C++, Objective-C, Fortran, Java, Ada and Go as well as library for these languages (libstdc++, libgcj, …). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

Characteristics

⇒ Compiler widely used on Linux/Unix systems.

Clang (LLVM)

Description

The goal of the Clang project is to create a new C, C++, Objective C and Objective C++ front-end for the LLVM compiler.

Characteristics

⇒ Fast compiles and low memory use

⇒ Expressive diagnostics

⇒ GCC Compatibility

⇒ Modular library based architecture

⇒ Supports diverse clients (refactoring, static analysis, code generation, etc)

⇒ Allow tight integration with IDE's.

⇒ Use LLVM 'BSD' License

⇒ A real-world, production quality compiler.

⇒ A simple and hackable code base.

⇒ A single unified parser for C, Objective C, C++ and Objective C++.

⇒ Conformance with C/C++/Objective C and their variants.

Xilinx Vivado

Description

It is a design suite offering a new approach for ultra-high productivity on designing, integrating and implementing systems using Xilinx FPGAs.

Characteristics

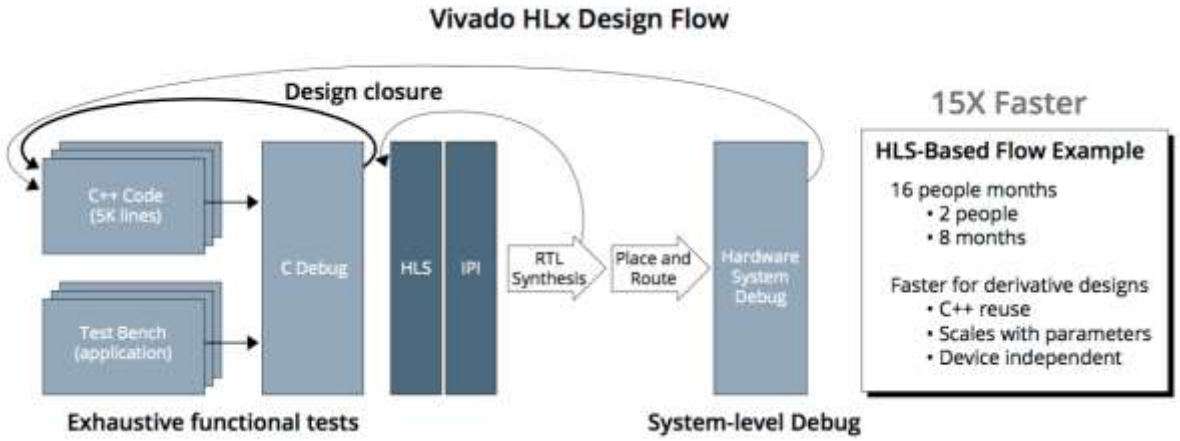
⇒ Software-defined IP generation

⇒ Block-based IP and Model-based DSP design integration

⇒ Integrated mixed language simulator

⇒ Integrated and standalone programming and debug environments

⇒ Accelerate verification by > 100X with C, C++ or systemC



A.6. Code Generator/Checker

Cetus

Description

Is a compiler infrastructure for the source-to-source transformation of software program, it currently supports ANSI C

Characteristics

⇒ It is written in JAVA programming language

⇒ Provides a driver that is used for automatic parallelization

⇒ Provides an API which automatically detect and inject OpenMP pragmas

⇒ It performs program and data dependence analysis

⇒ Used on source to source transformation of programs

COINS

Description

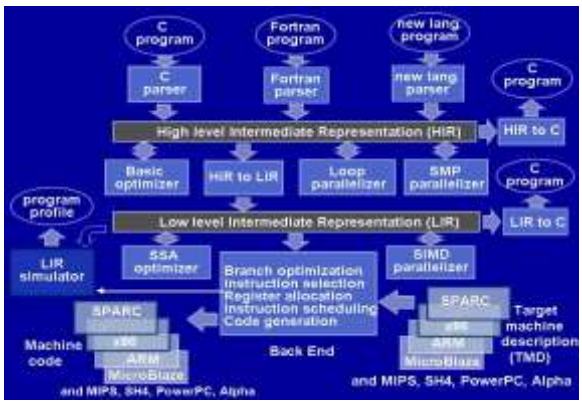
COINS compiler infrastructure provides modularized compiler components such as C front-end for Intel x86, Sparc, Arm, Mips, PowerPC, etc. So that it is easy to construct a new compiler by combining or modifying the components or adding new components.

