Modeling of applications and hardware to explore task mapping and scheduling strategies on a heterogeneous micro-server system

Lilia Zaourar, Massinissa Ait Aba, David Briand, Jean-Marc Philippe
CEA, LIST
Computing and Design Environment Laboratory
F-91191 Gif-sur-Yvette, France
{*lilia.zaourar, massinissa.aitaba, david.briand, jean-marc.philippe*}*@cea.fr*

***Abstract*—Many of todays important applications of our everyday lives, e.g. weather forecast, design of plane and car shapes, medical analysis or even search engine queries depend on massively-parallel computer programs that are executed in data centers hosting thousands of computers. A large amount of electrical energy is used to power them, and it is of primary importance to compute more efficiently to sustain the increasing demand of computing power while keeping energy consumption reasonable. One promising research path in this domain is heterogeneous systems. The rationale for that is that at least parts of applications execute more efficiently depending on the computing resource (processors, accelerators, etc.).**

Nevertheless, the exploitation of these heterogeneous platforms raises new challenges in terms of application management optimization on available computing resources. The aim of our work is to determine effective algorithms to exploit these heterogeneous platforms by finding the best mapping and scheduling of an application to optimize the execution time and energy consumption with respect to various constraints. To achieve this goal, there is a need of a detailed modeling of the applications and the underlying hardware to be able to find realistic solutions. In this paper, we propose such as model, provide two implementations with state-of-the-art tools and preliminary mapping and scheduling numerical results.

***Keywords*-heterogeneous computing, resource management, mapping, scheduling, energy consumption.**

I. Introduction

Nowadays, there is a huge use of services that need the execution of complex applications, such as weather forecasting, search engines, big data medical analyzes or intelligent transportation systems. These applications are studied, tested and simulated in data centers using massively parallel computing systems. Unfortunately, the energy consumption to maintain these systems is quite high and it is of primary importance to keep it reasonable. As an example, the U.S. data centers consumed an estimated 70 billion kWh (about 1.8% of total U.S. electricity consumption) [1].

Thus, computing servers powering these data centers must be built with prior energy efficient resource management. This focus on energy efficiency must also have as little impact on performance as possible. Heterogeneous systems offer the opportunity to achieve high performance while saving energy consumption [2], [3] and give an attractive alternative to massively parallel machines. In fact, today most High Performance Computing (HPC) systems are based on heterogeneous architectures and accelerators such as GPUS (e.g. Nvidia Tesla) or manycore architectures (e.g. Intel Xeon Phi). Even more heterogeneous systems can be found in the so-called micro-server systems which are designed to host fairly high computing power in a small form factor such as HP Moonshot or the Christmann RECS©|BOX [4].

However, taking advantage of such heterogeneous systems requires efficient use of resources to make profit from the performance of each part for application execution. In a heterogeneous environment, tasks have different execution times and power consumption when executed on different computing resources. Thus, one can change the performance and energy consumption of the system by using different resource allocations. In addition, each resource has its own ways of adaptation in order to optimize the performance per watt ratio such as dynamic voltage and frequency scaling. Thus, the exploitation of these heterogeneous platforms raises new challenges in terms of application management optimization on available resources. An important research challenge is how to assign the different tasks composing an application to the available resources in order to maximize some performance criterion of the heterogeneous architecture. This was one of the main objectives of the FiPS European project [5], which supported this work.

Tackling this challenge consists in determining effective strategies to exploit these heterogeneous platforms by finding the best allocation of tasks of an application to optimize the performance per watt ratio with respect to various constraints. For this, having the right models of applications and underlying hardware is paramount. Here, we consider a parallel application represented by a Directed Acyclic Graph (DAG) with precedence constraints between the tasks and a fully heterogeneous (in terms of computing resources and communications) micro-server platform. The mapping and scheduling problem consists in the assignment of tasks of the DAG to a set of heterogeneous Processing Elements, sequencing the order of tasks execution for each Processing Element such that precedence relationships between tasks are not violated, and orchestrating inter-node data transfers with the objective to optimize the performance per watt ratio.

The remainder of this paper is as follows: next section discusses related work on the problem we address. Section III describes preliminary experiments showing the potential gains regarding energy efficiency coming from heterogeneity. Problem formulation with mathematical model is detailed in Sections IV and V. Section VI provides an overview of resolution strategies. Then, Section VII analyzes our experimental results. Finally, conclusions and plans for future work are presented in Section VIII.

II. Related work

Since the performance of a parallel application running on a fully heterogeneous platform heavily depends on the mapping and scheduling of tasks composing the application onto the available processing elements of the system, it is crucial to properly handle tasks mapping and scheduling problem which remains a difficult challenge for this type of platforms. Thus, due to this key importance on performance, both problems have been extensively studied and numerous methods have been reported in the literature under various characterizations and hypotheses.

In [6] authors consider a task assignment problem for heterogeneous platform with identical communication links.The goal is to minimize the sum of the total execution time and communication costs. They proposed a heuristic approach and implemented several ones from the literature to compare with their strategy. In our work, we have fully heterogeneous platform with different communication costs. In [7], [8] authors propose a heuristic approach and an iterative algorithm to map tasks on heterogeneous systems with load balancing objective. Work in [9] propose compile-time task scheduling algorithms based on list scheduling, clustering strategy and task duplication for mapping and scheduling communications task graph on heterogeneous platform with respect to precedence constraints. They only deal with makespan minimization objective.

More recently, in [10], [11] authors model task scheduling problems in a heterogeneous system as bi-objective optimization problem between makespan and reliability. Also, in [12] authors consider the resource constrained scheduling problem as a bi-objective problem between makespan and robustness. They describe a weighted sum simulated annealing heuristic to find a solution. This differs from our research because they are not minimizing energy consumption.

Several works also exist in the literature of heterogeneous MPSoCs [13], [14] with embedded and real time constraints. The key difference of these works compared to heterogeneous HPC systems literature are that there is no effective real-time constraints. Since our objective is to find strategies to be executed during runtime, the execution time of the solution research algorithm will be of key importance to improve the efficiency of the system. Another point is that the architecture that will execute this algorithm will perhaps not be a high-performance CPU but an embedded CPU like an ARM core. The amount of processing power devoted to this algorithm will not be the same. This constraint will also have to be considered.

In [15] authors aim at founding a trade off between energy and system performance using classical genetic algorithm heuristic to find Pareto front solutions.

