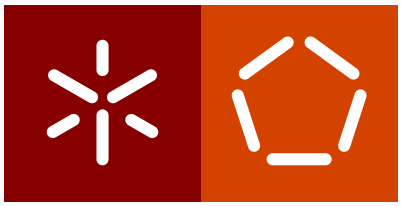
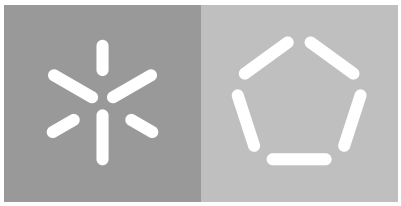


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Joao Paulo Fontoura Moutinho Magalhaes

Power aware scheduler for heterogeneous environments

October 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Joao Paulo Fontoura Moutinho Magalhaes

Power aware scheduler for heterogeneous environments

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Alberto Jose G. de C. Proenca
Joao Garcia Barbosa

October 2017

ACKNOWLEDGEMENTS

First and foremost I would like to thank my mother, the person that has always been there for me and without whom none of this would be possible.
I want to thank both my advisors Alberto Proença and João Barbosa for all the help they gave me and for the opportunities they have provided.
I also want to thank my family for supporting me when times were hardest, especially my sister for putting up with me all of this time.
Last but not least I would like to thank my friends for all the good times and especially thank Célia Ferreira, John Maia and José Luis Silva for helping me on some parts of this thesis.

ABSTRACT

Efficient or green computing is becoming a key issue in current programming techniques, going beyond high performance computing, by simultaneously considering issues such as energy or power consumption. In heterogeneous environments, where different processors and accelerators co-processors may coexist, there is a real opportunity to reduce the overall energy consumption of the system by using scheduling decisions in run-time that can have a good and quick response to changes in the different components.

Current tools to aid the development of efficient applications lack yet these run-time facilities. This motivated the development of a new framework with a power-aware scheduler for heterogeneous environments, PASH-Frame, whose prototype is the key object of this dissertation. This work extended previous performance-based scheduling work to include run-time power-aware features, adding tools to measure power consumption at each device and using different scheduling decisions to get the best outcome according to pre-defined targets by the end user.

To evaluate the overall behaviour of PASH-Frame, several tests were performed: 1000 SAXPY tasks with vector sizes varying from 16 thousand elements to 256 thousand; 200 SGEMM tasks with matrices varying from 64 thousand elements to 16 million and finally a test that combines the two previous ones. Results show that the scheduling algorithm implemented in the framework can achieve good results in some cases, in spite of not being able to make some critical decisions when it comes to energy consumption reduction like forcing a component to idle to save energy.

RESUMO

Computação eficiente (ou green computing) está a tornar-se um dos maiores desafios nas técnicas de programação actuais, considerando simultâneamente os problemas de computação de alta performance bem como a energia e o consumo total. Em ambientes heterogéneos, onde diferentes processadores e aceleradores como co-processadores podem coexistir, existe uma grande oportunidade para reduzir o consumo energético global do sistema ao utilizar decisões de escalonamento em tempo real que conseguem ter uma boa resposta rápida a mudanças nos diferentes componentes.

As ferramentas actuais para ajudar na programação de aplicações eficientes ainda não têm estas ferramentas de leitura de energia em tempo real. Isto serviu de motivação para criar uma nova framework com um escalonador para sistemas heterogéneos consiente do gasto de energia, a PASH-Frame, em que o seu protótipo vai ser explicado nesta dissertação. Este trabalho é uma continuação de trabalho prévio em escalonamento baseado em alta performance ao incluir ferramentas de medição de energia em tempo real e ao fornecer decisões de escalonamento baseadas nesses valores para ter o melhor desempenho de acordo com as escolhas do seu utilizador.

Para avaliar o comportamento da PASH-Frame, varios testes foram feitos: o primeiro teste foi de 1000 tarefas do algoritmo SAXPY com o tamanho dos vetores a variar entre 16 mil elementos e 256 mil; o segundo teste foi de 200 tarefas do algoritmo SGEMM com os tamanhos das matrizes a variar entre 64 mil elementos e 16 milhões de elementos e por fim o terceiro teste é uma combinação dos dois primeiros. Os resultados obtidos mostram que o algoritmo de escalonamento implementado na framework consegue obter bons resultados em alguns casos, apesar de não conseguir fazer algumas decisões críticas para o escalonamento com vista a reduzir o consumo global do sistema, como forçar um componente a ficar inativo para poupar energia.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 CONTEXT

The usage of heterogeneous architectures is increasing in *High Performance Computing (HPC)*. These heterogeneous architectures have coprocessors such as *Graphical Processing Unit (GPU)*s and the many core Intel Xeon Phi in addition to the multi-core processor to increase the computational power of the system and to specially increase the system performance when running particular tasks. Algorithms with a lot of parallelism such as *Single Instruction, Multiple Data (SIMD)* instructions are specially faster on coprocessors due to the high amount of parallelism in their architecture. Despite having a lot of advantages there are extra challenges that arise when using an heterogeneous architecture. When using coprocessors the memory has to be transferred from the processor's memory to the coprocessor's which takes time and costs extra energy since the bus between the two needs to transfer the data. Since the coprocessors are better or worse than the processor depending on the tasks being ran, the work distribution among the processing components can also influence the performance of an application. For simplicity, all the various processing components (processors, coprocessors and GPUs) will be called devices in this report.

An emerging concern in computer science is green computing. Green computing stands for the study and practice of environmentally friendly techniques to lessen the impacts of the information technology, including the decrease in the usage of hazardous materials in manufacturing, the maximisation of energy efficiency and the reusability and recyclability of dying products and waste. Green computing is particularly critical and effective in two different areas: mobile devices and large systems (data centres, supercomputers, large companies with lots of servers or terminals, ...). On mobile devices, the main issue is the limited energy availability since batteries can not store infinite energy. Large systems are the main energy consumers in the information technology and, due to the huge amount of components, a small improvement on several components can make a huge difference in the energy bill. This concern in energy efficiency is specially noticeable in HPC where two types of challenges exist: who can create the fastest supercomputer and who can create

the most efficient / green supercomputer (the efficiency is measured in *Mega Floating-Point Operations Per Watt (MFLOP/W)*). As of November 2016, three of the ten fastest supercomputers in the world are also in the top ten of the most efficient ones with the fastest supercomputer being in the fourth place in the top ten green supercomputer list. Because of this emerging concern in energy efficiency component manufacturers are trying to ease the measurement of energy consumption by creating measurements based on software instead of needing external devices, which allows the measurements to be made without the need for external devices that would make the energy measurement costly (both in time to read the data and in the amount of measuring devices needed to measure different components).

Since the majority of programmers are neither familiar with code optimisation (parallelization, vectorization, ...) nor the challenges that come with heterogeneous environments (programming for GPUs, workload distribution, ...) the usage of schedulers that help the user solving these problems is increasing. These schedulers overlay the *Operating System (OS)* scheduling to handle the workload distribution and are usually found as part of a framework that eases the user's work by managing the memory of all the work automatically and even being able to optimise the code.