Characteristics

⇒ It has been developed in java from scratch and is easily available for academic use.

⇒ Provides optimization to code (dead code elimination, common subexpression elimination within basic block, global variable temporalization within basic block, etc.)

⇒ It supports multiple source languages.



CodeRush

Description

Code refactor and debugger for Microsoft Visual Studio.

Characteristics

⇒ Visualize code and intuitively debug

⇒ Quick find symbols and files in your solution and easily navigate to code constructions related to the current context.

⇒ Simplify complex software systems.

AutoRefactor

Description

The AutoRefactor project delivers free software that automatically refactor code bases, the aim is to fix language/API usage in order to deliver smaller, more maintainable and more expressive code bases, the software will be an Eclipse plugin to automatically refactor Java code bases.

Characteristics

⇒ Provides easier maintenance.

⇒ Provides modernized code base.

⇒ Provides leaner and more compact code base.

⇒ Provides performance improvements.

DiscoPoP (LLVM)

Description

DiscoPoP is a tool that assists in identifying potential parallelism in sequential programs, it performs various types of static and dynamic analyses, and profiles the control and data dependences of the input program. The information from these analyses is used to identify sections of code called computational units (CUs), which follow a read-compute-write pattern and form the atoms of concurrent scheduling, DiscoPoP covers both loop and task parallelism. The dependences between CUs are used to detect different parallel patterns like pipeline. DoAll, etc. Experimental results show that reasonable speedups can be achieved by parallelizing sequential programs manually according to our findings.

Characteristics

⇒ Dependence profiling

⇒ Combined static and dynamic analysis to detect CUs: the analysis groups together pieces of code that perform a single task or computation

⇒ Control flow analysis

⇒ Analysis of loops for the presence or absence of the dependences to detect whether the loop can be parallelized

⇒ Identification of tasks, using CUs and related control and data dependence information, which can run in parallel.

⇒ Detection of different parallel patterns like pipeline, doAll, etc. Based on the dependence graphs of CUs.

ANTLR

Description

A powerful parser generator that can be used to read, process, execute or translate structured text or binary files, being widely used in academia and industry to build all sorts of language, tools and frameworks

Characteristics

⇒ Automatically constructs parse trees

⇒ Adaptative LL parsing algorithm - pushes all of the grammar analysis effort to runtime

⇒ The parser warms up like Java does with its JIT on-the-fly compiler, the code gets faster and faster the longer it runs

⇒ Version 4 supports code generation in: Java, C#, JavaScript. Python2 and Python3

⇒ Extensive grammar library: http://github.com/antlr/grammars-v4

A.7. Operating Systems

RTEMS

Description

The Real-Time Executive for Multiprocessor Systems (RTEMS) is an open source Real Time Operating System (RTOS) that supports open standard application programming interfaces (API) such as POSIX. It is used in space flight, medical, networking and many more embedded devices using processor architectures including ARM, PowerPC, Intel, Blackfin, MIPS, Microblaze and more.

Characteristics

⇒ Supports several CPU's architectures (arm, i386, powerPC, etc)

FreeRTOS

Description

FreeRTOS is the market leading real time operative system (RTOS) and the de-facto standard solution for microcontrollers and small microprocessors.

Characteristics

⇒ FreeRTOS never performs a non-deterministic operation, such as walking a linked list, from inside a critical section or interrupt

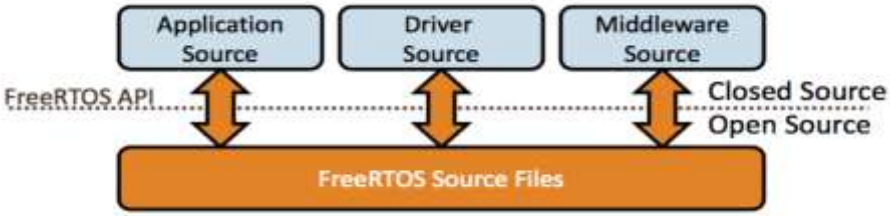
⇒ It implements an efficient software timer that does not use any CPU time unless a timer actually needs servicing.

