Towards an Energy-aware Framework for application development and execution in Heterogeneous parallel architectures

Karim Djemame, Richard Kavanagh, Vasilios Kelefouras, Adrià Aguilà, Jorge Ejarque, Rosa M. Badia, David García Pérez, Clara Pezuela, Jean-Christophe Deprez, Lotfi Guedria, Renaud De Landtsheer, and Yiannis Georgiou

Abstract The Transparent heterogeneous hardware Architecture deployment for eNergy Gain in Operation (TANGO) project's goal is to characterise factors which affect power consumption in software development and operation for Heterogeneous Parallel Hardware (HPA) environments. Its main contribution is the combination of requirements engineering and design modelling for self-adaptive software systems, with power consumption awareness in relation to these environments. The energy efficiency and application quality factors are integrated in the application lifecycle (design, implementation, operation). To support this, the key novelty of the project is a reference architecture and its implementation. Moreover, a programming model with built-in support for various hardware architectures including heterogeneous clusters, heterogeneous chips and programmable logic devices is provided. This leads to a new cross-layer programming approach for heterogeneous parallel hardware architectures featuring software and hardware modelling. Application power consumption and performance, data location and time criticality optimization, as well as security and dependability requirements on the target hardware architecture are supported by the architecture.

Karim Djemame, Richard Kavanagh, and Vasilios Kelefouras
School of Computing, University of Leeds, UK e-mail: {K.Djemame,R.E.Kavanagh,V.Kelefouras}@leeds.ac.uk

Adrià Aguilà, Jorge Ejarque and Rosa M. Badia
Barcelona Supercomputing Center (BSC), Spain e-mail: {adria.aguila, jorge.ejarque, rosa.m.badia}@bsc.es

David García Pérez, Clara Pezuela
Atos Research & Innovation, Atos Spain SA, Spain e-mail: {david.garciaperez, clara.pezuela}@atos.net

Jean-Christophe Deprez, Lotfi Guedria, and Renaud De Landtsheer
Name, Address of Institute, e-mail: name@email.address

Yiannis Georgiou
Bull Atos Technologies, France e-mail: yiannis.georgiou@atos.net

1 Introduction

The emergence of new applications (as well business models) in the Internet of Things (IoT), Cyber Physical Systems (CPS), embedded systems, cloud and edge computing domains are transforming the way we live and work [1].

As the range of these applications continues to grow there is an urgent need to design more flexible software abstractions and improved system architectures to fully exploit the benefits of the heterogeneous architectures on which they operate, e.g. CPU, GPU, heterogeneous CPU+GPU chips, FPGA and heterogeneous multi-processor clusters all of which with various memory hierarchies, size and access performance properties. In addition to showcasing such achievement, part of the process requires opening up the technologies to an even broader basis of users, making it possible for less specialized programming environments to use effectively hiding complexity through novel programming models. Therefore, software plays an important role in this context.

On the other hand, computer systems have faced significant power consumption challenges over the past 20 years. These challenges have shifted from the devices and circuits level, to their current position as first-order constraints for system architects and software developers. A common theme is the need for low-power computing systems that are fully interconnected, self-aware, context-aware and self-optimising within application boundaries [8]. Thus, power saving, performance and fast computational speed are key requirements in the development of applications such as IoT and related computing solutions.

The project Transparent heterogeneous hardware Architecture deployment for eNergy Gain in Operation (TANGO) aims to simplify the way developers approach the development of next-generation applications based in heterogeneous hardware architectures, configurations and software systems including heterogeneous clusters, chips and programmable logic devices. The chapter will therefore present: 1) the incorporation of a novel approach that combines energy-awareness related to heterogeneous parallel architectures with the principles of requirements engineering and design modelling for self-adaptive software-intensive systems. This way, the energy efficiency of both heterogeneous infrastructures and software are considered in the application development and operation lifecycle, and 2) an energy efficiency aware system architecture, its components, and their roles to support key requirements in the environment where it runs such as performance, time-criticality, dependability, data movement, security and cost-effectiveness.

The remainder of the chapter is structured as follows: Section 2 describes the proposed architecture to support energy-awareness. Section 3-5 discuss key architectural components and their role to enact optimal, in terms of requirements and Key Performance Indicators (KPIs), application construction, deployment and operation, respectively. Section 6 presents related work. In conclusion, Section 7 provides a summary of the research and plans for future work.

2 System Architecture

The high level architecture is introduced on a per component basis, as shown in Figure 1. Its aim is to control and abstract underlying heterogeneous hardware architectures, configurations and software systems including heterogeneous clusters, chips and programmable logic devices while providing tools to optimize various dimensions of software design and operations (energy efficiency, performance, data movement and location, cost, time-criticality, security, dependability on target architectures).

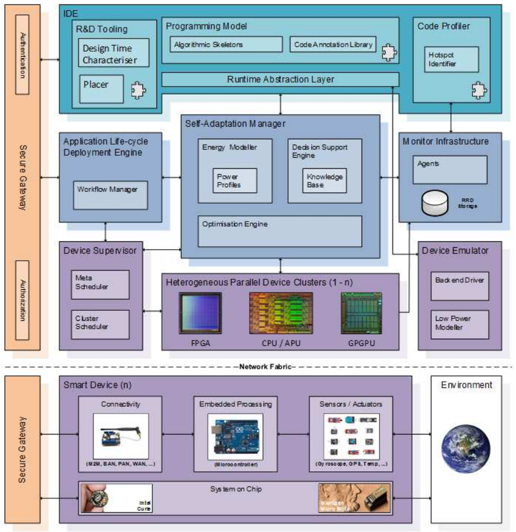

Fig. 1 Reference Architecture

Next, the architecture is discussed in the context of the application life cycle: construction, deployment, and operation. It is separated into remote processing capabilities in the upper layers, which in turn is separated into distinct blocks that support the standard application deployment model (construct, deploy, run, monitor, adapt) and local processing capabilities in the lowest layer. This illustrates support for secure embedded management of IoT devices and associated I/O.