Other works attempt to formulate the problem as an allocation problem but they do not take into account all the constraints of this problem. As in [6], the authors address the tasks assignment problem on heterogeneous resources with identical communication links to minimize communication costs and total execution time. However, in this study, the capacity constraint is not expressed in the formulation of the problem. Thus, it is easy to obtain optimal solutions since it is enough to assign tasks to the least expensive resources without any capacity limitation (memory, etc.). Or in [16], the authors address the tasks assignment problem on Fully Heterogeneous computational resources taking into account capacity as well as communication links. The objective is also to minimize communication costs and the total execution time but in the problem formulation the uniqueness constraint is not expressed. This does not guarantee that each application task is assigned to a resource. Furthermore, in [17] the authors only express the objective function for minimizing load on the heaviest loaded processor among all assignments in heterogeneous platform without any uniqueness or capacity constraints. This is a very relaxed assumption compared to real problems. Although in these previous works the authors seek to optimize an assignment problem for heterogeneous platforms and the proposed resolution strategies are effective, the assignment problem is not properly handled and formalized. This results in non-realistic solutions or solutions that do not fit the real problem. Also many works deal with the problem of mapping parallel applications on heterogeneous platforms as an assignment problem without explicitly describing the corresponding mathematical model [18]–[20]. They only present algorithms for assigning resources without a clear formulation of the underlying problem which make their results difficult to exploit.

Because of the intractable nature of the task assignment problem, new efficient techniques are always desirable to obtain the best-possible solution within a reasonable amount of computation time. In order to propose solutions to this issue, having a mathematical model of the different data of the problem is a must-have. Regarding application representation, there exists a large body of the literature covering many task and heterogeneous computing models [21]. An application is often seen as a set of tasks, typically represented by a Directed Acyclic Graph (DAG) [22], [23] that reflects data flow model like Kahn Process Networks (KPN) [13] for example. Having a task graph representing the application is well treated within the FiPS methodology, as presented in [24].

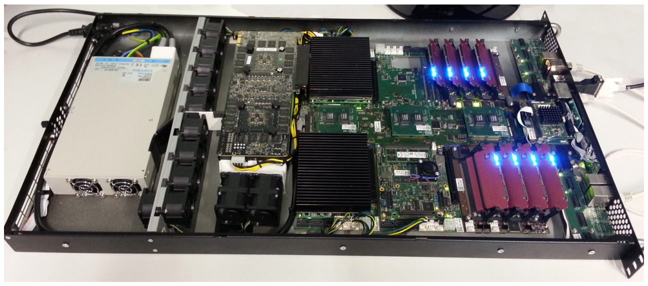

Figure 1. Picture of a RECS©|BOX server (two Intel CPUs, one NVdidia Tesla K80, eight Samsung Exynos ARM CPUs and a Xilinx Zynq 7045).

III. Presentation of server architecture and preliminary experiments

The abstract and introduction of this paper highlighted the availability of heterogeneous servers on the market. As an illustration, this section will briefly present an example of such a server that served to perform the experiments of this work. Then, preliminary experiments will show the interest of controlling task mapping and configurations of the resources to improve the performance/watt ratio.

A. Presentation of an heterogeneous server architecture

One goal of FiPS was to specify and design a new server architecture able to seamlessly host different types of computing architectures such as CPUs, embedded CPUs with possibly accelerators (e.g. ARM-based SoCs with GPUs or FPGAs), GPUs, FPGAs, or manycore accelerators. This task resulted in the RECS©|BOX server architecture [4].

An example of a configured server can host the following computing architectures (see Figure 1):

- Generic high-performance CPUs: Intel Core-I7 4600EQ nodes with 16GB of DDR3L memory and a 1TB SSD.
- GPU: one Tesla K80 from Nvidia (24GB memory).
- Embedded CPU: two ARM baseboards, each of them with four Apalis ARM nodes built by Christmann around a Samsung Exynos 5250 chip (dual ARM A15 processor with a Mali-T604MP4 GPU).
- FPGA: prototype board from University of Bielefeld, based on a Xilinx Zynq XC7Z45 (dual ARM A9 tightly coupled with the reconfigurable logic).

The nodes can communicate with a central switch using $1GbE$ netboards (management network) or using an external switch to optimize communication for computations. These two networks are interesting since additionally to out-of-band management, the internal network has sufficient performance (bandwidth and latency) to enable the configuration of resources as well as the transmission of information between schedulers and controllers.

B. Preliminary experiments

The RECS©|BOX infrastructure allows to perform experiments to show the influence of many variables on the performance/watt ratio. In heterogeneous servers, two main possibilities can be envisioned to optimize the execution of a given algorithm: the mapping of the tasks on a particular resource or the configuration of the execution of that task (such as increasing the parallelism or configuring the parameters of the resource such as DVFS - Dynamic Voltage and Frequency Scaling - behavior). Execution time and power consumption were measured for different resources under different conditions such as the runtime software and parallelism or the policy of the DVFS management (using either the Intel PState driver on the Intel processors or the cpufreq framework on the ARM platforms).

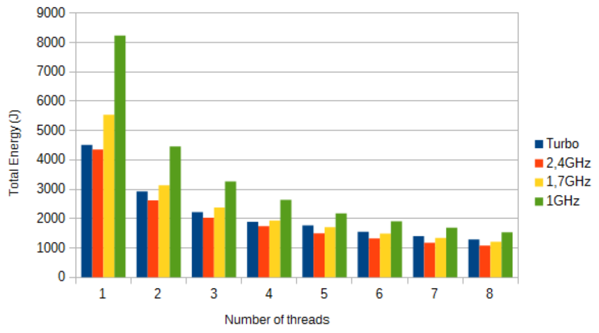

Figure 2. Energy at the node level for computing the Sysbench testbench on a I7-4700EQ processor with respect to the number of involved threads.

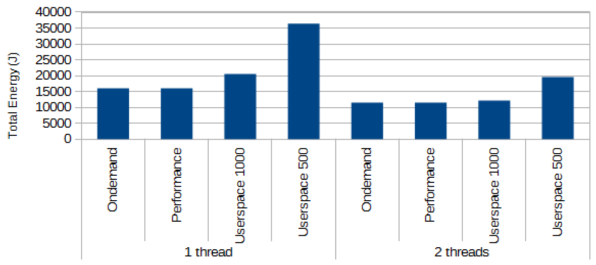

Figure 3. Energy at the node level for computing the Sysbench testbench on a Samsung Exynos 5250 processor with respect to the number of involved threads and DVFS governor policies and frequencies (MHz).

1) Energy efficiency on testbenches: The first set of experiments were done using benchmark applications. The compute benchmark of Sysbench (using cpu-max-prime=100000 option) was used to compare Intel and ARM processors. With standard configurations for DVFS mechanisms, the best configuration for both platforms is when the number of application threads is equal to the number of virtual cores thus 8 for the Intel and 2 for the ARM SoC. When playing with DVFS policies, one can see that the best configurations regarding the total energy spent are the one without Turbo mode on Intel (8 threads) (see Figure 2) and the one with the ”Performance” governor on ARM (2 threads) (see Figure 3).