⇒ List of blocked (pended) tasks do not require time consuming periodic servicing.

⇒ Direct to task notification allow fast task signaling, with practically no RAM overhead, and can be used in the majority of inter-task and interrupt to task signaling scenarios.

⇒ The FreeRTOS queue usage model manages to combine both simplicity with flexibility (in a tiny code size)

⇒ FreeRTOS queues are base primitive on top of which other communication and synchronization primitive are built. The code re-use obtained dramatically reduced overall code size, which in turn assists testing and helps ensure robustness.



Real-Time Linux

Description

RTLinux is a hard realtime RTOS microkernel that runs the entire Linux Operating System as a fully preemptive process. The hard real-time property makes it possible to control robots, data acquisition system, manufacturing plants and other time-sensitive instruments and machine from RTLinux applications

Characteristics

⇒ Contains a priority scheduler that supports both a "little POSIX" interface and the original V1 RTLinux API

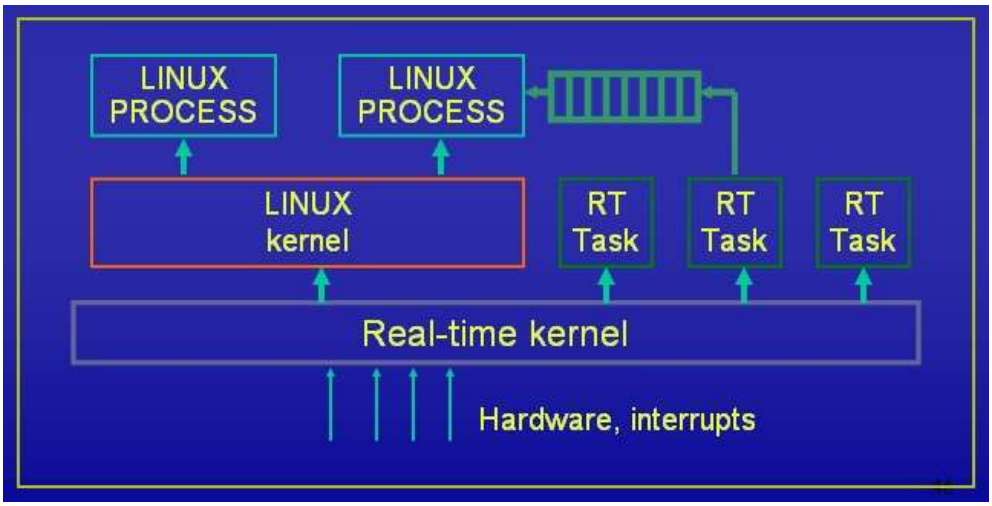
⇒ It controls the processor clock and exports an abstract interface for connection handlers to clocks

⇒ Supports POSIX style read/write/open interface to device drivers

⇒ Connects RT tasks and interrupt handlers to Linux processes through a device layer so that Linux processes can read/write to RT components.

⇒ A contributed package by Jerry Epplin which gives RT tasks blocking semaphores.

⇒ Provides shared memory between RT components and Linux processes.