The first block, *Integrated Development Environment (IDE)* is a collection of components to facilitate the modelling, design and construction of applications. The components aid in evaluating power consumption of an application during its construction. A number of plug-ins are provided for a front end IDE as a means for developers to interact with components within this layer. Lastly, this layer enables architecture agnostic deployment of the constructed application, while also maintaining low power consumption awareness. The components in this block are: 1) *Requirements and Design Tooling*: aims at guiding the development and configuration of applications to determine what can be targeted in terms of Quality of Service

(QoS), Quality of Protection (QoP), cost of operation and power consumption behaviour when exploiting the potential of the underlying heterogeneous hardware devices; 2) *Programming model (PM)*: supports developers when coding their applications. Although complex applications are often written in a sequential fashion without clearly identified APIs, the PM let programmers annotate their programs in such a way that the Programming Model Runtime can then execute them in parallel on heterogeneous parallel architectures. At runtime, applications described for execution with the Programming Model runtime are aware of the power consumption of components implementation, and 3) *Code Profiler*: plays an essential role in the reduction of energy consumed by an application. This is achieved through the adaptation of the software development process and by providing software developers the ability to directly understand the energy foot print of the code they write. The proposed novelty of this component is in its generic code based static analysis and energy profiling capabilities (Java, C, C++, etc. available in the discipline of mobile computing) that enables the energy assessment of code out-of-band of an application's normal operation within a developer's IDE.

The second block consists of a set of components to handle the placement of an application considering energy models on target heterogeneous parallel architectures. It aggregates the tools that are able to assess and predict performance and energy consumption of an application. Application level monitoring is also accommodated, in addition to support of self-adaptation for the purpose of making decisions using application level objectives given the current state of the application. The components in this block are: 1) *Application Life cycle Deployment Engine*: this component manages the life cycle of an application deployed by the IDE. Once a deployment request is received, this component must choose the infrastructure that is most suitable according to various criteria, e.g. energy constraints/goals that indicate the minimum energy efficiency that is required/desired for the deployment and operation of an application; 2) *Monitor Infrastructure*: this component is able to monitor the heterogeneous parallel devices (CPU, memory, network ...) that are being consumed by a given application by providing historical statistics for device metrics. The monitoring of an application must be performed in terms of power/energy consumed (e.g. Watts that an application requires during a given period of its execution), and performance (e.g. CPU that an application is consuming during a given period of its execution); 3) *Self-Adaptation Manager*: This component provides key functionality to manage the entire adaptation strategy applied to applications and Heterogeneous Parallel Devices (HPDs). This entails the dynamic optimisation of: energy efficiency, time-criticality, data movement and cost-effectiveness through continuous feedback to other components within the architecture and a set of architecture specific actuators that enable environmental change. Examples of such actuators could be: redeployment to another HPD, restructuring a workflow task graph or dynamic recompilation. Furthermore, the component provides functionality to guide the deployment of an application to a specific HPD through predictive energy modelling capabilities and polices, defined within a decision support engine, which specify cost constraints via Business Level Objectives (BLOs).

The last block above the network fabric line, addresses the heterogeneous parallel devices and their management. The application admission, allocation and management of HPDs are performed through the orchestration of a number of components. Power consumption is monitored, estimated and optimized using translated application level metrics. These metrics are gathered via a monitoring infrastructure and a number of software probes. At runtime HPDs will be continually monitored to give continuous feedback to the Self-Adaptation Manager. This will ensure the architecture adapts to changes in the current environment and in the demand for energy. Optimizations take into account several approaches, e.g. redeployment to another HPD, dynamic power management policies considering heterogeneous execution platforms and application energy models. The components in this block are: 1) *Device Supervisor*: provides scheduling capabilities across devices during application deployment and operation. This covers the scheduling of workloads of both clusters (Macro level, including distributed network and data management) and HPDs (Micro level, including memory hierarchy management). The component essentially realises abstract workload graphs, provided to it by the Application Life-cycle Deployment Engine component, by mapping tasks to appropriate HPDs; 2) *Device Emulator*:is responsible for delivering the initial mapping of the application tasks onto the nodes/cores (at compile time), i.e., which application task should run on each node/core. The mapping procedure is static and thus it does not take into account any run-time constraints or run-time task mapping decisions. The TANGO user can choose between a) a good solution in low time and b) a (near)-optimum solution in a reasonable amount of time (depending on the application complexity and on the number of the available nodes/cores). Emulation of the application tasks on the HPDs is necessary in order to compute the corresponding performance and energy consumption values. The novelty of the DE component is that it reduces the number of different emulations required by order(s) of magnitude and therefore the time needed to map the tasks on the HPDs.

Furthermore, a Secure Gateway supports pervasive authentication and authorization, which at the core of the proposed architecture enables both mobility and dynamic security. This protects components and thus applications from unauthorised access, which in turn improves the dependability of the architecture as a whole.