For more insights about power consumptions, the Tur-

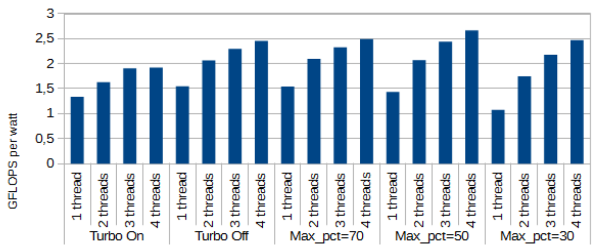

Figure 4. Power efficiency in GFLOPS/watt (node level) with respect different configurations of the application (number of threads) and of the computing resource (maximum reachable CPU frequency).

bostat utility was used to estimate power consumptions of the package and the cores of the Intel processor for the previous experiments. An interesting conclusion here is that the energy efficiency of the cores is increasing when the clock frequency is decreased. Thus, the energy gains coming from decreasing the core clock frequency is hidden by the host system power consumption. One issue with Sysbench is that it does not stress the processor that much. The power consumption of the Intel node was less than 35W during all the experiments. Since the measured power consumption is the one from the node, the overall system integration does not have to be neglected.

Using a more compute-bound application such as the Intel Optimized Linpack (enabling the node power consumption to reach up to 78W before the thermal protection activates), one can see in Figure 4 that the best configuration (maximizing the GFLOPS/W ratio) is when the processor is at 50% of its maximum speed with four threads (2.6 GFLOPS/W). Compared to the standard configuration (Turbo mode enabled), the gain is 40% (1.9 GFLOPS/W). Of course, the speed of computation is reduced, but with the advantage of lower power consumption. For highly compute-bound applications, lowering the clock speed can greatly benefit to the power efficiency, since the node infrastructure is dominated by the strategies applied to the processor itself.

2) Real applications: Neural Networks: Neural networks are widely used for applications involving recognition since they perform quite well as understanding the content of pictures, sounds, etc. The use of deep learning in data centers is growing (Facebook, Twitter, Microsoft, Google, Baidu, etc.). Two neural networks (AlexNet and a simple Convolutional Neural Network for embedded image recognition) were ported on several platforms of the RECS©|BOX server, with different runtime softwares (OpenCL and OpenMP) and under different conditions (frequencies for CPUs and GPUs, number of threads, batch size for GPUs, etc.).

AlexNet [25] is quite famous for having won the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) by introducing deep learning in this competition. AlexNet is a very complex network composed of several layers and each layer featuring different operators. The complexity is about 771 millions of Multiply-Accumulate operators. On the Intel processor, the most efficient OpenCL implementations were the ones without Turbo mode and the one at 70% of the maximum achievable frequency, with turbo mode enabled. Compared to the default configuration, the performance/watt ratio increases by 18%. For the K80-OpenCL implementation, the network is so complex that the host CPU is waiting the GPU, thus the frequency of the host CPU can be reduced without any noticeable penalty on the overall performance. The best performance/watt ratio (0.208 stimuli/sec/watt) is obtained when all frequencies (CPU and GPU) are set to the minimum (30% for the CPU and 324-324MHz for respectively the cores and memories of the K80). The gain is compared to the default configuration is however below 4%. Without any batch, the GPU implementation is 2.55 times more efficient compared to the CPU and 7.44 times more efficient compared to the ARM-OpenMP best configuration, due to the size of the network.

The complexity of a network such as AlexNet is not needed by all applications. Simple convolutional neural networks (such as the one shown in Figure 5) can also be of great interest to classify images in embedded applications. This network has a low complexity of 205.000 MAC operations. Additionally to the implementations described above, the MALI GPU of the ARM SoC was also tested using OpenCL. Without entering too much into details, forcing the most efficient configuration of the PState driver on the Intel CPU can improve the performance/watt ratio by 11% of the OpenCL implementation (which is more power-efficient compared to the OpenMP one). Without any batch mode, using the K80 for such a network is not very interesting since the CPU version is more efficient (46 versus 120 stimuli/sec/watt respectively). The default configuration of the platform only achives 37 stimuli/sec/watt (as a comparison, a batch of 100 stimuli reaches more than 380 stimuli/sec/watt). Regarding the ARM SoC, the best configuration without any batch is when using the OnDemand governor with a 2-thread OpenMP configuration (100 stimuli/sec/watt). When batch is available (ability to load images in advance), the MALI OpenCL implementation reaches 175 stimuli/sec/watt when limiting the frequency of the ARM cores to 1GHz.

All these experiments confirmed the potential gains if appropriate configurations are chosen for each task implementation and for the resource that executes them. Next sections introduce a formulation of the overall problem.

IV. Application and hardware modelling

This section presents the formalization of inputs (platform and application), outputs (decision variables) to enable a detailed formalization of the problem.

A. Modeling of the inputs

Inputs are provided by the FiPS flow during design time: the platform configuration and the application partitioned

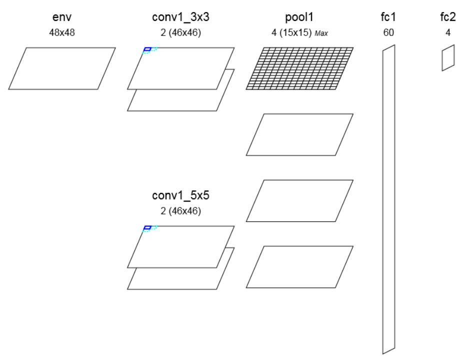

Figure 5. Structure of a simple CNN.

into tasks. Resources states will be given by observations from the different sensors of the system, as presented in section III. We recall here the set of data needs for the mathematical model.

1) Platform model: let $PE = \{PE_1, ..., PE_m\}$ a set of m Processing Elements ($M = \{1, ..., m\}$) in the heterogeneous computing system. Each Processing Element PE_k represents a CPU node in the system. In our work, accelerators (GPU, FPGA, etc.) are seen as resources of the CPU node since use of the accelerator generally involves its host CPU (especially when using message passing between the nodes). This assumption enables better modeling of the server behaviour by decoupling the host CPU from its accelerator. The k index reflects the geographic location of this node in the server, not its type (it can also be seen as a unique identifier of the processing element). To model the properties of these resources, the following data is provided:

- R_{PE_k} : the set of resources offered by the Processing Element PE_k (e.g., memory, processing cores, accelerators, etc.). This is an abstract data composed of the different resources offered by the Processing Element.
- $C_{PE_k,r}$: the capacity of the Processing Element PE_k for resource r, $r \in R_{PE_k}$ (e.g., memory capacity, processing capability such as the number of cores, etc.). The main idea of this data is that it will provide constraints on the execution of a task onto the resource.
- $B_{PE_k,PE_{k'}}$: represents the available bandwidth between Processing Elements PE_k and $PE_{k'}$. This bandwidth can be the one from an Ethernet link or the bandwidth of a shared memory if $k = k^{'}$.
- $L_{PE_k,PE_{k'}}$ represents the latency between Processing Element PE_k and $PE_{k'}$. For example, latency cost can be 0 for a communication if $k = k^{'}$.