A.8. Repositories

GIT

Description

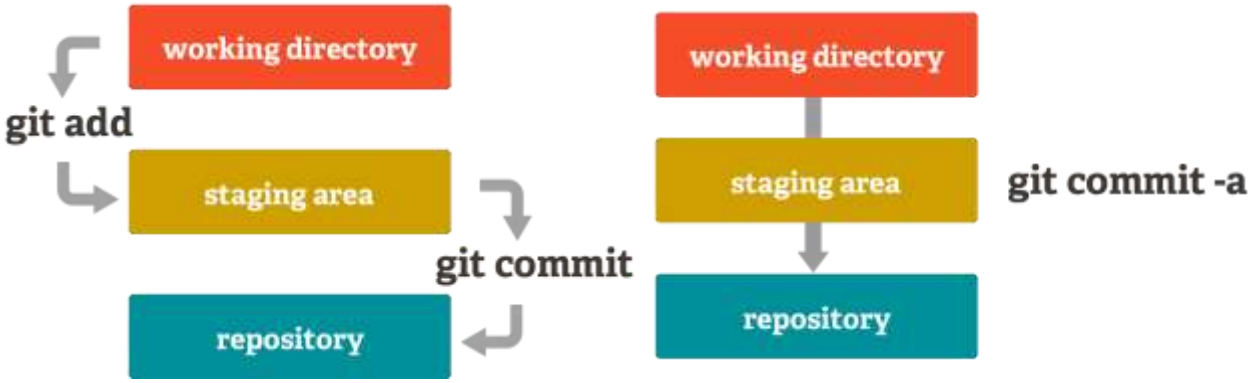
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

Characteristics

⇒ Supports branching and merging

⇒ It is small and fast

⇒ Free and open source



Subversion

Description

Subversion is an open source version control system. Founded in 2000 by CollabNet, Inc., the Subversion project and software have seen incredible success over the past decade. Subversion has enjoyed and continues to enjoy widespread adoption in both the open source arena and the corporate world

Characteristics

⇒ Directories, copying, deleting and renaming are versioned.

⇒ Branching and tagging are cheap (constant time) operations

⇒ Symbolic links can be versioned

⇒ Executable flag is preserved

⇒ It contains an Apache network server options, with WebDAV/DeltaV protocol

⇒ It contains a standalone server option.

⇒ Binary files are handled efficiently.

⇒ Costs are proportional to change size, not data size.

SQLlite

Description

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain.

Characteristics

⇒ Doesn't need to be "installed" before it is used, since SQLite uses no configuration files.

⇒ Stable Cross-Platform Database File.

⇒ SQL Statement compile into virtual machine code

⇒ SQL Language extensions

BaseX

Description

BaseX is a light-weight, high performance and scalable XML Database engine and XPath/XQuery 3.1 processor, which includes full support for the W3C update and full text extensions. An interactive and user-friendly GUI frontend gives you great insight into your XML documents.

Characteristics

⇒ High performance database storage with text, attributes, full-text and path indexes.

⇒ Efficient support of the W3C XPath/Xquery Recommendations, Full Text and Update extensions.

⇒ One of the highest available compliance rates for all supported specifications.

⇒ Client/Server architecture, supporting ACID safe transactions, user management and logging.

⇒ Highly interactive visualizations, supporting very large XML documents.

⇒ The only realtime XQuery editor available, with syntax highlighting and error feedback.

⇒ A wide range of interfaces: REST/RESTXQ, WebDAV, XQJ, XML:DB, clients in many different languages.

eXist

Description

A NoSQL document database application platform built entirely around XML technologies.

Characteristics

⇒ Schema-less database

⇒ eXistdb applications are packaged as single archive files that are installed directly in the database

⇒ eXistdb is fully based upon open standards

⇒ It uses a browser based IDE.

Berkley DB XML

Description

Oracle Berkeley DB XML is an open source, embeddable XML database with XQuery-based access to documents stored in container and indexed based on their content. Oracle Berkley DB XML is built on top of Oracle Berkley DB and inherits its rich features and attributes.

Characteristics

⇒ Build on top of Oracle Berkley DBm inheriting all its rich features, such as transactions and replication

⇒ Provides a document parser, XML indexer and XQuery engine on top of Oracle Berkeley DB to enable the fastest, most efficient retrieval of data.

⇒ Application perform database administration, eliminating the need for DBA and allowing continuous, unattended operation.

⇒ Lowers total cost of ownership with extreme performance to reduce hardware costs

A.9. Monitoring Systems

ATOM

Description

Profiles applications with high resolution, focuses on energy measurements, and supports a heterogeneous infrastructure.

Characteristics

⇒ Low-intrusive, highly scalable architecture based on open-source tools and libraries

⇒ Flexible, language-independent plug-in system in order to support collecting specific data including the energy consumption of embedded systems or connecting to external power measurement devices.