1.2 Goals, Motivation, Contribution

With some experience with schedulers (creating a simulator that can test scheduling functions and a scheduling function for balancing different task types in an heterogeneous environment), there is a great motivation for me to be a part of this project.
The goals of this project are the following:

- create samplers to measure the energy consumption of the different devices connected to the system without the need for external equipment;
- explore ways to use the energy measurements obtained from the samplers to schedule the work among the devices connected to the system;
- create a framework that enables the use of the samplers as well as the scheduler.

The main contribution of this work was the development of a framework prototype to aid the development of energy efficient applications on heterogeneous environments, by providing samplers to measure the energy consumption of each device in a compute server and containing alternative scheduler techniques that consider both the performance and energy consumption of each device in the overall solution)

1.3 OVERVIEW

Section 2 explains the different problems of performance and power-centered schedulers on heterogeneous environments. Some of the schedulers currently used are revised, and compared between each other. Section 3 presents the PASH-Frame framework. Section 4 presents the tests done to the PASH-Frame framework and the results. Section 5 concludes this document and present suggestions for future work.

2

SCHEDULING

A scheduler is a method to assign work to computational resources based on the characteristics of the work itself and on its behaviour when being ran in each of the computational resources. Schedulers are used in a variety of ways: some can be used try to balance the workload given to each resource so as to minimise the idle time of the resources while other are used to allow multiple users to share access to system resources keeping the best *Quality of Service (QoS)* possible.

Schedulers can be placed in two extreme positions of the software: as a part of the operating system or embedded in the application. There is also a solution in between: as an *Application Programming Interface (API)* that schedules independent tasks across the available resources, usually available within a development framework. This work follows this mid approach proposing and developing a prototype of a power-aware scheduler for heterogeneous environments framework, the PASH-Frame. This framework is mainly a library of functions to aid the adequate scheduling of task-based programs on heterogeneous environments, aiming the following key goals: maximum performance with minimum energy consumption.

Scheduling of independent tasks favours task-based programming, which uses logical tasks instead of threads to formulate the program with some advantages, such as an easier load balancing among the computational devices and allowing opportunities to apply runtime parallelisation techniques. When using task-based programming with expressions, constructs and a set of conditional statements, a program can be represented as a *Directed Acyclic Graph (DAG)*, where each code block is represented as a task and each dependency between code blocks is represented as a link between tasks.

Figure 1 shows a possible DAG for a simple program where the first tasks ran are A and B with D starting only after B ends and C only able to start after A and B finish. While continuous links represent the data transfer needs of the tasks, the dotted lines represent logical conditions that can not be true at the same time. In Figure 1 tasks E and F are connected to C through a logical condition so either task E or task F is executed, never both.

Since tasks E and F are mutually exclusive they can be assigned to the same time slot on a computational resource.

DAGs like the one presented in Figure 1 can be used not only to represent the work of a big independent task divided into small, and usually dependent, tasks but also to express the relations between a large number of dependent tasks.

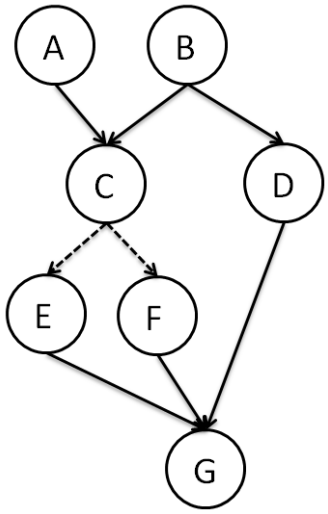


Figure 1.: DAG example

The next section presents some challenges of heterogeneous environments and the following sections cover the main issues related to a scheduling framework: performance-based and power-aware.

2.1 Challenges of scheduling on an heterogeneous environment

When scheduling for heterogeneous environments extra challenges arise. An heterogeneous environment means that coprocessors (like GPUs or the Intel Xeon Phi) are used in combination with the, usually multi-core, *Central Processing Unit (CPU)*. These coprocessors are specially useful in particular tasks, depending on the architecture. In the case of most GPUs and also the Intel Xeon Phi, the parallel oriented architecture makes these devices specially effective in algorithms with a great number of SIMD instructions, which are applied to a range of data and can be ran at the same time. Vectorization can greatly improve the performance of a code by applying the same instructions to a range of different data within a vector instead of just doing a simple scalar operation. In the case of the *Streaming SIMD Extensions (SSE)* instruction set, it is possible to do four operations with single precision floating point values at the same time and with the *Advanced Vector Extensions (AVX)* extensions it is possible to do eight operations with single precision floating point

values at the same time, with a multiplication and addition able to be fused into a new operation called *Fused Mutiply-Add (FMA)*. Vectorization is applied within a single core so a multi-core CPU with eight cores equipped with the AVX extension can be able to do 64 (8 cores * the 8 floating point operations done by each one) operations of the same type at the same time, as long as the memory is able to be loaded at the same time which obligates the programmer to think about the spatial locality (the data needed to be accessed is aligned continuously in memory, as a one-dimensional array for example) of the data. With the use of vectorization and coprocessors with a lot of processing elements the speedup of a code with a lot of SIMD instructions can be huge.

Despite the advantages of using heterogeneous environments there are challenges that appear which would not be a problem in an homogeneous environment. Since the co-processors are separated from the processor, their memory is also different from the main memory in the processor. To be able to transfer memory from the processor to the coprocessor (or the other way around) a BUS between the two has to be used. Since coprocessors have a huge performance on some types of tasks the BUS connecting the coprocessor to the processor may not be able to handle the transfer of all the data needed to keep the coprocessor running at maximum performance which causes a bottleneck. The memory transfer overhead can be masked by doing asynchronous communication, i.e. by decoupling the execution with the data transfer and transferring the data from the next task while running the current one.

Since the performance and the general behaviour of the coprocessors is different from the processor and depends on the type of task being computed, the workload balancing between various devices within the same system is critical to the overall performance. The problem of load balancing can not be easily resolved since it depends on the properties of both the tasks being ran and the devices running them.

2.2 Scheduling oriented to performance

When trying to obtain the best performance possible on heterogeneous systems, there are two types of scheduling that are equally important to the overall throughput of the system: inter and intra-device scheduling where the first refers to the scheduling of tasks between the devices available in the system and the latter refers to the scheduling of tasks within the same device.

Traditionally, the intra-device scheduling was done by the task programmer which needed to handle the load balancing between computational resources within the same device (for example handling the code each core runs within the same multi-core CPU) as well as guarantee that the data could be accessed in a way that vectorization was possible (by using one dimensional arrays with sequential accesses for example) and that the cache of the computational resources was coherent throughout the device. APIs like OpenMP (OpenMP (2016)) can help the programmer parallelise the code by automatically distributing the work among the computational resources available within the device. Libraries like the Intel *Math Kernet Library (MKL)* and the NVidia cuBLAS have highly optimised implementations of some standard tasks like GEMM or AXPY for various types of data like single or double precision floating point numbers for each of the device types (cuBLAS for CUDA capable GPUs and the MKL library for the multi and many core Intel processors and coprocessors).

This dissertation is an extension of the work of Ribeiro et al. (2015), who developed a performance-oriented framework to handle both regular and irregular applications on heterogeneous systems. The framework presented in Ribeiro et al. (2015) has an inter-device scheduler based on a demand driven approach where each device is able to request work from the global task queue (the same for every task ready to be computed).