2) Application model: we consider an application $a \in A$ modeled by a directed acyclic task graph (DAG) $G = (V, E)$, where V is the set of tasks $t_i, i \in \{1, ..., n\}$ to be executed and E represents precedence constraints between two tasks t_i and t_j. The weight of the arc $e_{i,j}$ noted by $\omega_{i,j}$ represents the volume of communication between t_i and t_j. We denote by $prec(i)$ the list of predecessor for each task $t_i, i \in \{1, ..., n\}$.

The execution model of the DAG is close to the macro data flow model where a task t_i can be executed only after all its predecessors have terminated and when communications from its predecessors and this task have been performed. We do not allow task duplication, or preemptions. Since a task $t_i, i \in \{1, ..., n\}$ can have several implementations depending on the execution resource. A particular implementation of the task t_i for each or some of the PEs available on the heterogeneous system provides us the following data:

- $exec_{ik}$: defined on $E \times M \to \mathbb{R}$, the execution time of task t_i on Processing Element PE_k.
- P_{ik} defined on $E \times M \to \mathbb{R}$, the average power consumption for the execution of task t_i on Processing Element PE_k.
- $energ_{ik}$: the energy consumption of task t_i on Processing Element PE_k, $energ_{ik} = exec_{ik} \times P_{ik}$.
- $\omega_{i,j}$: expresses the communication rate between task t_i and task t_j, $i, j \in \{1, ..., n\}$.
- $consum_{t_i,r_k}$: the consumption (not necessary power) of task t_i in resource r of the Processing Element PE_k when executing. For example, memory needs mem_{ik} is seen as such a resource.

These data will enable us to derive a system of inequality to be solved for finding an optimal solution.

B. Modeling of the outputs - Decision Variables

The expected result is an optimal assignment of all application tasks on available resources. Multiple tasks can be assigned to one particular resource thus we also need to schedule the execution of these tasks. Therefore, for each task, we need to decide a time at which its execution is started and a processing element that is responsible for its execution. Thus, decision variables will be defined as the following:

$$X_{i,k} = \begin{cases} 1 & \text{if task } t_i \text{ is placed on the Processing} \\ & \text{Element } PE_k. \\ 0 & \text{otherwise} \end{cases}$$

St_i : the starting time of the task $t_i, i \in \{1, ..., n\}$.

Then, objective functions must be defined so as to model the objectives of the optimization process.

V. Mathematical formulation

The objective of this section is to derive from the above-presented data the different equations that model the problem from a mathematical point of view.

A. Objective Functions definition

The global objective is to optimize the performance/watt ratio for application execution. Therefore, the problem is decomposed into two objectives: energy consumption and execution time. Unfortunately, the goals of saving energy and achieving high performance often conflict with each other. In fact, resource allocations using more energy will allow one to achieve greater performance, while resource allocations trying to conserve energy will reduce system performance. Thus we have to find the trade-offs between energy consumption and computing performance, we model this dilemma as a bi-objective optimization problem.

To measure system performance, we examine the makespan (denoted by C_{max}) of a batch of tasks for a given resource allocation. Makespan is the total amount of time it takes for all the tasks in the batch to finish executing across all Processing Elements. Energy is measured in number of joules consumed by that same batch of tasks for a given Processing Element. An optimal resource allocation would be one that minimizes both makespan and energy consumed.

Here, we present the analytic formulation for these two objectives functions.

- Minimize the energy consumption of task executions:

$$Min(Z_1) = \sum_{i=1}^{n}\sum_{k=1}^{m} energ_{ik} X_{i,k} \tag{1}$$

with : $$energ_{ik} = exec_{ik} \times P_{ik}$$

- Minimize the total execution time which is expressed here by minimizing the makespan (C_{max}):
We need first to calculate the earliest completion time $C_i^{earliest}$ which is equal to the minimum time to start a task plus the time to execute it.

$C_i^{earliest} = \min_{k \in M}\{St_i + exec_{ik} X_{i,k}\}$.
Then the makespan is $C_{max} = \max_{i \in V} C_i^{earliest}$.

$$Min(Z_2) = C_{max} \tag{2}$$

This final objective function is the one that will give us the best trade off between energy consumption and system performance. Since the problem deals with the physical world, constraints must be defined.

B. Constraints

two different set of constraints are considered. The first set (3 and 4), deals with the mapping constraints and the second one (5 and 6 and 7) with the scheduling constraints.

Unicity: constraints of type (3) simply express that each task $t_i, i \in \{1, ..., n\}$ must be assigned to one and only one Processing Element:

$$\sum_{k} X_{i,k} = 1, \forall i, \forall k \tag{3}$$

Capacity constraint: constraints (4) require that the node capacity is not exceeded when a task t_i is executed on it:

$$\sum_{k} consum_{t_i,r_k} X_{i,k} \leq C_{PE_k,r}, \forall i, \forall r \tag{4}$$

Task execution time: the set of constraints (5) expresses that the C_{max} (the makespan) corresponds to the total length of the schedule (i.e. the maximum of time when the tasks have finished processing).

$$St_i + \sum_{k=1}^{m} exec_{ik} X_{i,k} \leq C_{max}, \forall i \tag{5}$$

Task execution model: the set of precedence constraints (6) describes that each task t_j which follows a task t_i must be carried out after the starting time of the task t_i, plus the execution time of its task and communications time.

$$St_i + \sum_{k=1}^{m} exec_{ik} X_{i,k} + \sum_{k=1}^{m}\sum_{l=1}^{m} Com_{ik,jl} X_{i,k} X_{j,l} \leq St_j, \quad \forall i \in prec(j) \tag{6}$$

We suppose here that the communication time between two tasks t_i mapped on processing element PE_k and t_j mapped to on Processing Element $PE_{k'}$ is equal to :

$$Com_{ik,jl} = \frac{\omega_{i,j}}{d_{PE_k,PE_{k'}}} + L_{PE_k,PE_{k'}}$$

Precedence: disjunctive constraints (7) asserts that two tasks t_i and t_j assigned to the same Processing Element PE_k must not overlap. We use here B, a big value such that the constraints hold only for the tasks assigned to the same processing element (i.e. $X_{i,k} = X_{j,k} = 1$).

$$\begin{aligned} St_i + exec_{ik} - St_j &\leq B(1 - X_{i,k}X_{j,k}) \vee \\ St_j + exec_{jk} - St_i &\leq B(1 - X_{i,k}X_{j,k}), \forall i, j, k \end{aligned} \tag{7}$$

Finally, Figure 6 depicts the mathematical program for mapping and scheduling an application represented by a DAG on a heterogeneous system. This optimization problem can be classified as Mixed Integer Quadratic Constrained Program (MIQCP) because constraints (6) and (7) are quadratic but the objective function and all the other constraints are linear and mixed variables. Based on this problem formulation, different solving methods can be investigated as presented in section VI.