⇒ Light-weight and easy-to-grasp user library that allows code instrumentation in order to gather application specific data not accessible globally

⇒ Integration with the PBS resource manager to allow monitoring applications without any prior knowledge by end users or software developers

⇒ Interactive front-end allowing for near real-time exploration of collected data as well as exporting historical data for further analysis

Nagios

Description

Nagios is a cost effective, flexible open source tool that is one of the standard bearers for network monitoring in the open source community.

Characteristics

⇒ Incredible flexibility

⇒ Easy installation

⇒ Scalable

⇒ Easy to navigate web-interface

⇒ Plenty of plugins

⇒ Alert system with e-mail and/or SMS

⇒ Event handlers help with problem remediation

⇒ Capacity planning and trending

⇒ Reporting system

⇒ Web content monitoring

⇒ Nagios suports all main protocols (HTTP, FTP, SSH, POP3, SMTP, SNMP, MySQL, etc)

Zabbix

Description

Zabbix offers great performance for data gathering with the possibility to be scaled to very large environments (up to 100 000 devices)

Characteristics

⇒ It monitors several performance indicators like CPU, memory, network, disk space and processes

⇒ Native process available for Linux, UNIX and Windows platforms

⇒ Supports SNMP agents

⇒ Customizable: No limits on the scripting of programming language in use - have your own checks in shell, Perl, Python or anything else

⇒ Provides support for IPMI access which can be used to gather statistics from sensor, a send command to turn on or off devices.

⇒ Supports both agent and agent-less monitoring approaches.

⇒ Designed for small and large-scale environments

⇒ Supports IPv4 and IPv6

A.10. Model-Based Testing

Papyrus

Description

Papyrus is an industrial-grade open source model-based engineering tool. Papyrus has notably been used successfully in industrial projects and is the base platform for several industrial modelling tools.

Characteristics

⇒ Papyrus have a graphical editing tool for UML2 as defined by object management group (OMG).

⇒ Papyrus provides also a complete support to SysML in order to enable model-based system engineering.

⇒ Thanks to MoKa, Papyrus can execute models using rich and extensible animations and simulation framework.

⇒ All the modelling features of Papyrus are designed to be customized and to maximize reuse

SIRIUS

Description

Sirius is an eclipse project which allows you to easily create your own graphical modelling workbench by leveraging the Eclipse Modelling technologies, including EMF and GMF.

Characteristics

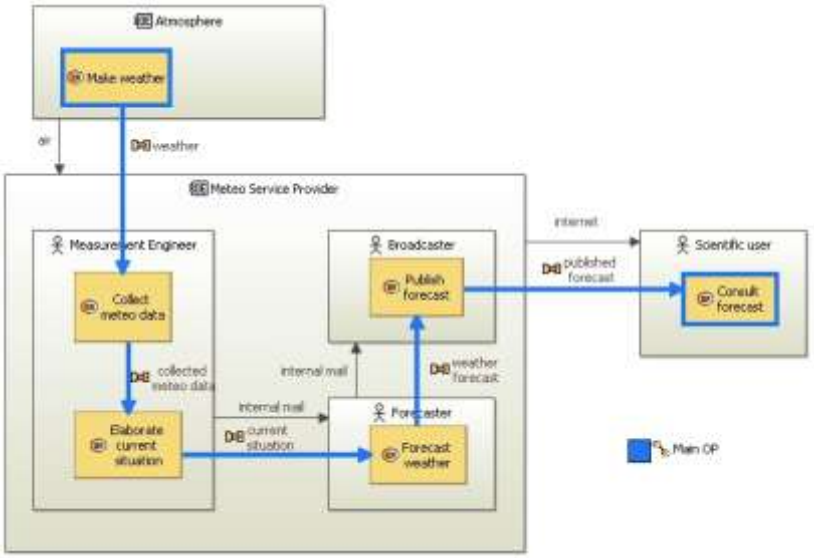
⇒ Create a graphical designer without writing any code

⇒ A graphical designer provides viewpoints adapted to the user's role or activity

⇒ A modelling workbench created with Sirius is composed of a set of Eclipse editors (diagrams, tables and trees) which allow the users to create, edit and visualize EMF models.