3 Application Development

3.1 Design-time Tooling: Guiding software design decision for exploiting heterogeneous hardware platforms capabilities

Designing and developing software with an efficient execution on distributed environment with fairly standard homogeneous processing devices is already a difficult exercise. This complexity explodes when targeting a heterogeneous environment composed not only of distributed multi-core CPU nodes but also including accelerators with many-core CPUs, GPUs and FPGAs. In the current era of heterogeneous hardware, software development teams thus face the daunting task of designing software capable to exploit underlying heterogeneous hardware devices available to the most of their capability with the goal to achieve optimal runtime and energy performance.

The algorithmic decomposition chosen to solve the problem at hand and the selected granularity of computing task determine software execution efficiency on a given underlying hardware hence affect time and energy performance. For instance, many algorithms exist for matrix operations, data sorting, or finding a shortest path in a graph. Developers already take into account data properties such as matrix sizes or degree of graph connectedness to select an algorithm with optimal time and energy performance. Nowadays, they must also consider capabilities offered by hardware in terms of parallel processing and data throughput. Such hardware capabilities influence design decision on algorithmic decomposition and task granularity choices to achieve efficient performance. For instance, time and energy performance associated with matrix multiplication on GPU or FPGA is directly influenced by matrix data sizes as well as the level of parallelism possible on each different kind of processing nodes as well as their clock speed, their memory capacity, their data transfer latencies, internally within the chip and externally through their I/O interfaces. In other words, the most appropriate algorithmic decomposition and task granularity is jointly influenced by data properties as well as the capabilities of the underlying heterogeneous hardware available.

In addition to designing software for today's operational conditions, developers must strike the right balance between achieving an optimal performance now and keeping a design implementation flexible and evolvable for tomorrow's new hardware. The most efficient algorithmic decomposition and task granularity for todays heterogeneous hardware and dataset properties might evolve. In the worst case, evolution in hardware or data properties impacts software design and architecture forcing developers to adapt drastically the application code, that is, another algorithm must be implemented in order to better exploit the new hardware or the new kind of data. In less radical situations, a given overall software architecture and algorithms can remain unaltered. Only the task granularity must be adapted to process larger quantity of data at once for instance. New technologies and programming model such as OpenCL or OmpSs/COmpS can facilitate accommodating task granularity changes without much effort hence keeping software implementation fairly evolvable. However, it still remains the job of developers to identify the appropriate task granularity for achieving improved time and energy performance and to provide this granularity information to the underlying technology or programming modelling tools. One of the goal of TANGO project is to provide design-time tooling to help developers to make insightful design decisions to implement their software so as to exploit the underlying hardware irrespective of the programming technologies and programming models chosen.

The initial approach to guide design decision, proposed in the first year of TANGO, relies on the rapid prototyping of the various simple software building blocks needed in a given application. The first step for developers consists of developing a set of simple prototypes for selected building blocks, for instance, for the different algorithms needed to solve multi-physics problems or to perform efficient image processing. Each prototype implements a particular algorithmic decomposition and task granularities for one of the identified simple software building blocks. For instance, a C or CUDA implementation of matrix multiplication will respectively target CPUs or NVIDIA GPUs nodes.

Developers can usually find alternative implementations of simple software building blocks that targets processors with fixed instruction set such as multi-core, many-core and GPU. These implementations rely on programming technologies such as MPI, OpenMP, CUDA or OpenCL. On the other hand, the use of FPGA and other reconfigurable hardware has so far remained more complex and only used by much fewer experts. To address this issue, the TANGO development-time tooling proposes a tool, named Poroto, to ease porting segments of standard higher-level code to FPGA. While OpenCL has recently proposed synthetisation for FPGA as part of its compilation tool chain, in many cases, developers only have implementation of simple building blocks in C code (or other programming languages). In such cases, an initial prototype implementation may be easier with Poroto than having to re-write the current C code in OpenCL. By annotating portions of C code with Poroto pragmas enables the generation of associated FPGA kernels and their interfacing to the code running on the CPU of the host machine through a PCI bus. The main processing program remains in C and is augmented with the necessary code, encapsulated in a C wrapper file that handles data transfer control to and from the offloaded FPGA computations. The C portion to be offloaded on FPGA is actually transformed into an equivalent VHDL program leveraging open source C to VHDL compilers such as ROCCC, PandA or other HDL code generation tools. Subsequently, the VHDL can be passed to the lower level synthesis tool chains from the particular FPGA vendor like Xilinx or Intel/Altera to generate to bitstream for a specific FPGA target. Concerning data transfer to/from the FPGA, Poroto currently relies on a proprietary technology. However an on-going TANGO effort consists of replacing this proprietary technology with RIFFA (an open source framework) to achieve similar data transfer operations.

Once the various prototypes of the different simple software building blocks have been implemented, compiled and deployed on the different targeted heterogeneous hardware, it becomes possible to obtain benchmarks with different representative datasets on each of the prototype variants. The benchmarking exercise is not restricted to FPGA implementation, the initial code of simple software block can be executed on multicore and many-core CPUs and if code also exists for GPU, it can also be included in the benchmarking exercise.

After time and energy benchmarks for the different prototype implementations of the various simple blocks have been collected from the execution on the different heterogeneous hardware targeted, developers must then identify an optimal way to place a combination of prototype implementations on the various hardware devices available in order to implement their complete solution. This optimisation problem between time and energy is not simple to solve in particular when considering different prototype implementations of several simple blocks competing for various heterogeneous hardware resources thus it becomes very useful to automate this optimisation exercise.