VI. RESOLUTION

In terms of resolution strategies, there are two common ways to tackle task mapping and scheduling problems: the exact methods which are limited in the number of instances that can be solved in a reasonable amount of time and the heuristic approaches that provide fast and effective means for finding approximate solutions, aiming for a near-optimal solution. Thus, as a first approach, we propose here both an exact method and a heuristic approach. We perform this study in a *static* and *off-line* environment, so that we can evaluate the mapping and scheduling and analyze the trade-offs between the two objectives.

$$(MIQCP)\begin{cases}\sum_k X_{i,k}=1,\forall i\\ \sum_k conso_{t_i,r_k}X_{i,k}\leq C_{PE_k,r},\forall i,\forall r\\ St_i+\sum_k exec_{ik}X_{i,k}\leq C_{max},\forall i\\ St_i+\sum_k exec_{ik}X_{i,k}+\sum_k\sum_l Com_{ik,jl}X_{i,k}X_{j,l}\leq St_j,\forall i\in prec(j)\\ St_i+exec_{ik}-St_j\leq B(1-X_{i,k}X_{j,k})\vee St_j+exec_{jk}-St_i\leq B(1-X_{i,k}X_{j,k})\ \forall i,j,k\\ X_{ik}\in\{1,0\},St_i\in\mathbb{R},B\ const\end{cases}$$

Figure 6. Mixed Integer Quadratic Constrained Program resulting from problem formulation.

A. Exact methods

Concerning exact methods, although the problem is $NP-hard$, some special instances are polynomial time solvable: two-processor systems in the time complexity of a maximum flow algorithm [26], tree DAG on heterogeneous networks in $O(nm^2)$ time [27] tree DAG on homogeneous networks in $O(nm)$ time [28]. These exact methods are unlikely to be used based on the characteristics of the FiPS problem.

For solving the introduced mathematical models (MIQCP), *IBM ILOG CPLEX* Optimization Studio [29] (often referred to simply as *Cplex*) was used. It is a commercial solver designed to tackle (among others) large scale (mixed integer) linear problems. $Cplex$ is now actively developed by *IBM*. We use here the *Cplex 12.5* version. The software also features several interfaces to connect the solver to different programming languages and programs. However, a stand-alone executable is also provided. The optimizer is also accessible through modeling systems.

However, the disjunctive quadratic constraint (7) is not supported even by using the quadratic version of the solver due to the non positive-definite matrix The linearizion of this constraint has to be performed as described in the next subsection. Concerning the constraint (6), there is no need to linearize it because the matrix $Com_{ik,jk'}$ is semi-defined positive and therefore the quadratic solver supports it.

1) Linearizion of the constraint: the disjunctive constraints (7) could be linearized in two steps as follow.

First we replace $B(1-X_{i,k}X_{j,k})$ by $B(2-X_{i,k}-X_{j,k})$, this is possible because the decision variables $X_{i,k}$ and $X_{j,k}$ are binary variables. We obtain the constraints (8) :

$$\begin{aligned}St_i+exec_{ik}-St_j\leq B(2-X_{i,k}-X_{j,k})\vee\\ St_j+exec_{jk}-St_i\leq B(2-X_{i,k}-X_{j,k}),\forall i,j,k\end{aligned}\quad(8)$$

Second, we introduce another set of binary decision variables $Y_{i,j}$. These new variables could be seen as the decision that either we chose to execute task t_i before t_j, in this case $Y_{i,j}=1$ or the opposite, so $Y_{i,j}=0$. The disjunctive constraints become the constraints (9) :

$$\begin{aligned}St_i+exec_{ik}-St_j\leq B(2-X_{i,k}-X_{j,k})+B(1-Y_{i,j})\vee\\ St_j+exec_{jk}-St_i\leq B(2-X_{i,k}-X_{j,k})+BY_{i,j},\forall i,j,k\end{aligned}\quad(9)$$

Finally, the following linear constraints (10) are obtained:

$$\begin{aligned}St_i+exec_{ik}-St_j\leq B(3-X_{i,k}-X_{j,k}-Y_{i,j})\wedge\\ St_j+exec_{jk}-St_i\leq B(2-X_{i,k}-X_{j,k}+Y_{i,j}),\forall i,j,k\end{aligned}\quad(10)$$

Therefore, program (MIQCP) is equivalent to (MIQCP'), depicted on Figure 7.

2) Objectives functions setting: unfortunately, *Cplex* solver does not support multi-objective problems. Thus, we set one objective each time with all the constraints, then we get two mathematical programs (P_{Energy}) that aims at minimizing the energy with all scheduling and mapping constraints while ($P_{Makespan}$) aims at minimizing execution time with respect to mapping and scheduling constraints too.

This allows us to know the optimal value of each objective (energy consumption and performance) under the whole set of constraints (mapping and scheduling) and then compare that value with the compromise between the two objectives. This is permitted by the fact that *Cplex* is able to qualify the optimality of a solution. We therefore use these values as an indication of bounds for energy and performance ratio. Numerical results will be presented in section VII.

B. Heuristic approach : local search

The literature regarding heuristics to solve mapping and scheduling problems is particularly large as they offer powerful tools in solving optimization problems. We choose here to explore the possibilities of local search heuristics to compare to exact methods. Local search is the paradigm of choice to tackle large-scale real-life optimization problems. In the context of the FiPS project, we need interactivity with decision support systems to adapt the results to customer requirements regarding performance per watt ratio (optimality is one thing, speed is also important).

For optimization approach, it means quickly obtaining good-quality solutions. Therefore, fast iterative improvement methods, like local search, are suited to satisfy such needs. When a solution space is gigantic like the set of application addressed in this work, a complete search becomes unpractical. Given a (possibly infeasible) solution to the problem, local search consists in modifying some parts of this one - that is, some decision variables - to reach a new, hopefully better solution. Exploring a so-called neighborhood of the incumbent has a major advantage: the new solution can be evaluated quickly through incremental calculation. Then,

$$(MIQCP')\begin{cases} Min(Z_1) = \sum_{i=1}^{n}\sum_{k=1}^{m} energ_{ik}X_{i,k} \\ Min(Z_2) = C_{max} \\ \sum_k X_{i,k} = 1, \forall i \\ \sum_k consum_{t_i,r_k} X_{i,k} \leq C_{PE_k,r}, \forall i, \forall r \\ St_i + \sum_k exec_{ik}X_{i,k} \leq C_{max}, \forall i \\ St_i + \sum_k exec_{ik}X_{i,k} + \sum_k \sum_l Com_{ik,jl}X_{i,k}X_{j,l} \leq St_j, \forall i \in prec(j) \\ St_i + exec_{ik} - St_j \leq B(3 - X_{i,k} - X_{j,k} - Y_{i,j}), \forall i,j,k \\ St_j + exec_{jk} - St_i \leq B(2 - X_{i,k} - X_{j,k} + Y_{i,j}), \forall i,j,k \\ X_{ik}, Y_{i,j} \in \{1,0\}, St_i \in \mathbb{R}, B\ const \end{cases}$$

Figure 7. Modified Mixed Integer Quadratic Constrained Program resulting from problem formulation.

local search can be viewed as an incomplete and nondeterministic but efficient way to explore a solution space.