⇒ The editors are defined by a model which defines the complete structure of the modelling workbench, its behaviour and all the edition and navigation tools.

⇒ For supporting specific needs for customization, Sirius is extensible in many ways, notably by providing new kinds of representation, new query languages and by being able to call java code to interact with Eclipse or any other system.



IBM RSAD

Description

IBM Rational Software Architect Designer (RSAD) is a comprehensive design, modelling and development tool for end-to-end software delivery. It uses the Unified Model Language (UML) for designing enterprise JAVA applications and web services. RSAD is built on the Eclipse open-source software framework and is extensible with a variety of Eclipse plugins.

Characteristics

⇒ UML-based modelling support and model-driven development (MDD) tool help streamline the creation of Java and Web 2.0 applications and services.

⇒ Powerful tools and process guidance help reduce complexity and support higher quality and efficiency

⇒ Access to cloud service enables you to take advantage of scalable infrastructure services

⇒ A flexible, extensible platform helps you to deliver high-quality software with faster return on investment (ROI)

DIVERSITY

Description

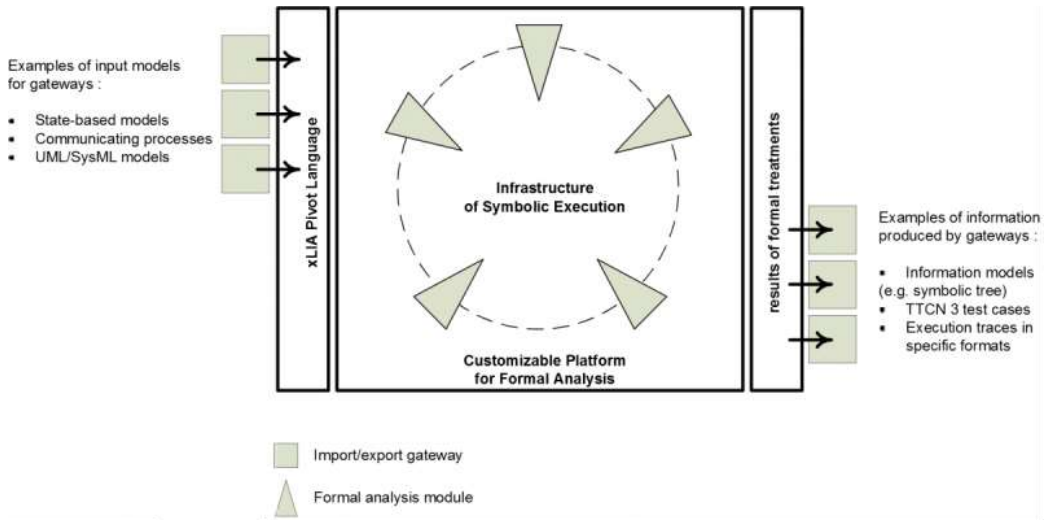
DIVERSITY is a multi-purpose and customizable platform for formal analysis based on symbolic execution which consists in executing programs, not for the concrete numerical values but for symbolic parameters.

Characteristics

⇒ The ability to analyse a wide range of (modelling) languages, this has led to the definition of a pivot language "xLIA" which stands for eXecutable Language for Interaction & Assemblage.

⇒ The ability to express a wide range of coverage objectives, such as structural coverage, condition/decision coverage MC/DC, inclusion criterion.

⇒ The customizability of the exploration strategies, in addition to the usual ones: Breadth First Search (BFS), Depth First Search (DFS), random traversal, weighted traversal, etc., DIVERSITY provides heuristics to alleviate the combinatory explosion problem when targeting a coverage objective.



Certifyit

Description

CertifyIT support business process and business rules representation to drive automated test generation. Test cases, requirements traceability and test scripts are automatically generated and maintained to accelerate your functional test cycles.