The framework proposed in Ribeiro et al. (2015) uses the *Heterogeneous Earliest Finish Time (HEFT)* technique, similar to the one used on StarPU. The data management system of the framework uses a MSI cache coherence protocol, also similar to the one used in StarPU, enabling data replication and ensuring consistency among replicas which is combined with a lazy data transfer policy to reduce the data movement overhead. It also supports data pre-fetching and the overlapping of asynchronous data transfers with computation. The main difference between the framework presented in Ribeiro et al. (2015) and the StarPU framework is that the one presented in Ribeiro et al. (2015) has a different performance model and a mechanism to partition the work in run-time.

The framework developed by Ribeiro et al. (2015) can handle two types of kernels (tasks): consumer and consumer-producer tasks. The first type represents a full task and is assigned as a whole to a device. The parallelisation and optimisations made to the task itself are the responsibility of the task's programmer. The second type of task represents the *Basic Work Unit (BWU)* of a task, i.e the operation that is done to a single piece of data during the task's execution. This task is independent from all the BWU tasks (this independence must be assured by the task's programmer) so it can also be assigned independently from the other BWU tasks of the same task so they can be ran together as a bundle on a device that supports vectorization. These tasks spawn new ones that perform the same BWU on

a different set of data. These new tasks are enqueued into the local outbox queue of the device and then enqueued into it's local inbox queue if there is still room or into the global inbox queue if there is not enough space in the local inbox queue.

2.2.1 *A simple simulator for load balancing*

To test the scheduling model and the scheduling decisions a simple simulator was developed in in a previous team work internship at University of Texas at Austin (co-author: John Maia), based on the work of Ribeiro et al. (2015). This tool is based on the consumer tasks explained in section 2.2 and can simulate a set of pre-defined synthetic tasks (with a sleep timer which is bigger or smaller depending on the attributes chosen by the user) running in a set of pre-defined computing devices, the workers.

The attributes for the two main entities (the tasks and the workers) of the simulation are introduced by the user in the source code. Attributes that pre-define each task behaviour include its weight, number of loads/stores, cost of branching and cache misses. Attributes that pre-define each worker behaviour include its latency, throughput, capacity, time to transfer tasks and costs of load/store/instruction executions. The outcomes of each simulation run include information about the workload balancing between workers, the behaviour of each worker during the run and the total time taken to complete all of the work.

This simulator works in iterations. An iteration starts when the scheduler assigns work to the workers and ends when all the workers complete the tasks assigned to them. In each iteration the scheduler assigns a task type to each worker based on what task type best fits the objective function of the scheduler, which in this case is the maximum performance possible. The load balancing step is different from the task type decision step in this simulator. Since the behaviour of each task type is different from worker to worker, the load balancing has to take into account which worker is running which type of task, this is called a combination (explained in section 3.1).

The simulator starts with a small block of tasks and distributes them between the workers. When the combination is ran again, the information about how much time each worker waited for the others to finish their work on the last iteration is used to redistribute the block of tasks between the workers. When the distribution of the block of tasks between the workers is so stable that no worker waits more than a defined percentage of time and that stability endures through a number of iterations, the size of the block of tasks is dou-

bled maintaining the workload percentages of each worker.

This simulator can be modified to support accurate simulation of the behaviour of different tasks and workers in terms of both time and power consumption. This would require the addition of a technique to measure the power consumption of each type of instruction on each type of worker, similar to the ones presented in Hao et al. (2013) and Brabrand et al. (2012).

To measure the power consumption of the tasks in each worker, the simulator would need (i) to estimate the power consumption of each instruction type on each worker, (ii) to compute the number of instructions of each type on a given task and (iii) to estimate the idle consumption of the workers. Only then can the simulator be used to simulate tasks on that specific system.

Since these required power-aware modifications outweigh the simulator usefulness, it was decided not to use the simulator, opting by developing a framework that can use the power samplers created (see section 2.3).

2.3 Measuring power consumption in devices

Before the introduction of software power counter by device manufacturers, the energy measurement of a single devices or an entire systems would require external measuring equipment connected to the device or to the power supply. In cases where there is the need to measure N devices inside the same system, there would be the need to have at least N external measuring devices which is not only expensive but completely impossible when there are hundreds or maybe thousands of systems to be measured at the same time.

Before these power counters were introduced, design-time techniques for evaluation of time and power consumption were used, specially on *Software Product Lines (SPL)* applications since most of the code is the same for two different applications within the same SPL. Hao et al. (2013) and Brabrand et al. (2012) use these kind of techniques to estimate the power consumption of a code by measuring the power consumption of each basic instruction and data transfer and estimating the total power consumption of a given code accordingly to the number of instructions of each type and the data of those instructions. SPLs are specially good test cases for this type of estimations since, by testing code for a given application, the probability of another application within the same SPL having a large amount of equal code is very high. Accordingly to tests made in Hao et al. (2013), the estimation of the energy can be within 10% of the actual measured energy consumption,

tests made on a set of mobile applications of the Google Play store.

With the increase of concern regarding power consumption device manufacturers started to add software energy measuring tools. These tools can be found in most devices created now-a-days and vary depending on the manufacturer, the device type and even in the product's family.

Intel multicore devices

In case of Intel processors, from the Sandy Bridge generation on, there is an interface that can calculate the energy consumption, by providing a software power model that uses hardware performance counters and I/O models, called *Running Average Power Limit (RAPL)*. RAPL provides three different interfaces to access the accumulated consumption of the processor, in Joules. The RAPL interface used for the power readers was the *Machine-Specific Registry (MSR)* counters (available on Linux under */dev/cpu/*/msr*) since is the one with the least overhead. This counter can overflow, which is a problem that the reader has to handle. Because of the differences in the architecture, different generation processors have different overflow values. In spite of the low overhead the MSR counters must be accessed with root privileges.

Intel many-core devices

In case of the Intel Xeon Phi power counters there are two different ways they can be accessed: using the CPU or the coprocessor itself. The information about the Xeon Phi consumption is measured using in the CPU by using the command *"micsmc -f"* which returns the current power consumption of the device, in Watts. Measurements through the device itself were not done but the values are available under *"/sys/class/micras/power"*.

NVidia GPU devices

In NVIDIA GPUs there are two interfaces that can be used to measure the power consumption of the card. One of them is *CUDA Profiling Tools Interface (CUPTI)* that works based on events chosen by the user to return system counters which can sound great but has a terrible measurement rate (in tests made every measurement needed at least one second difference from another). The other interface is *Nvidia Management Library (NVML)* which is part of the NVIDIA *Software Development Kit (SDK)* and does not have a minimum time between measurements. Since the NVML's interface level is lower level than CUPTI's, measurements made using NVML have a much lower overhead.

With the use of these power readers, a library was developed to enable easy access to the time and energy cost of each piece of code. Using the developed library, the user only has to start the device's power reader before running the code and stop it after finishing the code. The reader returns information like average power consumption, minimum and maximum readings, and total power consumed and can also be reset with a simple function. It is possible to save the readings into a file, which is useful for analysing both performance and power consumption hotspots.

It is possible to use the power readers to help schedulers make decisions that decrease the overall consumption of the computation. Power readers can only measure the energy consumption of the device so, since the computational resources within the device can not be measured separately, the power readers are only useful for inter-device scheduling and are not fine-grained enough to be used on in intra-device scheduling yet.