In TANGO, the development-time tooling relies on an open source optimisation engine originated from operational research named OscaR to search optimal ways to map the implementation of different software blocks on the different heterogeneous hardware nodes. Specifically, Placer finds optimal mappings of software component onto heterogeneous hardware, selects appropriate implementations of these tasks, and performs software tasks scheduling for optimising energy performance while meeting specified timing constraints. Placer is implemented on the top of the constraint programming engine of OscaR.

From an initial performance application design, it is then possible to further optimise application code by migrating from Poroto annotations to the ComPS and OmpSs programming model in order to achieve concurrent execution of an algorithm on different heterogeneous processing node. This programming model is presented in the next subsection.

3.2 Programming Model

To manage the implementation of parallel applications for heterogeneous distributed computing environments, TANGO programming model proposes the combination of two StarSs programming models and runtimes developed at Barcelona Supercomputing Center (BSC). StarSs is a family of task-based programming models where developers define some parts of the application as tasks indicating the direction of the data required by those tasks. Based on these annotations the programming model runtime analyzes data dependencies between the defined tasks, detecting the inherent parallelism and scheduling the tasks on the available computing resources, managing the required data transfers and performing the task execution. The StarSs family is currently composed by two frameworks: COMP superscalar (COMPSs) [5], which provides the programming model and runtime implementation for distributed platforms such as Clusters, Grids and Clouds, and Omp Superscalar (OmpSs) [9], which provides the programming model and runtime implementation for shared memory environments such as multicore architectures and accelerators (such as GPUs and FPGAs).

In the case of TANGO, we propose to combine these programming models in a hierarchical way, where an application is mainly implemented by a workflow of coarse-grain tasks annotated with the COMPSs Programming Models. Each of these coarse-grain tasks can be implemented as a workflow of fine-grain tasks developed with OmpSs. At runtime, coarse-grain tasks will be managed by COMPSs runtime optimizing the execution in a platform level by distributing tasks in the different compute nodes according to the task requirements and the cluster heterogeneity. On the other hand, fine-grain tasks will be managed by OmpSs which will optimize the execution of tasks in a node level by scheduling them in the different devices available on the assigned node.

This combination presents different advantages with respect of other approaches: First, it allows developers to implement parallel application in a distributed heterogeneous resources without changing the programming model paradigm. The programmer do not require programming model and APIs. It just require to decide which parts are tasks, the direction of its data and its granularity. Second, developers do not have to deal with programming data movements like in MPI. The programming model will analyze data dependencies and keep track of the data locations during the execution. So, it will try to schedule tasks as close as data or transparently doing the required data transfer to exploit the maximum parallelism. Third, we have extended the versioning and constraints capabilities of these programming models. With these extensions, developers will be able to define different versions of tasks for different computing devices (CPU, GPUs, FPGA) or combinations of them. So, the same application will be able to adapt to the different capabilities of the heterogeneous platform without having to modify the application. During the execution, the programming model runtime will be in charge of optimizing the execution to the available resources in a coordinated way. In platform scheduling the runtime will schedule the task in the different compute node resources, deciding which task can run in parallel in each node and managing that the different tasks are not colliding in the use of resources by the affinity of task to devices. At the node level, the runtime is in charge of scheduling the fine-grain tasks in the resources assigned in the platform level scheduling.

3.2.1 Application Implementation Example

An example of how an application is implemented with TANGO programming model is shown next. This example implements a matrix multiplication by blocks in two levels. The first level splits the matrices in blocks and computes the matrix multiplication by block. Each block multiplication is defined as coarse-grain task. Each matrix block can be decomposed in smaller blocks, and each block multiplication can be decomposed as a workflow of small block multiplications.

Figure 2 shows the main code of the benchmark application where a loop of the *multiplyBlocks* coarse-grain tasks is implemented.

Figure 3 shows the interface file where the developer can define the methods which are defined as tasks. In this case we have defined a task which has two implementations one which runs in 4 CPU cores and another which runs in a GPU.

Finally, Figure 4 depicts the implementation of the big block multiplication. In the first case, the fine grain tasks are the computation of the different elements of the resultant matrix block. In the second case, the big matrix block is decomposed in smaller block in order to fit in the GPU device memory and finer-grain tasks are defined as the multiplication of these small blocks. The fine-grain task in this case is the CUDA kernel defined by the Muld function.

```
int main(int argc, char **argv) {
  int  N = atoi(argv[1]);
  int  M = atoi(argv[2]);

  compss_on();

  cout << "Loading␣Matrices...\n";
  Matrix A = Matrix::init(N,Ml);
  Matrix B = Matrix::init(N,M);
  Matrix C = Matrix::init(N,M);

  cout << "Executing␣Multiplication...\n";
  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      for (int k=0; k<N; k++) {
        C.data[i][j]->multiplyBlocks(*A.data[i][k], *B.data[k][j]);
      }
    }
  }
  compss_off();
}
```

Fig. 2 Main workflow of the Matrix multiplication.

```
interface Matmul
{
  @Constraints(processors={@Processor(ProcessorType=CPU,
                                ComputingUnits=4)});
  void Block::multiplyBlocks(in Block block1, in Block block2);

  @Constraints(processors={@Processor(ProcessorType=GPU,
                                ComputingUnits=1)});
  @Implements(Block::multiplyBlocks);
  void Block::multiplyBlocks_GPU(in Block block1, in Block block2);
};
```