To this end, we used *LocalSolver 6.0* [30], developed by *Innovation24*, for combinatorial optimization entirely based on local search with an extension to mixed-variable optimization. It implements modified local search heuristic, offering the power of local search through a model-and-run solver for large-scale $0-1$ nonlinear programming.

This solver addresses combinatorial optimization problems by performing autonomous moves which can be viewed as an assignment and scheduling applied to the polyhedron induced by boolean variables and constraints of the mathematical program (MIQCP). In addition, *LocalSolver* could handle a multi-objective problem. For this, the first objective of the mathematical program is the one it optimizes by priority. For the second objective, it attempts to satisfy it the best with all the constraints of the problem.

Figure 8 depicts the modified optimization problem $(MIQCP{-}LS)$ (for *LocalSolver*) of mapping and scheduling an application on a heterogeneous system with the objectives and constraints. Obtained results are reported and compared to previous approach using *Cplex* in section (VII).

VII. Numerical results

We have conducted several experiments in order to compare the approaches. In this section, we first detail the applications instances and platform used for the experiments. Then, we expose and comment the numerical results.

A. Applications instances

Two benchmarks were used in this study. The first one is based on generated instances and the second one on the Cholesky Matrix Decomposition kernel.

1) Generated instances: to compare the mapping and scheduling strategies described above, several application instances were generated using a custom Python-based graph generator. The main parameters that characterize the generated instances are described below:

- nb_tasks : sets the number of application tasks. Corresponds to the number of vertices n of the DAG.
- $nb\_processing\_elements$: sets the number of Processing Elements (m) in the heterogeneous system. This parameter allows to generate all tasks weights depending on the Processing Elements execution.
- id_task : represents the identifier of each task t_i.
- $execution\_time, power\_consumption$: represent task execution time $exec_{ik}$ and power consumption P_{ik} for each task t_i on each Processing Element PE_k. To best match with the real problem, the generator was modified to include a correlation between these two parameters to avoid irrelevant situations were a resource can perform the execution of a task very quickly without consuming power.
- $memory\_consumption$: corresponds to memory foot print of each task.
- $degree$: sets the degree of a vertex (task t_i). Represents the number of edges incident to the vertex. It will affect the list of predecessor $prec(i)$ for each task $t_i, i \in \{1, ..., n\}$.
- $\omega_{i,j}$: represents the volume of communication between two tasks t_i and $t_j, i, j \in \{1, ..., n\}$.

Figure 9 shows an example of a generated application represented by a DAG with 10 tasks.

2) Real instance - Cholesky Matrix Decomposition: the 256x64 use case of the Cholesky Matrix Decomposition was chosen has a real application instance. The weighted task graph presented in Figure 10 was generated using communication analyzers from CoTS toolchain [24].

B. Hardware execution platform

The RECS©|BOX server is represented by the following data:

- $platform\_name$: label of the server instance.
- $number\_of\_pe$: represents the number of Processing Elements in the server. It is equal to 11 CPUs (8 ARM + 1 being a Xilinx Zynq, 1 Intel + 1 hosting the GPU).
- id : represents the identifier of each Processing Element PE_k.
- $type$: type of each Processing Element PE_k (CPU, CPU/GPU or CPU/FPGA).

$$(MIQCP-LS)\begin{cases} Min(Z_1)=\sum_{i=1}^{n}\sum_{k=1}^{m} energ_{ik}X_{i,k} \\ Min(Z_2)=C_{max} \\ \sum_k X_{i,k}=1, \forall i \\ \sum_k consum_{t_i,r_k}X_{i,k}\le C_{PE_k,r}, \forall i, \forall r \\ St_i+\sum_k exec_{ik}X_{i,k}\le C_{max}, \forall i \\ St_i+\sum_k exec_{ik}X_{i,k}+\sum_k\sum_l Com_{ik,jl}X_{i,k}X_{j,l}\le St_j, \forall i\in prec(j) \\ St_i+exec_{ik}-St_j\le B(1-X_{i,k}X_{j,k}) \\ \vee St_j+exec_{jk}-St_i\le B(1-X_{i,k}X_{j,k})\forall i,j,k \\ X_{ik}\in\{1,0\}, St_i\in\mathbb{R}, B\ const \end{cases}$$

Figure 8. Modified Mixed Integer Quadratic Constrained Program for using LocalSolver software.

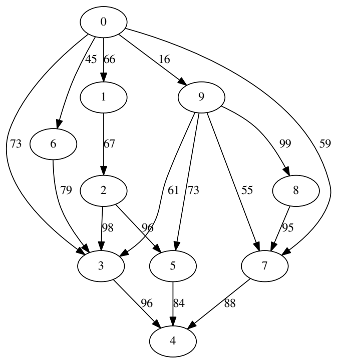

Figure 9. Example of a generated application DAG with 10 tasks.

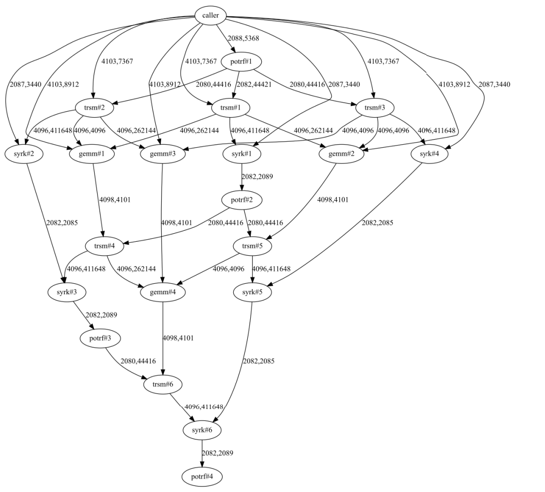

Figure 10. Task graph of a Cholesky Matrix Decomposition application for 256x64, generated using advanced CoTS tools.

- $nb\_ressources$: represents the number of resources inside each Processing Element PE_k (cores, memory, accelerator, etc.).
- nb_cores : represents the number of CPU cores for each Processing Element PE_k.
- $memory\_capacity$: gives the memory capacity for each Processing Element PE_k.
- $accelerator$: indicates if the Processing Element PE_k contains an accelerator (e.g. a CPU with a GPU).
- $memory\_accelerator$: represents the memory capacity of the optionnal accelerator.
- $communication\_between\_pe$: represents the available bandwidth between two Processing Elements PE_k and $PE_{k'}$. Can be updated during runtime if measures are available or calculated using communication modelling. We assume that all Processing Elements can communicate with each other using the network.
- $latency\_between\_pe$: represents the latency between each pair of Processing Elements PE_k and $PE_{k'}$.