2.4 POWER-AWARE SCHEDULING

Power-centered scheduling focuses on minimising the energy consumption of the system when running the tasks. In addition to the scheduling goals of performance oriented schedulers, the power-centered schedulers also have to take into account the energy consumption of each task when running on each device and the energy consumption of transferring data from one computational resource to another. Some techniques that would rarely be used on a performance-centered scheduling become the go-to tools for the power-centered schedulers. The *Dynamic Voltage and Frequency Scaling (DVFS)* technique, explained in Le Sueur and Heiser (2010), consists of the frequency of a given component, changing the supply voltage. On one hand when increasing the frequency of a given component the tasks being ran there will complete faster but more energy will be used. On the other hand reducing the frequency reduces the energy consumption but increases the time needed to complete the tasks. This technique is particularly useful when dealing with memory-bound algorithms.

Scheduling tasks onto computational resources when obeying a set of constraints defined by the user can be modelled as a *Integral Linear Programming (ILP)* problem, usually NP-hard as mentioned in Gruian and Kuchcinski (1999) and Garey and Johnson (1990). Since solving an NP-hard problem in run-time to decide the next set of tasks to be executed is not feasible, most schedulers use the design-time to obtain information about the behaviour of the tasks on each computational resource and, in cases where the DVFS technique is being used, the behaviour of the tasks on each of the various frequencies available on each device. Schedulers that do most of the computations and predictions during the design-time usually create a Pareto set of possible solutions (the set of solutions that have the best usage of

the resources available and that can not improve on a given criteria without deteriorating the others) and the run-time part selects the solution within the Pareto set that best fits the scheduler's objective function accordingly to the existent real-time constraints (maximum energy consumption allowed or maximum time until the task has to finish).

By surveying the schedulers present in Wang et al. (2012) and Singh et al. (2013) a set of 50 scheduling techniques were analysed. After analysing the 50 scheduling techniques, 5 techniques were chosen to illustrate the current state of the art in power-aware scheduling techniques. The full list of techniques and their respective papers can be consulted in annex A. The 5 chosen techniques (and their respective published papers) are the following:

- Xie and Wolf (2001) use DAGs to separate the tasks' computations from the communications. First, mutual exclusions between computations or between communications are identified and assigned to the same time slot. After the tasks are analysed a value called "static urgency" is assigned to each task and corresponds to the distance between the task and the final task of the DAG, i.e. tasks that are the closest to the first task in the DAG are the most urgent and need to be ran first, and is computed in design-time. This algorithm considers the transfer cost between two tasks within the same device is 0 so it tries to assign blocks of tasks to a single device when possible to save time and energy in communication. In run-time a new value called "dynamic urgency" is given to each task and device pair based on the "static urgency" value of the task as well as the earliest start possible of the task on the device and the *Worst-Case Execution Time (WCET)* of the task on the device. During this scheduling, the task-device pairs with the highest "dynamic urgency" are scheduled first. Since this algorithm is designed for repeating tasks, there is a time deadline restriction that must be obeyed and that restriction is the period of the application (the time until the next iteration of the application starts).

- Gruian and Kuchcinski (1999) uses the design-time to measure the time consumption of each task in each processor and also the average power consumption of each processor and bus. Using DAGs, the communications and computations are separated into different tasks. Using only the branch-and-bound algorithm the optimal solution could be found but it would take a very long time so a credit search algorithm explained in Beldiceanu et al. (1997) is used to find a good solution within a limited time. By using the processor's average consumption and the time a given task needs to run on that processor the consumption of that task on the processor is calculated. Similarly the consumption of the bus while transferring data between tasks is calculated by multiplying on the average consumption of the bus and the time needed for the transfer. Three scheduling algorithms are presented in this paper: the first algorithm only focuses on minimising the consumption of the processors, the second one

focuses on minimising the communications consumption by trying to assign tasks with communication to the same processor and, finally, the third algorithm merges the two previous algorithms and tries to minimise the overall consumption. Results show that the third algorithm was the one with the best results and that the second algorithm was the worst overall and only performed better than the first one in test cases with a lot of communication.

- Chiesi et al. (2015) focus on managing the energy consumption of computationally intense tasks on heterogeneous environments. Each new task type added is analysed to find the power consumption of the task type on the devices in design-time. By using 16 current sensors they were able to measure the power consumption of the entire system (motherboard, *Random Access Memory (RAM)*, CPU, *Hard Disk (HD)* and the two GPUs connected) which is great to measure the consumption of each of the system's components but would be unfeasible in large HPC systems since 16 sensors per node can easily be very expensive. When the task types are analysed, the consumption per second is calculated (in Watts). The scheduling algorithm proposed in this paper uses a peak power performance in watts (defined by the user or calculated by the scheduler) to limit the energy consumption of each node. While there are still tasks in the queue to be ran, the scheduler tries to find a node that can run the task without exceeding the peak consumption. In the case where two or mode nodes can run the task without exceeding the peak consumption the task is assigned to the node closest to its power peak, which leaves the other nodes available to run tasks that require more power consumption. In this paper two ways of organising the task scheduling order are presented. The first is a simple *First In First Out (FIFO)* algorithm, i.e. if the task cannot be executed without violating the peak consumption constraints, the algorithm waits until a node is available. The second algorithm is a *Back fill (BFF)* algorithm that tries to assign tasks that are later in the queue when the first task violates the peak consumption constraints as long as the scheduling of the first task in the queue is not changed by assigning the latter tasks. The scheduling technique presented in the paper can be very useful in HPC systems since the algorithm can both be applied to the nodes and the racks.

- Ykman-Couvreur et al. (2011) implements a two phase algorithm. First, each task is analysed with a tool called HLSim, a fast *single-Design Space Exploration (DSE)* tool capable of deriving a large set of application (task) configurations (which device to run on, level of paralellization, frequency of the processor, ...), each with its own time and energy consumption prediction. Next, and still in the design-time, a second tool called TLMsim (which is a SystemC-based cycle-accurate *Transaction-Level Model (TLM)*) is used only on the Pareto set of configurations outputted by HLSim. TLMsim is used after HLSim and only applied to a specific set of solutions (the Pareto

set) because, in spite of being able to have much more accurate power and time estimations, it takes a lot longer to complete compared to HLMsim. In run-time, and after all the tasks' configurations are tested and evaluated accordingly to their time and energy consumption, a *Multi-dimension Multiple-choice Knapsack Problem (MMKP)* solver is used to rapidly find the best possible scheduling solution accordingly to the deadlines of each task and of the complete system. When no solution is feasible with the current deadlines, the deadline of the task with the lowest priority is relaxed and the MMKP solver is used again.

- Yang et al. (2002) use both design-time and run-time to evaluate and schedule the tasks. In design-time the first step is to transform the various tasks into a single DAG. After the DAG representing the tasks is complete, concurrency improvement transformations (identification and organisation of tasks accordingly to the mutual exclusion of computation and communication) are done. After the concurrency improving transformations, the several scheduling possibilities are evaluated accordingly to the time and energy used by the resources. Since there is a huge number of possibilities for scheduling, only the Pareto set of possibilities is outputted. In run-time the scheduler only has to analyse the different environment and user constraints and choose a scheduling solution that best fits the desired solution (less time, less consumption, ...). This separation of computation in design-time and run-time is really useful because the run-time part of the scheduler can find near-optimal solutions without requiring too much computation time.