Fig. 3 Coarse-grain tasks definitions

4 Application Deployment

The application deployment is taking care by the Application Lifecycle Deployment Engine (ALDE) that takes cares of the following tasks: provide the application development tools information about the possible targeted architectures; build the application for different configurations of heterogeneous hardware architectures and libraries; prepare the application packets for deployment also, if possible, deploy the application to the targeted testbed; finally, if the connection with the device supervisor is possible, it will report and monitor the execution of the application to

```
void Block::multiplyBlocks(Block block1, Block block2) {
  for (int i=0; i<M; i++) {
    for (int j=0; j<M; j++) {
      #pragma omp task in(block1.data[i][0:M], \
                          block2.data[0:M][j]) out(data[i][j])
      for (int k=0; k<M; k++) {
         data[i][j] += block1.data[i][k]*block2.data[k][j];
      }
    }
  }
  #pragma omp taskwait
}

void Block::multiplyBlocks_GPU(Block block1, Block block2) {
  int NB = M/BSIZE;
  for (int i=0; i<NB; i++) {
    for (int j=0; j<NB; j++) {
      for (int k=0; k<NB; k++) {
        Muld(block1.data[i*NB+k], block2.data[k*NB+j],
             data[i*NB+j], NB);
      }
    }
  }
  #pragma omp taskwait
}

#pragma omp target device(cuda) ndrange(2, 64, 64, 32, 32)
#pragma omp task in(A[0:NB*NB], B[0:NB*NB]) inout(C[0:NB*NB])
__global__ void Muld(double* A, double* B, int wA, int wB,
                     double* C, int NB);
```

Fig. 4 Fine-grain tasks definitions

the user. Each one of these steps are going to be explained in more detail in the following paragraphs:

An installation of ALDE can register several testbed that can have a TANGO device supervisor or not. If the testbed does not have a device supervisor, the user or administration needs to input the hardware heterogeneous characteristics of it: RAM, CPUs, GPUs, number of nodes, etc. If the testbed has a TANGO device supervisor, ALDE will automatically connect to the testbed an recollect the node hardware information. This information will be exposed to the application development tools so they can notify the application developer the testbed heterogeneous capabilities also, some of the development tools could use this information to determine which is the best testbed to run a given application.

The building process of the application will be done by ALDE compiling the application for different combinations of targeted heterogeneous architectures and libraries. The usage of tools like EasyBuild [26] or Spack [16]. The different compilations could then be manually selected by the user of the self-adaptation manager to deploy the optimal code for the given available resources by the device supervisor.

After the application is compiled it needs to be packetized. The final packet format will depend on the targeted architecture. ALDE supports just submitting the application to the device supervisor by simple binaries (typical HPC scenario); It also supports the creation of containers based on Docker [14] or Singularity [15], this is both targeted to HPC and embedded environments that allow containers as an application distribution system; Finally, it also supports the generation of ISO images to be installed into heterogeneous embedded devices.

If the targeted heterogeneous architecture has an on-line device supervisor, ALDE has the possibility to connect to it and monitor the execution of the application. During the third year of the project, in this case, it is also expected that ALDE would supervise the data transfer to the selected architecture for the execution of the application.

5 Application Execution

5.1 Device Supervisor

The device supervisor (DS) is responsible for efficiently delivering the computing power of heterogeneous devices to the applications based on their needs. It provides the means to enable the execution of applications upon the platforms' resources. In particular, it offers a number of parameters that enable the fine specification and usage of different types of resources (CPUs, GPUs, Memory, etc) and their constraints for the optimal execution of the applications. Furthermore, it enables task placement and isolation upon devices during application deployment and operation.

Besides the various features and parameters for single application execution; this component allows the usage of the compute platform by multiple users where jobs may even compete for the same resources. Hence its main intelligence relies on resource selection techniques to find the most adapted resources to schedule the users' jobs while keeping a high system utilization and low fragmentation.

Within the Tango framework, the DS can get inputs from the Application Lifecycle Deployment Engine to execute jobs under particular parameters and it can follow the execution of the application and return intermediate state or final results to the ALDE. Optimization criteria (such as power consumption) and environment state are provided as input by the Self-Adaptation Manager and Monitoring Infrastructure components respectively.

The device supervisor in Tango is represented by Slurm [28] which is an open-source resource and job management system. Slurm performs workload management on five of the ten most powerful computers in the world of the Top500 list[1]

[1] https://www.top500.org/list/2016/06/

including the system ranked number two, Tianhe-2, which features 3,120,000 computing cores.

Slurm is specifically designed for the scalability requirements of state-of-the-art supercomputers. It is based upon a centralized server daemon, `slurmctld` also known as the controller, which communicates with client daemons `slurmd` running on each computing node. Users can request the controller for resources to execute interactive or batch applications, referred to as jobs. The controller dispatches the jobs on the available resources, whether full nodes or partial nodes, according to a configurable set of rules. The Slurm controller also features a modular architecture composed of plugins responsible for different actions and tasks such as: job prioritization, resources selection, task placement or accounting.

Most Resource and Job Management Systems today do not handle heterogeneous resources efficiently. They provide a complete SPMD (Single Program Multiple Data) support but limited MPMD (Multiple Program Multiple Data) support. Limited MPMD support means that even if users can specify different binaries to be used within a parallel job, all the tasks are currently associated with the same resources requirements. To be able to leverage all the benefits of platforms with heterogeneous resources we need to be able to specify different heterogeneous resources within the same job and be able to support the MPMD model. This support will enable users willing to harness different types of hardware resources inside the same MPI application, having part of their code run on GPUs while other parts are executed on standard CPUs with specific low amount of memory and a last part on CPUs with large amount of memory. Currently we are obliged to request the most complete set of resources for each task wasting some of the hardware with tasks that will not need all of them. In some cases, the total configuration required to run such a job does not even exist as all the nodes of the cluster may not provide all the hardware features.