In our RECS©|BOX configuration, bandwidth between different Processing Elements is set to $1Gb/s$ whereas a single Processing Element uses its shared memory which is faster ($200Gb/s$ for Intel CPUs, $12,8Gb/s$ for ARM CPUs, $32Gb/s$ for ARM on Zynq FPGA). Latencies matrix was filed with representative values with respect to the number of network switches between PEs. These numbers can be easily updated if the characteristics of execution platform are changed or during runtime thanks to measurements.

C. Exact and heuristic approach implementations

The exact and heuristic strategies presented in Section VI were implemented in $C++$. The exact solution is obtained with $Cplex$ 12.5 running on a 48-core AMD Opteron 6172 server with 64 GB RAM and Debian GNU/Linux 8.2 operating system. The heuristic solution is obtained using $LocalSolver$ 6.0 running on a desktop machine with an Intel Xeon $X5550$ processor, 2.67 GHz, 8 GB RAM. For these experiments, instances varying from 5 to 1000 tasks were randomly generated.

D. Makespan

Table I shows results from implementations using $Cplex$ and $LocalSolver$ for makespan minimization with respect to the whole mapping and scheduling constraints. The first column gives the name and size of the instance (e.g. $test\_5\_11$ refers to an application with 5 tasks running on 11 PEs).

Table I
NUMERICAL RESULTS FOR MAKESPAN MINIMIZATION.

Instances	Makespan Cplex			Makespan LS	
	Solution	Time	Optimal	Solution	Time
test_5_11	237.656	3s	X	237.656	1m40sm
test_8_11	230.5	28s	X	230.5	12m
test_10_11	242.05	57s	X	242.05	13m20s
test_20_11	333.4	21m55s	X	333.4	15m
test_30_11	425.4045	2h46m40s		382.169	1h56m40s
test_40_11	682	4h		747.384	2h30m
test_50_11	1485.0002	2h30		1655.07	1h50m40s
test_60_11	680.19	1h3m20s		654.502	1h6m40s
test_80_11	741.5629	3h		724.975	3h20m
test_1000_11	/	15h		448525	10h
Cholesky_256_64	/	/		47704200000	1h30

Table II
NUMERICAL RESULTS FOR MAKESPAN MINIMIZATION FIRST AND ENERGY MINIMIZATION AS A SECOND OBJECTIVE WITH LOCALSOLVER.

Instances	Makespan−Energy LS		
	Makespan	Energy	Time
test_5_11	237.658	400	8m
test_8_11	236.989	737	12m
test_10_11	242.554	1198	25m
test_20_11	335.139	1473	50m
test_30_11	423.974	2248	2h
test_40_11	747.384	2034	3h
test_50_11	1655.07	361	2h30m
test_60_11	654.502	3457	2h30m
test_80_11	724.975	4510	8h
test_1000_11	448525	46140	15h
Cholesky_256_64	47704200000	18847900000000	2h

The second and third groups of columns (Makespan Cplex and Makespan LS) respectively represent numerical results for $Cplex$ and *LocalSolver* implementations. $Solution$ is the value of the objective function (2), $Time$ shows the CPU time for finding this solution and for $Cplex$, *Optimal* is checked if an optimal solution is found.

This table shows that for bigger instances (> 80 tasks) the exact method is unable to find an optimal solution even after hours of search. The heuristic method is able to find solutions even for big instances at the price of a high CPU time. A more careful examination of the experimental data shows that $Cplex$ yields better results than $LocalSolver$ for very small instances (from 5 to 20 tasks) in term of makespan values and CPU time to reach optimal solution. However, for bigger instances $LocalSolver$ gives good results in term of makespan minimization while being able to give solutions for bigger problems. Thus, one can see that the $LocalSolver$ strategy provides a reasonable value of the makespan minimization. Often it is a bit bigger than one provided by the $Cplex$ strategy in particular for small instances but sometimes it is even a bit lower (for big instances) for example instances with 60 and 80 tasks.

Table II presents results concerning the values of the two function objectives (2) and (1) using $LocalSolver$ and CPU time. In this case, the priority objective is the makespan,

Table III
NUMERICAL RESULTS FOR ENERGY MINIMIZATION.

Instances	Energy Cplex			Energy LS	
	Solution	Time	Optimal	Solution	Time
test_5_11	259	0.08s	X	259	1s
test_8_11	399	0.13s	X	399	1s
test_10_11	512	0.09s	X	512	2s
test_20_11	920	0.41s	X	920	2s
test_30_11	1380	1.39s	X	1380	3s
test_40_11	1895	4.61s	X	1895	3s
test_50_11	308	33.94s	X	309	1m40s
test_60_11	2740	24.56s	X	2740	7s
test_80_11	3641	1m37s	X	3641	10s
test_100_11	4201	3m20s	X	4255	8m20s
test_120_11	5282	5m43s	X	5282	4m33s
test_150_11	6540	22m14s	X	6661	33m20s
test_160_11	7244	44m20s	X	7390	1h20m
test_165_11	8219	51m39s	X	8313	1h40m
test_170_11	8435	1h10m	X	8693	2h
test_180_11	8469	1h38m	X	8827	2h
test_250_11	12311	2h40m	X	12629	2h30m
test_1000_11	/	24h		45826	24h

thus $LocalSolver$ optimizes execution time and do the best for minimizing energy with respect to all mapping and scheduling constraints. We observe that the gap is less than 3% for small instances ($< 40\ tasks$) between solution obtained by $Cplex$ and $LocalSolver$ in bi-objective mode. One can also see that for big instances, it seems that there is only one solution since the values for the makespan are the same that the ones in the previous table.

E. Energy

Table III presents the results from implementations using $Cplex$ and $LocalSolver$ for energy minimization with respect to mapping and scheduling constraints. The labels of the columns follow the same rules as presented in previous subsections. The comparative results presented in Table III show that $LocalSolver$ heuristic performs well in terms of energy value with a reasonable CPU time compared to $Cplex$ strategy. In fact, for instances with less than 40 tasks often the heuristic strategy reaches the optimal solution in a similar CPU time in comparison to exact method and the

Table IV
NUMERICAL RESULTS FOR ENERGY MINIMIZATION USING LOCALSOLVER AND MAKESPAN MINIMIZATION AS A SECOND OBJECTIVE.