The StarPU framework has also a power-aware scheduler, also based on the HEFT method, that takes into account the energy consumption estimation for the task on each device which is defined by the user along with the consumption of the device while idle. The optimisation function of the scheduler now takes into account the task consumption as well as the execution time and the memory transfers.

2.5 Performance vs. Power-aware: a SWOT analysis

The *Strengths, Weaknesses, Opportunities and Threats (SWOT)* analysis is a method that allows the identification of internal and external factors that are favourable and unfavourable to the project being analysed. In this analysis there are four different ways that the properties of a project can be identified: (i) the strengths are the characteristics of the project that give it advantages over others, (ii) the weaknesses are the disadvantages that the project has compared to others, (iii) the opportunities are elements of the environment that can be exploited to the advantage of the project and finally (iv) the threats that could cause trouble for the project. This analysis method is going to be used to evaluate the schedulers

presented above.

The performance framework presented in Ribeiro et al. (2015) has a clear weakness against the others since it does not take into account the energy consumption of the tasks when making inter-device scheduling. The strengths of this algorithm is that it handles irregular tasks well and can perform intra-device scheduling efficiently. There is a clear opportunity to improve the inter-device scheduling by adding the power readers and altering it to also be aware of the energy consumption of each task.

The StarPU framework has the strengths of being able to take energy consumption into account when scheduling the tasks and it has also a good inter-device scheduling technique that is able to handle regular tasks well. The weaknesses of this framework are the need for the user to manually define the consumption of the tasks on each device and the worse performance when scheduling irregular tasks compared to the framework created by Ribeiro et al. (2015). There is also a clear opportunity for this framework to implement the power readers so that the information about the task consumption would be obtained automatically.

The scheduling technique presented in Xie and Wolf (2001) has the overlapping of mutually exclusive communication and computation as a clear strength. It can also schedule the tasks based on the consumption but has a weakness since the measurements of the energy consumption on readings done in design-time.

The scheduling techniques created in Gruian and Kuchcinski (1999) have the strength of allowing the user to choose the scheduling function (minimise consumption of processors, communication or both) based on the characteristics of the tasks. The energy consumption of the tasks is calculated based on the consumption while running the task multiplied by the time it takes for the task to complete, which is obvious less precise than measuring the power consumption in real time making it a weakness to this techniques.

Chiesi et al. (2015) present a scheduling technique that can measure the power consumption in real time and can make scheduling decisions based on the power consumption readings obtained, which is a clear strength against all the others scheduling techniques. The scheduling technique can be applied to both a single node or an entire rack when running on a HPC system. The weakness of this approach is need for hardware power measuring equipment which makes the implementation of the measuring tools on a big number of systems unfeasible.

The schedulers proposed by Ykman-Couvreur et al. (2011) and Yang et al. (2002) have a clear advantage of being able to find a very good solution in real time, without requiring too much computation. The disadvantages of these approaches are the design-time evaluation and computation required to calculate the Pareto set of solutions that are used by the run-time scheduler and the power consumption measurements that are only done in the beginning, also in design-time.

There is a clear advantage for these schedulers to use the power consumption readers created in this dissertation to obtain the real power consumption of the tasks in real time.

3

PASH-FRAME, AN EFFICIENT SCHEDULING TOOL

Since most schedulers currently available do not take into account the energy consumption of the devices connected to the system there is a clear opportunity to create a new scheduler that takes into account the power readers implemented (explained in section 2.3) to make task scheduling decisions.

To test the scheduling technique created, a prototype framework was created. This framework is capable of assigning work to the devices as well as obtain information about the task behaviour on each device (being able to measure time, energy consumption, number of bytes transferred and number of *Floating-Point Operations (FLOP)* operations). This framework does not have any kind memory control, so all the data transfers between devices have to be specified by the task programmer. Since the heterogeneous system where the scheduler will be tested (see section 4.1) does not have DVFS control, the scheduling technique does not take the various frequencies at which the devices can operate into account when scheduling the tasks.

3.1 THE SCHEDULER MODEL

In this scheduling model there are two basic entities: the workers and the tasks. Workers are the devices capable of doing computation and tasks represent the work that needs to be done. Tasks are defined as a set of instructions performed over a range of data. For example a task can be a matrix multiplication of two matrices with 1024^2 elements. Tasks are viewed as black boxes, where the code being run is not accessible by the scheduler so improvements of the code inside a single task need to be done by the task programmer and the scheduler can not divide the work its work. The scheduler works in iterations. An iteration starts when a set of tasks are assigned to each worker and ends when all the workers finish the tasks that were assigned to them. In each iteration the scheduler chooses how much tasks of what type are assigned to each worker (a worker only works on tasks of the same type in an iteration). The group of tasks that are assigned to each worker on each

iteration is called the workload. The goal of the scheduler is to assign the best task type to each worker and find the best workload balance between the workers so that each worker is idle the least time possible.

Initially, the only information that the scheduler has about the tasks that need to be run is the type of task, which needs to be provided by the user. By dividing tasks into different types the scheduler can make assumptions about the behaviour of the next tasks based on the behaviour of previous tasks of the same type. Since the environment where the scheduler works is heterogeneous, the behaviour of the tasks depends not only on the tasks themselves but also on the worker that runs them so the scheduler needs information about the behaviour of tasks on each worker. For that the scheduler has a table called relations table that stores the information about the behaviour of each task type when being ran on each worker. Each cell from the relations table has information about the behaviour of a task type on a given worker. This information contains the time spent running, the consumption, the number of FLOP and the size of the task measured in amount of data handled, all on average per task. The scheduler does not have a memory model, so currently the user has to provide the amount of data that is handled in each task. Each cell of the relations table is updated based on equation(1) where newValue refers to any variable in the relation, the oldValue is the previous value of the same variable in the relation, the readValue is the value obtained by the readers and the update percentage (updatePer) is chosen by the user. Equation(1) is used instead of a general average of all the tasks to give the user the ability to define how much the scheduler takes into account the last run in comparison to the previous ones.

$$newValue = oldValue * (1 - updatePer) + readValue * updatePer \tag{1}$$

With a low update percentage the values read by the reader on each new iteration will not have much influence on the values of the relations' cells so the scheduler will not be able to adapt to sudden changes in the devices. With a high update percentage the scheduler changes its behaviour accordingly to sudden device changes but sometimes this is not desired (for example when scheduling highly irregular tasks the average of all the tasks can be more accurate to future iterations than the information of the last ones). An update percentage of 0 only takes into account the first measurement made, in the tuning stage (see section 3.2), while an update percentage of 100 only takes into account the last measurement made.

The work type assigned to each device in each iteration is decided based on the scheduling decision chosen by the user (see 3.1.1). A sort is done to all the cells of the relations

table that correspond to the same worker, where the task types are ordered according to the best fitness of the decision chosen by the user. After the scheduler chooses which task type each worker will work on each iteration, it chooses a balanced workload for each worker so that all workers finish doing their work at the same time, avoiding the idling of the workers. To be able to have a balanced workload for the workers, the scheduler has to store the information about the previous iterations. To do that a history list was created where each item of the list corresponds to one combination of workers and task types. Each item contains information about the iteration time, the time each worker took to complete their work, the energy consumption (total and per worker), the number of FLOP and the total number of bytes handled. Figure 2 shows a simple example of possible combinations with 2 different workers and 3 task types. Since workers A and B are different workers, combination 1 and combination 2 are not the same, and each one has a different history. When a combination is chosen, the scheduler searches for the information about the last iteration ran with the same combination. If a combination is ran for the first time, a base workload is given to each worker and the load balance is made when the combination runs again.