Hence, the device supervisor component of Tango will be represented by an enhanced version of Slurm resource and job management system specifically designed to support heterogeneous resources.

5.2 Energy Modeller

Energy modelling can be used at multiple phases of an application life cycle. At deployment time it helps with the assignment of resources to an application and at runtime it aids a continuing energy mitigation strategy.

The Energy Modeller (EM) provides power and energy consumption information for compute devices in the current, future and historical contexts. Thus providing key information that guides the selection of the most appropriate configuration of an application within a heterogeneous environment, with the aim of minimising energy consumption. Acting as a key advisory component in the energy reduction process. It provides the mathematical models that estimate the power consumption and energy usage of a given deployment decision. Thus it is able to advise and drive the selection of hardware for service deployment and advise the process of self-adaptation.

The energy modeller's facility to assess historic energy consumption forms the heart of any advisory service for end users who wish to understand the energy consumption of their application. The advice to end users goes further by informing them of the current power consumption of their software and hardware setup, thus they can gauge the current impact of running their applications.

The energy modeller requires the use of models to determine from a hosts resources usage the likely future energy consumption, as well as providing a means of attributing power consumption to a particular application.

An estimation of the power consumption of an application or physical resource derives from two aspects. The first is the correct profiling of the resources characteristics, encompassing aspects such as its idle energy consumption and energy consumption under various load conditions. The second aspect is the profiling of the workload to be performed. This workload derives from the application that is to be characterised based upon the hardware it runs upon. These two profiles combined therefore advances the understanding of how much energy a application is expected to consume in the future.

The aspect of correctly charactering, resource takes care during the calibration process. The calibration process must provide repeatable conditions that generate a sequence of precise loads on the physical host undergoing measurement. The aim is to tightly control the environment while running an experiment to gain an accurate mapping between the resource utilisation and power consumption. This data can then be used as the basis of predicting future power consumption/energy usage, attributing power to a given workload as well as providing faster and more responsive measures of current power consumption especially for short runs of an application. The process of applying fixed calibration loads is illustrated in Figure 5.

A sequence of runs (marked as (a)) are shown with increasing utilisation, with small gaps between each run. The duration (a) can be chosen based upon any averaging window of the reported sensor data. A longer time period (a) gives a greater chance of the reported utilisation and power level stabilising. Issues such as: averaging, unsynchronised metrics, network delay, or caching mechanisms can all have their effects on calibration accuracy. A key solution to this is to discard values at the start and end of an experimental run (indicated by (c) in Figure 5). In addition to this, it eliminates experimental error such as load spike above the intended target load when each run starts. The final set of datapoints in the area indicated by (b), represent the best calibration data. One advantage of a model is that Once it is calibrated, even if the power measurement sensor reports an average value, an instantaneous estimated power consumption value can be obtained without averaging and at a higher temporal granularity through the model.

The second aspect of attributing power to a given application needs reasonable way to allocate power consumption. One such way is to consider the system's idle energy usage as well any active power consumption, given a specific application's load. The idle energy/power consumption should be evenly distributed among the applications that are running upon the host machine. The remaining energy is then

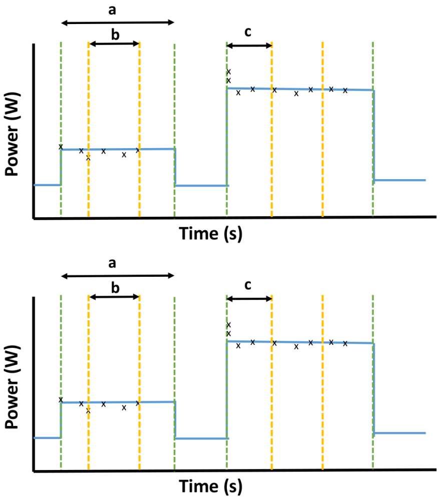

Fig. 5 The construction of artificial traces for Calibration

allocated based upon the induced load. This is described in Equation 1 where EU_P_x is the application's power consumption, $Host\_P$ is the measured host power consumption. EU_Util_x is the application's CPU utilisation, EU_Count is the count of applications on the host machine. EU_Util_y is the CPU utilisation of a member of the set of applications on the named host. $Host\_Idle$ is the host's measured idle power consumption.

$$EU\_P_x = Host\_Idle + (Host_P - Host\_Idle) \times \frac{EU\_Util_x}{\sum_{y=1}^{EU\_Count} EU\_Util_y} \tag{1}$$

5.3 *Self Adaptation Manager*

The Self-Adaptation Manager (SAM) is the principle component in the middleware for co-ordinating self-adaptation. It plays an essential role in maintaining power, energy and performance and goals of an application at runtime. Its primary focus is upon providing the Infrastructure runtime self-adaptation capabilities with a particular focus on trade-off management for the applications. This is achieved through the careful consideration of violations in service quality and the actuators that can be utilised to perform self-adaptation. This adaptation covers both macro and micro aspects of an applications deployment upon heterogeneous parallel architectures. The levels differ in that deployed applications may be submitted to a given node, but the node also has heterogeneity in that various accelerators might be utilised and configured for usage by the varying applications that are running.