Instances	Energy−Makespan LS		
	Energy	Makespan	Time
test_5_11	259	382.041	12m
test_8_11	399	358.096	20m
test_10_11	512	428.207	30m
test_20_11	920	564.126	35m
test_30_11	1380	771.55	1h5m
test_40_11	1895	1734.95	1h30
test_50_11	308	4759.76	1h30m
test_60_11	2740	3664.55	1h40
test_80_11	3641	4487.67	4h
test_100_11	4217	7740.06	4h
test_120_11	5281	22313.3	3h

gap is less than 4% for bigger instances (> 80 tasks).

Table IV presents the values of the two objective functions (2) and (1) using $LocalSolver$ and CPU time. In this case, the priority objective is energy, thus $LocalSolver$ optimizes energy consumption and do the best for minimizing makespan with respect to all mapping and scheduling constraints. The CPU time was too high for bigger instances (> 120 tasks), this explains why no result is given. The local search heuristic performs very well for the multi-objective version. In fact, if we compare the value of objective function (1) it is equal to the optimal value and the second objective (2) is not far compared with results in Table I.

These results reveal that the proposed $LocalSolver$ strategy is more effective for task mapping and scheduling with the goal of minimizing the total energy consumption than $Cplex$ and $LocalSolver$ with the goal of minimizing the total execution time (makespan C_{max}).

These tables show that $LocalSolver$ produces good quality solutions within reasonable computational times for bi-objective optimization. In fact, the objective value for both makespan and energy are not far from the optimal value obtained by $Cplex$ using multi-objective mode for $LocalSolver$. This is an interesting feature in practical situations where efficient algorithm is needed for a trade off solution. On the other hand, this results also confirms that $LocalSolver$ benefits from more computational time. Indeed, long runs lead always to solutions of better quality. The improvement is important for some contexts where the resources should be utilized as optimally as possible. Finally, these numbers are important to confirm the goodness of our mathematical formulation for mapping and scheduling parallel application on fully heterogeneous platform.

VIII. CONCLUSION

Heterogeneous computing systems are widely used today in data centers and high performance computing systems due to increase their performance per watt ratio. However, exploiting theses systems brings new challenges in terms of application management optimization both in research and user community. For tackling these challenges, detailed models of applications and underlying hardware are very important. These models must take into account the main characteristics of these heterogeneous systems, especially the facts that heterogeneity lies both on the computing side and also on the communication side.

We presented in this work different experiments using various applications to understand the energy and performance characteristics of the system. They were performed on an innovative heterogeneous micro-server platform (Christmann RECS©|BOX). Then, we formulated a detailed mathematical model for mapping and scheduling a application represented by a DAG with non independent tasks on a fully heterogeneous platform. This gave us a bi-objectives mixed integer quadratic problem that take into account the whole set of constrained related to this problem.

To solve it, we provided an exact method based on *Cplex* solver and a heuristic approach using *LocalSolver* with local search technique. We also conducted several experiments on randomly generated instances and the real-world Cholesky Matrix Decomposition application. Future work will include additional tests on real-life applications and dynamic strategies to manage on-line mapping and scheduling of applications onto fully heterogeneous servers.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No 609757 (FiPS - Developing Hardware and Design Methodologies for Heterogeneous Low Power Field Programmable Servers).

REFERENCES

[1] S. Arman, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, , and W. Lintner, "United States Data Center Energy Usage Report," June 2016.

[2] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," *Computer*, vol. 26, no. 6, pp. 18–27, June 1993.

[3] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra, "PaRSEC: Exploiting Heterogeneity to Enhance Scalability," *Computing in Science & Engineering*, 2013.

[4] R. Griessl, M. Peykanu, J. Hagemeyer, M. Porrmann, S. Krupop, M. vor dem Berge, L. Kosmann, P. Knocke, M. Kierzynka, and A. Oleksiak, "Fpga-accelerated heterogeneous hyperscale server architecture for next-generation compute clusters," in *First International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC15), held in conjunction with Supercomputing*, 2015.

[5] FiPS consortium. Fips project website. [Online]. Available: http://fips-project.eu/

[6] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci, "Task assignment in heterogeneous computing systems," *Journal of parallel and Distributed Computing*, vol. 66, no. 1, pp. 32–46, 2006.

[7] K. Taura and A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, 2000, pp. 102–115.

[8] A. Legrand, H. Renard, Y. Robert, and F. Vivien, "Mapping and load-balancing iterative computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 6, pp. 546–558, June 2004.

[9] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, April 2004, pp. 107–.

[10] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '07. New York, NY, USA: ACM, 2007, pp. 280–288. [Online]. Available: http://doi.acm.org/10.1145/1248377.1248423

[11] E. Jeannot, E. Saule, and D. Trystram, *Bi-objective Approximation Scheme for Makespan and Reliability Optimization on Uniform Parallel Machines*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 877–886. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85451-7˙94

[12] B. Abbasi, S. Shadrokh, and J. Arkat, "Bi-objective resource-constrained project scheduling with robustness and makespan criteria," *Applied Mathematics and Computation*, vol. 180, no. 1, pp. 146 – 152, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.amc.2005.11.160

[13] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous mpsocs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, p. 14, 2015.

[14] L. Benini, D. Bertozzi, A. Guerri, and M. Milano, "Allocation, scheduling and voltage scaling on energy aware mpsocs," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2006, pp. 44–58.

[15] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environment," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, May 2013, pp. 19–30.

[16] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *Journal of Systems and Software*, vol. 84, no. 6, pp. 985–992, 2011.

[17] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1997.(HCW'97) Proceedings., Sixth*. IEEE, 1997, pp. 135–146.

[18] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*. IEEE, 1999, pp. 3–14.

[19] H. J. Siegel and S. Ali, "Techniques for mapping tasks to machines in heterogeneous computing systems," *Journal of Systems Architecture*, vol. 46, no. 8, pp. 627–639, 2000.

[20] A. Kalinov and S. Klimov, "Optimal mapping of a parallel application processes onto heterogeneous platform," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 123b–123b.

[21] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, vol. 3, no. 3, pp. 195–208, 2000.

[22] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[23] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "Dague: A generic distributed dag engine for high performance computing," *Parallel Computing*, vol. 38, no. 1, pp. 37–51, 2012.

[24] Y. Lhuillier, J. M. Philippe, A. Guerre, M. Kierzynka, and A. Oleksia, "Parallel Architecture Benchmarking: From Embedded Computing to HPC, a FiPS Project Perspective," in *2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, Aug 2014, pp. 154–161.

[25] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems (NIPS), 2012 Conference on*, 2012.

[26] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *Software Engineering, IEEE Transactions on*, no. 1, pp. 85–93, 1977.

[27] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," *Software Engineering, IEEE Transactions on*, no. 6, pp. 583–589, 1981.

[28] A. Billionnet, "Allocating tree structured programs in a distributed system with uniform communication costs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 445–448, 1994.

[29] CPLEX, IBM ILOG, *V12. 1: User´ s Manual for CPLEX*.

[30] Innovation 24. Localsolver. [Online]. Available: http://www.localsolver.com/