By using the information about the last iteration that ran with the same combination, the scheduler tries to assign the best workload to each worker so that all the workers finish at the same time. Equation(2) shows how the new workloads are decided based on the results obtained in the previous iteration that had the same combination.

$$newWorkload = oldWorkload + oldWorkload * waitingPercentage \tag{2}$$

The way of balancing the workload between devices was taken from the simulator explained in section 2.2.1.

The scheduler simply assigns more work to each worker proportionally to the time each worker waited in the previous iteration with the same combination. Equation 2 should be specially effective when handling regular tasks since their time and consumption can be easier to estimate. Since sometimes a worker can have very bad performance on all the types of tasks, there was a need to add a way to decrease the workload assigned to a worker from one iteration to another. A condition was added to reduce the workload of a worker by half in the next iteration if the other workers waited more than half the total time of the current iteration. With this condition the scheduler does not have to increase so much the workload of the workers that waited, since the worst worker will have much less tasks to work on.

Workers: A, B
Task Type 1, 2, 3

Possible combination 1: A-1, B-2
Possible combination 2: A-2, B-1
Possible combination 3: A-1, B-1

Figure 2.: Explanation of what combinations are on this scheduler

Measuring consumption with instant consumption readings

The information about the consumption of the Intel Xeon Phi and the NVIDIA GPUs is obtained in Watts instead of Joules, i.e. instead of receiving the accumulated energy like in the Intel processors the value received is the current power. To calculate the energy consumption of these coprocessors, a minimum of two different power readings have to be collected. A simple example can be viewed in Figure 3 where 3 measurements are taken within 2 milliseconds. To calculate the total energy consumption of the device the consumption between every reading is computed. In the example of Figure 3 the consumption between measurements M1 and M2 and M2 and M3 correspond to the total energy consumption. Each of these consumptions can be easily calculated as the sum of the area of a right triangle with the area of a rectangle, which is used as an approximation to the integral of the energy consumption. Equation 3 describes how the total energy consumption between two measurements is calculated.

$$\begin{aligned} ConsumptionM_iM_j = {} & min(M_iconsum, M_jconsum) * (M_jtime - M_itime) \\ & + ((max(M_iconsum, M_jconsum) \\ & - min(M_iconsum, M_jconsum)) \\ & * (M_jtime - M_itime)/2) \end{aligned} \tag{3}$$

The total consumption between the first measurement and measurement N is calculated as shown in Equation 4.

$$totalConsumptioM_1M_n = \sum_{i=1}^{n-1} ConsumptionM_iM_{i+1} \tag{4}$$

For this example the total energy consumption between M1 and M2 would be:

$$\begin{aligned} consumptionM_1M_2 &= min(10,20) * (2-1) + ((max(10,20) - min(10,20)) * (2-1)/2) \\ &= 10 * 1 + ((20-10) * 1/2) \\ &= 10 + 5 \\ &= 15mJ \end{aligned}$$

Similarly, between M2 and M3 the consumption would be:

$$\begin{aligned} consumptionM_2M_3 &= min(20,15) * (3-2) + ((max(20,15) - min(20,15)) * (3-2)/2) \\ &= 15 * 1 + ((20-15) * 1/2) \\ &= 15 + 5 \\ &= 20mJ \end{aligned}$$

So the total consumption in this example would be 35 mJ (15 + 20).

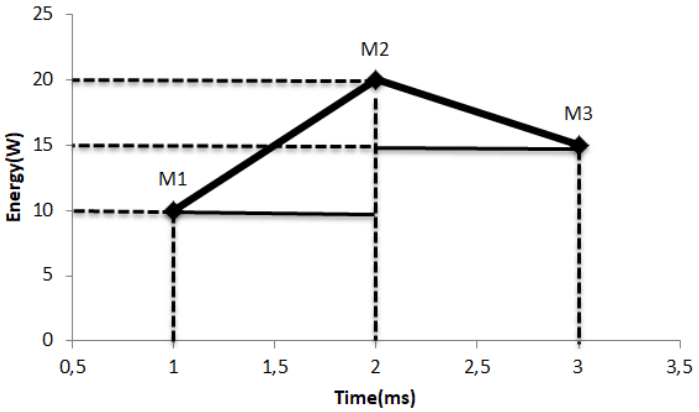


Figure 3.: Consumption calculation with instant consumption readings

3.1.1 *Scheduling decisions*

Since the scheduler can not apply any scheduling technique directed at an individual task, it can only choose which task type is better for each worker, accordingly to the scheduling decision selected by the user. To choose the best task type for each worker, a sort is made to each of the lines in the relations table (that corresponds to each worker) where the compare function is chosen by the user. Currently there are 6 different compare function available:

- Energy consumption per FLOP - focuses on reducing the energy consumption per FLOP operation

- Energy consumption per byte - focuses on reducing the energy consumption per byte handled
- Energy consumption - focuses on reducing the energy consumption of the tasks overall
- Performance per FLOP - focuses on maximising the throughput per FLOP operation
- Performance per byte - focuses on maximising the throughput per byte of memory handled
- Performance - focuses on maximising the throughput per task

	Task type A	**Task type B**
Worker 1 - time	0.5s	1s
Worker 1 - energy	250J	100J
Worker 2 - time	2s	3s
Worker 2 - energy	50J	100J

Table 1.: Relation table values for the scheduling decision combination example

There was an attempt to combine two or more different scheduling decisions by calculating the fitness percentage of each task type for each worker on both scheduling decisions and then combining the percentages accordingly to weights defined by the user. The fitness for each scheduling decision is calculated by the value in the relations table divided by the sum of all the values on the same line of the relations table (corresponding to all the values of the same type (energy, time, energy/FLOP, ...) for a given worker). For example, with two task types A and B and two workers 1 and 2, and with the values of the relations table presented in table 1 the computations of the task type's fitness on worker 1 would be calculated the following way:

- **Case 1 -** the scheduling decisions selected were energy consumption and performance and the weight of both is 50%:

$$\begin{aligned} FitnessPerformanceTypeA &= 0.5/1.5 \\ &= 33.33\% \\ FitnessConsumptionTypeA &= 250/350 \\ &= 71.43\% \end{aligned}$$

$$\begin{aligned} FitnessPerformanceTypeB &= 1/1.5 \\ &= 66.67\% \\ FitnessConsumptionTypeB &= 100/350 \\ &= 28.57\% \end{aligned}$$

Since both time per task and consumption need to be minimised, the task type with the smaller percentage value will be selected:

$$\begin{aligned} OverallFitnessTypeA &= (33.33 * 0.5 + 71.43 * 0.5) \\ &= 52.38\% \\ OverallFitnessTypeB &= (66.67 * 0.5 + 28.57 * 0.5) \\ &= 47.62\% \end{aligned}$$

In this case the scheduler would assign tasks of type B to worker 1 on the current iteration.