The self-adaptation manager follows a MAPE [13] control loop pattern of monitor analyse, plan and execute. Adaptation in this cycle considers these main aspects:

- the varying level of QoS required, mainly either real-time high quality of service, or best effort services
- the various implementations of an application, which will have means to use various accelerators
- the performance of each implementation on the accelerators (affinity towards an adaptor)
- the availability/demand for accelerators and resources
- the malleability of an application
- the required pace of response (how quickly change occurs and real time requirements)
- the acceptable frequency of adaptation (avoiding over adaptation)

These collectively will give an application an affinity towards various accelerators and application configurations. QoS in Tango is formed of two distinct categories of application. The first is real-time applications that require a high level of quality of service, i.e. that they have priority to resources while, best effort services are expected to comprise the rest of the tasks.

Applications are expected to have various different implementations, each of which will be able to be executed on only some of the accelerators. The availability of these variations will be important as it offers the possibility for the SAM to select and switch between actuators dependent upon their availability. The quality of each implementation varies and will depend upon if the implementation can be structured in a way that takes advantage of the accelerator. This will give varying degrees of speed up, which will give a notion of affinity of an application to a given adaptor. The adaptors are a limited valuable resource that is not in all cases shareable. Fine grain pre-emption in NVidia GPUs has only become available in the Pascal architecture, which was released in mid-2016 [24]. Given the limited access to resources some applications may throughout their lifecycle be able to scale or shrink their resource usage.

A portion of jobs will be rigid and unable to change their resource requirements, while some others will be mouldable to the resources that are available at deployment. A further set will be malleable and will be able to dynamically change their resource requirements. This is particularly useful in regards to the availability of accelerators. Added to the limits of access to accelerators, some adaptations will be required to be completed with very limited delay, such as video processing. This places limits on the types of actuation that is permitted. This gives rise to a bias towards smaller control loops that handle specific QoS requirements. In cases of adaptation there is an acceptable frequency by which actuation is allowed to occur, such as once per minute. This is an important factor as in ensures applications are not interfered with and are allowed to perform useful work.

6 Related Work

Computing nodes are incorporating different types of devices in order to be more efficient when computing different types of applications by accelerating the computation at lower-power. However, this heterogeneity brings more complexity in the application development, each of these devices have their own programming language or API to spawn the computation in the different devices. For instance, for FPGA, they are traditionally programmed with the VHDL language; and for deploying and running the computation, developers have to use the tool chain provided by the FPGA vendor. A similar problem happens with the General Purpose GPUs. nVIDIA offers the CUDA framework [20] for programming and running applications in its devices and other vendors offer similar frameworks to do the same.

Current research is focusing their efforts in reducing the complexity of programming these heterogeneous nodes, as well as, providing portability between architectures allowing the reuse the code for similar devices. One of these examples is OpenCL [25]. It was born with the ambition of providing a common programming interface for heterogeneous devices (including not only GPUs, but also DSPs and FPGAs). With a syntax based on C, it has had a significant impact because the same code could be used in several accelerators. However, similar to CUDA, it requires the programmer to write specific code for the device handling, which reduces programmability. OpenACC [2] is another example of programming standard for parallel computing designed to simplify parallel programming of heterogeneous CPU/GPU systems. Based on directives, the programmer can annotate the code to indicate those parts that should be run in the heterogeneous device. The OpenMP standard [21] tackles the programmability issues in a similar way as OpenACC with regard the heterogeneous devices and also considers many other aspects of parallelism which makes it a stronger option.

However, these solutions are just managing the heterogeneity inside a node. If the application requires to run in several nodes (e.g. big amount of data or large parallelism), solutions mentioned before must be combined with other frameworks which manages the spawning of processes and data movements between the different computing nodes. Developers can attempt to do it by hand by using the TCP/IP and threading libraries, which require a lot of programming effort and skills or use one of the parallel distributed computing frameworks. One of this framework is MPI [18], which provides an API for interchanging data messages between the different processes for SPMD applications. Another option are PGAS programming models such as UPC [10], which allow to create a global address space and use shared memory programs in different nodes. Both options are working quite well running Single Process Multiple Data (SPMD) application in homogeneous clusters interconnected with a very fast network and SPMD application. However, in heterogeneous environments distributed across different locations they are to reaching good performance. For these reason we have proposed to combine COMPSs and OmpSs which have good results in managing heterogeneity at platform and node level and its programmability relies in the same task-based paradigm.

Prior to runtime considerations, developers must provide application code tailored for executing on heterogeneous hardware. Whether relying on MPI, OpenACC, OpenMP, OpenCL, CUDA or plain old C in application code, developers must craft their implementation, i.e., decompose their algorithms and identify granularity of subtask of these algorithms in order to exploit the available heterogeneous hardware devices optimally. While full development tool chains exist in open source for fast prototyping on standard multi- and many-core CPU [22] and on GPU [11, 19], no integrated tool chain is available in open source for fast prototyping offloading on FPGA. Poroto provides such an integration over existing open source frameworks such as ROCCC for the high level synthesis from C to VHDL. Future plans include support of pandA framework and also better integration with RIFFA for generic and portable handling of data transfers between the main CPU and the FPGA board. Regarding the optimal mapping of software component and scheduling of tasks using this software component, PREESM [22] and Silexica [23] both have studied the problem. However they are not publishing their ad-hoc algorithms. *Placer* is implemented on the top of the operational research optimisation framework OscaR, whose foundations have been validated in several Industry grade projects and products. Thus Placer only needs to implement additional problem specific code. This allows for high flexibility to support various requirements such as power cap, DVFS among others as well as better readability for verifying the correctness of the optimisation search algorithm.