Characteristics

⇒ Formalize graphically the business process of the application-under-test

⇒ Define the application business rules and functional behaviour

⇒ Detects possible inconsistencies in the description of the expected application behaviour

⇒ Set priority scoring on the functional scopes to be tested

⇒ Ensure that the designed test cases properly cover the functional needs

⇒ Have all information and techniques to design your test scenarios ready at hand in CertifyIt

⇒ Automatically generate the test plan and all the test scenarios

⇒ Publish your test scenarios in script format for your test automation tool together with the reusable keyword library

⇒ Visualize the impact of a change in your scenarios and update your test repository at a click of a button

MISTA

Description

MISTA supports automated generation of executable test codes. It is suitable for function testing, acceptance testing, GUI testing, security testing and programmer testing.

Characteristics

⇒ It uses visual notation for building test models, such as function nets and finite state machines. Function nets, which are lightweight high-level Petri nets, can specify both control-oriented and data-oriented test models.

⇒ It provides test generators for comprehensive coverage criteria of test models, including reachability coverage, reachability with sneak paths, state coverage, transition coverage, depth coverage, goal coverage among others.

⇒ It supports a number of language (JAVA, C, C++, C#, PHP, Python, HTML and VB) and test frameworks (xUnit, Selenium IDE and Robot framework) for offline test execution.

⇒ It supports on-the-fly testing and online execution of generated tests through Selenium WebDriver or a RCP protocol (JSON-RPC or XML-RPC).

TITAN

Description

Titan is a TTCN-3 compilation and execution environment with an Eclipse-based IDE, TTCN-3 is a modular language specifically designed for testing

Characteristics

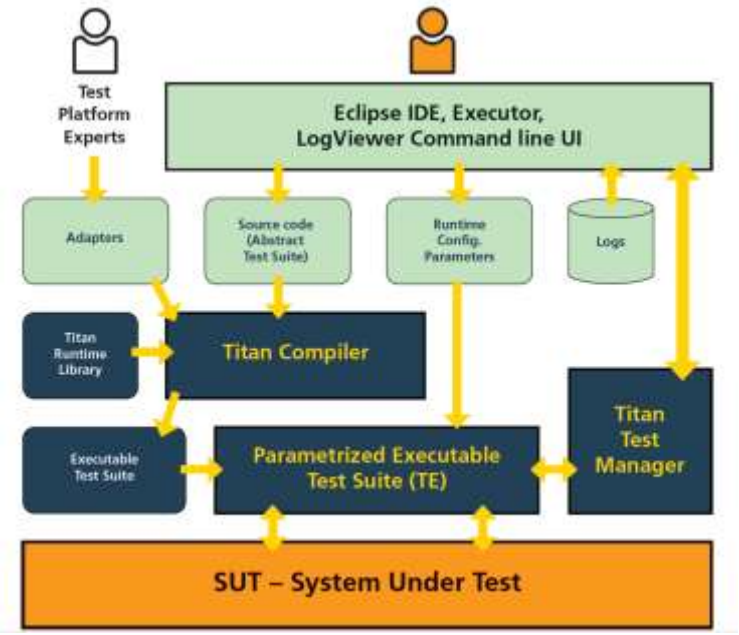
⇒ Eclipse plugin used for test execution and result reporting

⇒ The Titan main controller (MC) read and distributes to all test components the runtime configuration parameters.

⇒ It keeps up and running the test system, in case of a runtime error, Titan runtime control cleans up the test system, assigns an “error” verdict to the given test case and starts execution of the next test case

⇒ It produces logs in different formats which is done via different plugins

⇒ Is able to instrument the generated C++ code and output code coverage data in xml during runtime.



PragmaDev studio

Description

PragmaDev studio is a set of tools that helps specifiers, developers and testers to manage complexity in the development of today's system

Characteristics

⇒ Divided into 4 independent but integrated modules (PragmaDev Specifier, PragmaDev Developer, PragmaDev tester and PragmaDev tracer)

⇒ PragmaDev specifier: aims at helping system engineers, architects and specifiers to express their needs in a graphical and executable functional model

⇒ PragmaDev developer: helps software developers to define their software architecture and the concurrent behaviour of the different agents

⇒ PragmaDev tester: supports TTCN-3 international standard testing language including syntax and semantic verification, simulation, code generation, debug and graphical traces.

⇒ PragmaDev tracer: used for graphical requirements, properties and traces using a standard graphical representation.