- **Case 2 -** the scheduling decisions selected this time were still energy consumption and performance by this time the weight is of 70% to performance and 30% to energy consumption. The fitness of the task types for a single scheduling decision are the same as the example above, with only the combination between them changing. The task type selected will still be the one with the smallest percentage:

$$\begin{aligned} OverallFitnessTypeA &= (33.33 * 0.7 + 71.43 * 0.3) \\ &= 44.76\% \\ OverallFitnessTypeB &= (66.67 * 0.7 + 28.57 * 0.3) \\ &= 55.24\% \end{aligned}$$

This time the worker 1 will be assigned tasks of type A, since it has the smallest percentage.

Unfortunately there was no time to properly test the combination between different scheduling so it is not presented on section 4.2. The proper testing of this algorithm is suggested as future work.

3.2 SCHEDULING TUNING

When the scheduler starts, it collects information about the workers present in the system. The information about the CPU is collected by accessing the */proc/cpuinfo* file on Linux which can tell the number of sockets and the number of cores per socket. The GCC compiler command "*gcc -march=native -Q –help=target—grep march*" is also used to obtain information about the family of the CPU. For information about the Xeon Phi devices, the *micinfo* command is used. Finally for detecting graphic cards the *nvmlUnitGetCount* function, which is part of the NVML library, is called.

Since there is no more information in the beginning and the relations table is empty. A tuning stage is used to populate the information of the relations table and in some entries of the history list. Three different ways of tuning the scheduler were created. All tuners have the same behaviour where a block of tasks of the same type is analysed in each worker on each tuning iteration.

The tuning technique currently available are the following:

- By block - in this tuning technique the user defines how many tasks (in constant number) are used by each worker in each iteration
- By percentage - in this tuning technique the user defines a percentage of tasks that are assigned in each training iteration. The difference between this training and the training by block is that task types with different number of tasks will have different block sizes
- By time - in this tuning technique the user defines a time (in seconds) in which each worker tests as much tasks as it can of the same type. The scheduler assigns tasks to the workers and wait for the workers to finish to assign them more tasks. If the time defined by the user has already passed, the scheduler stops assigning tasks.

3.3 THE SUPPORT FRAMEWORK

Since the simulator presented in Section 2.2.1 is not able to accurately simulate the energy consumption of heterogeneous tasks and workers, a framework was created to test the behaviour of the scheduler when altering the scheduling parameters and to test the potential

of using time and power consumption measurements to schedule the following iterations. This framework was created to be able to fully monitor the time and the consumption of the tasks on each device while having least impact possible on the overall computational time.

Since the framework was created with the goal of testing different tuners and scheduling techniques each of the tuners and scheduling decisions can be easily replaced. Even the scheduling algorithm itself can be easily changed if needed.

The algorithm used for assigning tasks, starting the workers and doing all the measurements is presented in Algorithm 1. The algorithm in the case of the tuners is very similar to Algorithm 1 but there is no need to choose the best task type since all the task types have to be sampled and there is also no need to choose the workload for each worker because it is chosen by the user, unless the user chooses the timed trainer. When the scheduler needs to calculate the base workload for a history combination that was not used before, the algorithm 2 is used. Algorithm 2 does not provide any guarantees about the load balancing, it only calculates the total base workload (the workload for all the workers combined) based on the number of tasks in the queue. As can be seen in Algorithm 1, the only difference in the behaviour of the different scheduling decisions currently available is the ordering of the best task types for each worker at the end of each iteration.

```
Data: workers; queue of tasks; relations; history; scheduling decision
Result: all the tasks for the iteration are assigned and run and the measurements
        about time and energy consumption are made
while there are still tasks in the queue do
    compute the best task type for each worker based on the relations;
    compute the best workload for the workers based on the history;
    assign tasks to the workers;
    start the iteration timer;
    forall workers with work do
        start a thread running each worker's work function;
    end
    wait for all the workers to finish;
    stop the iteration timer;
    forall workers do
        store the information about the last iteration in the relations and history
         structures;
    end
    sort each relation table line accordingly to the scheduling decision selected;
end
```

Algorithm 1: Work assignment and measurement algorithm

Data: the total number of tasks on a single task type queue
Result: the base workload for that task type based on the size of the queue
initiate the variables (i and n, both int);
for $n = o; i \leq queueSize; ++n$ **do**
  i = 10^n;
end
return queueSize / 2^n;
Algorithm 2: Calculation of the base workload for a single task type queue

The function that handles the work of each worker is presented in Algorithm 3. It simply starts the power reader associated with the device and a time counter before the worker starts working. The power reader and time counter are stopped after all the work assigned to the worker this iteration is complete. Similarly when measuring the number of FLOP inside the tasks, a FLOP counter using *Process API (PAPI)* is started and stopped only when all the work is complete.

Data: worker
Result: the worker finishes it's work and stores information about the iteration so it can be handled by the scheduler
start a thread with the power sampler of the worker;
start a timer;
while *there is still work in the worker's queue* **do**
  do the task's work;
end
stop the power sampler's thread and the timer;
store the information about time, power consumption and work done;
Algorithm 3: Work function of the worker

Since there are three different tuning techniques, six different scheduling decisions and other variables an additional structure was created to serve as a control structure for the framework, decreasing the amount of work the user needs to do before being able to run the code on the framework. This control structure detects the devices available in the system (number of CPUs, GPUs and coprocessors, the number of cores available on each device and also the family of the processor used in the system, which is needed for the power reader (the default value for the processor family is Sandy Bridge)) and receives and handles the choices of the user relative to the tuner and scheduling decision selected.

To use this framework, the user must create a new project and link the scheduling or the power sampler libraries. Each of the task types created must be extensions of the task base class available on the scheduler libraries in order to use the scheduler. Since there is no memory control, the user has to define how the data is transferred to the devices. Each task

returns the number of bytes handled to and from the devices which is used to simulate part of a memory control and allows the scheduler to make decisions based on the number of bytes handled. The value of number of bytes handled is provided by the user as the return of the task (this was only done to be able to test the scheduling decisions that need the number of bytes handled as an argument).

4

TEST AND EVALUATION

The key motivation to develop a new framework was the lack of a real-time tools that can take in-time scheduling decisions considering not only the measured devices performance but their energy consumption as well. To evaluate the quality of the framework outcomes several tests were devised and applied, and their results are here presented and discussed.

All tests used two different task types: squared matrix multiplication and the *Single Precision A.X Plus Y (SAXPY)* algorithm. These tasks are both regular, where the amount of data transfers and computations can be found by knowing the size of the matrices and vectors. With regular tasks the workload balancing between devices should be very effective. Both tasks are implemented using MKL for both multicore and many-core Xeon devices and using cuBLAS for the implementation on the NVidia GPUs.

The first set of tests aimed to find out the performance and energy consumption of each device (and all the combinations between them) when running SAXPY tasks. To test the behaviour of the devices when handling SAXPY tasks 5 different array sizes were tested (16k, 32k, 64k, 128k and 256k). For each of the tests, 1000 SAXPY tasks were ran. Two different comparisons were made with the results: in the first one different device combinations (CPU only, CPU and GPU, all workers, ...) were tested for arrays with the same size; in the second the different task sizes were compared for the same worker combination.

The second set of tests, similar to the first, was done to test the behaviour of each device when running matrix multiplication tasks. Once again 5 different sizes were tested with the number of elements varying between 64k, 256k, 1M, 4M and 16M. Since the matrix multiplication tasks have much more operations than the SAXPY tasks, the number of tasks on each test was set to 200. With a small number of tasks, the scheduler does not have many iterations to find the best tasks for each worker and the balance between them so the scheduler needs to be able to quickly find a good balance.