Resource Management and Job Scheduling in traditional HPC systems is being performed by specialized software called RJMS. This software holds an important position in the HPC stack since it stands between the user workloads (jobs) and the hardware platform (resources). It is responsible for delivering computing power to applications efficiently. More than 2 decades of research and developments in the field has resulted into various open-source and proprietary versions of RJMS that exist today [6],[17],[29],[7],[4] offering basic and advanced functionalities to deal with HPC specialized platforms and workloads.

Since 2010 some newer generation schedulers such as Mesos [12] and Yarn [27] can execute both compute and data intensive workloads based on new types of internal architectures trying to deal with scalability, efficiency and fault-tolerance issues. In this group we can also add Flux [3] which is currently under active development and destined for extreme scale HPC systems.

7 Conclusions

This chapter has highlighted the importance of providing novel methods and tools to support software developers aiming to optimise energy efficiency resulting from designing, developing, deploying and running software on HPAs while maintaining other quality aspects of software to adequate and agreed levels.

The specification of a proposed architecture has been presented, which includes the architectural roles and scope of the components. This architecture complies with standard HPAs and supports an IDE, an application deployment on HPA environments, and heterogeneous parallel device environments. The design of the various architectural components was described, with emphasis on the requirements in order to support energy efficiency management, which is addressed during the complete life cycle of an application.

Future work includes the implementation of the capabilities to perform continuous autonomic self-adaptation during runtime. This leverages fine-grained monitored metrics of heterogeneous parallel devices and application software to create an adaptation plan supporting the performance and cost goals of an application. It is achieved through advances in modelling and prototyping that enable power, cost and performance awareness during operation through emulation and simulation under various ”what-if” scenarios.

8 Acknowledgment

This work has been supported by the European Commission through the Horizon 2020 Research and Innovation program under contract 687584 (TANGO project) by the Spanish Government under contract TIN2015-65316 and grant SEV-2015-0493 (Severo Ochoa Program) and by Generalitat de Catalunya under contracts 2014-SGR-1051 and 2014-SGR-1272.

References

1. Iot’s challenges and opportunities in 2017: A gartner trend insight report, April 2017.
2. OpenACC Application Programming Interface Specification. Web page at http://www.openacc.org/specification, (Date of last access: 3rd May, 2017).
3. Dong H. Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. Flux: A next-generation resource management framework for large HPC centers. In *43rd International Conference on Parallel Processing Workshops, ICPPW 2014, Minneapolis, MN, USA, September 9-12, 2014*, pages 9–17, 2014.
4. Altair. Pbs pro open source, 2017.
5. Rosa M Badia, Javier Conejero, Carlos Diaz, Jorge Ejarque, Daniele Lezzi, Francesc Lordan, Cristian Ramon-Cortes, and Raul Sirvent. Comp superscalar, an interoperable programming framework. *SoftwareX*, 3:32–36, 2015.
6. Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *5th International Symposium on Cluster Computing and the Grid (CCGrid 2005), 9-12 May, 2005, Cardiff, UK*, pages 776–783, 2005.
7. Adaptive Computing. Moab hpc basic edition, 2017.
8. K Djemame, D Armstrong, RE Kavanagh, JC Deprez, AJ Ferrer, DG Perez, RM Badia, R Sirvent, J Ejarque, and Y Georgiou. Tango: Transparent heterogeneous hardware architecture deployment for energy gain in operation. In *Proceedings of the First Workshop on Program Transformation for Programmability in Heterogeneous Architectures, arXiv:1603.01407*. arXiv preprint, 2016.

9. Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193, 2011.
10. Tarek El-Ghazawi and Lauren Smith. Upc: unified parallel c. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 27. ACM, 2006.
11. GPU Open Consortium. Code XL. Web page at http://gpuopen.com/compute-product/codexl/, (Date of last access: 17th May, 2017).
12. Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*, 2011.
13. IBM. An architectural blueprint for autonomic computing, 2005.
14. Docker Inc. Docker - a better way to build apps, 2017.
15. Gregory M. Kurtzer. Singularity, 2017.
16. Lawrence Livermore National Laboratory. Spack - package management tool, 2017.
17. Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems, San Jose, California, USA, June 13-17, 1988*, pages 104–111, 1988.
18. MPI Forum. Message Passing Interface Specification. Web page at http://mpi-forum.org/, (Date of last access: 3rd May, 2017).
19. NVIDIA. NVIDIA CUDA Toolkit. Web page at https://developer.nvidia.com/cuda-toolkit, (Date of last access: 17th May, 2017).
20. NVIDIA Corp. CUDA Homepage. Web page at http://www.nvidia.es/object/cuda_home_new.htm, (Date of last access: 3rd May, 2017).
21. OpenMP Architecture Review Board. OpenMP Application Programming Interface Specification. Web page at http://www.openmp.org/specifications/, (Date of last access: 3rd May, 2017).
22. M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*, pages 36–40, Sept 2014.
23. Silexica GmbH. SOFTWARE DESIGN FOR MULTICORE. Web page at https://silexica.com/, (Date of last access: 17th May, 2017).
24. R. Smith. Preemption Improved: Fine-Grained Preemption for Time-Critical Tasks, 2016.
25. John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73, 2010.
26. HPC UGent. Easybuild: building software with ease, 2017.
27. Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop YARN: yet another resource negotiator. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16, 2013.
28. AndyB. Yoo, MorrisA. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. 2003.
29. Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Softw., Pract. Exper.*, 23(12):1305–1336, 1993.