The third set of tests combined the previous two and was done to evaluate the behaviour of the scheduler when having two different task types to choose from when assigning work to each of the devices.

For these tests the scheduling decision used was the power consumption, the tuner selected was the tuner by block with a block of 10 tasks. The update rate chosen was 80 per cent. The rate of readings for each power sampler is 1 millisecond, the lowest possible for all of the device power samplers.

4.1 TESTBED

The testbed where the framework will be tested is one of the nodes available on the Stampede supercomputer which is one of the supercomputers available at *Texas Advanced Computing Center (TACC)*. The Stampede supercomputer is comprised of 6404 nodes of which 4 are login nodes and the other 6400 are computation nodes. Table 2 shows the configuration of each of he 6400 nodes and Table 3 shows the overall system configuration and performance (these tables were obtained from the Stampede User Guide, available in TACC (2017)). As can be seen in Table 2 each compute node has two Intel Xeon E5-2680 on a single board, as a *Symmetric multiprocessing (SMP)* unit with a performance of 346 GFLOPS/node. Additionally each node has a special production Intel Xeon Phi SE10P with a peak performance of 1.0 DP TFLOPS/Phi. Each node contains 32GB of memory, with 4 channels from each processor's memory controller to 4 DDR3 ECC DIMMS, obtaining 51.2GB/s for all four channels in a socket. The memory for each coprocessor is 8GB GDDR5 with 8 dual-channel controllers and a peak performance of 320GB/s.

Component	**Technology**
Sockets per Node/Cores per Socket Coprocessors/Cores	2/8 Xeon E5-2680 2.7GHz (turbo, 3.5) 1/61 Xeon Phi SE10P 1.1GHz
Motherboard	Dell C8220, Intel PQI, C610 Chipset
Memory Per Host Memory Per Coprocessor	32GB 8x4G 4 channels DDR3-1600MHz 8GB GDDR5
Interconnect Processor-Processor Processor-Coprocessor	 QPI 8.0 GT/s PCI-e
PCI Express Processor PCI Express Coprocessor	x40 lanes, Gen 3 x16 lanes, Gen 2 (extended)
250GB Disk	7.5K RPM SATA

Table 2.: Compute node configuration

Component	Technology	Performance/Size
Nodes(sled)	2 8-core Xeon E5 processors, 1 61-core Xeon Phi coprocessor	6400 Nodes
Memory	Distributed, 32GB/node	205TB (Aggregate)
Shared Disk	Lustre 2.1.3, parallel File System	14 PB
Local Disk	SATA (250GB)	1.6PB (Aggregate)
Interconnect	InfiniBand Mellanox Switches/HCAs	FDR 56 GB/s

Table 3.: Configuration of the Stampede system

The framework testing will be done on special computation nodes within the Stampede supercomputer dedicated to visualisation and *General Purpose Graphics Processing Unit (GPGPU)* processing. There are a total of 128 visualisation nodes in the Stampede supercomputer. These nodes are equipped with an NVIDIA K20 GPU with 5GB of on-board GDDR5 memory in addition to the two Xeon E5-2680 and the Xeon Phi SE10P coprocessor. Since the power consumption is one of the main concerns, the *Thermal Design Power (TDP)* of each device as well as the maximum total consumption of the system can be seen in Table 4. The values presented in Table 4 were taken from Intel (2017) for the Intel Xeon E5-2680 processor, CPU-World (2017) for the Intel Xeon Phi SE10P and CNET (2017) for the NVIDIA K20 GPU.

Device	TDP
2x Intel Xeon E5-2680	260W
Intel Xeon Phi SE10P	300W
NVIDIA K20 GPU	225W
Maximum combined energy consumption	785W

Table 4.: Power consumption of each device and combined

Since no power-aware schedulers that work with both the CPU, GPU and Intel Xeon Phi and do not receive any kind of information before runtime were not found, all the tests were made comparing different scheduling decisions, trainers and other variables like the update rate of the values on the relations table.

4.2 Discussion of results

In this section the results of the tests made to the framework are analysed and both the performance of each device when running each task type (with different sizes) and the behaviour of the scheduler are evaluated.

To be able to estimate the energy consumption for the entire system when not all of the devices are being used, the idle power consumption of each device was measured ten times

Device	Idle consumption per second
2x Intel Xeon E5-2680	196,99 J
Intel Xeon Phi SE10P	103,87 J
NVIDIA K20 GPU	39,13 J

Table 5.: Power consumption of each device and combined

and the average was set as the idle consumption of each device. Table 5 shows the results obtained for the idle consumption of each one of the devices.

The first test done was meant to test the performance of all the combination of workers when running SAXPY tasks of different sizes. 7 different worker combinations were tested: CPU, GPU, Xeon Phi, CPU and GPU, CPU and Xeon Phi, Xeon Phi and GPU and finally all the workers combined. The different sizes for the vectors used in the SAXPY task test were the following: 16k, 32k, 64k, 128k and 256k elements.

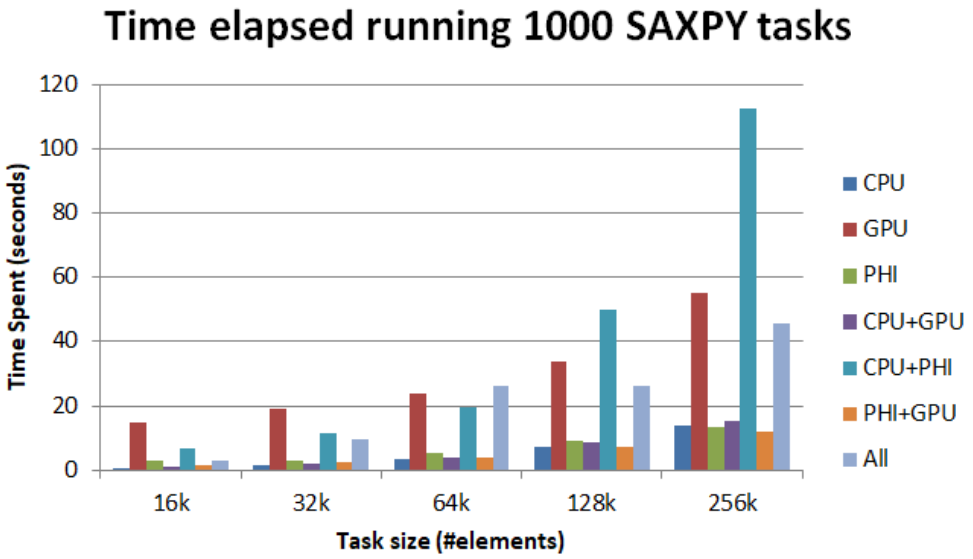


Figure 4.: Total time spent running 1000 SAXPY tasks with different worker combinations and task sizes

Images 4 and 5 show the results for the test in time spent running the tasks and energy consumed, respectively. Clearly the GPU is the worst worker when running SAXPY tasks, which is predictable since the SAXPY tasks do not have enough computations to compensate for the cost of the memory transfers. The CPU is the best worker when running SAXPY tasks alone, except for the case where the vectors have 256k elements and the Xeon Phi has the best performance. In terms of workers combined, the CPU and GPU combination is the best one for task sizes of 16k, 32k and 64k elements while the Xeon Phi and GPU combina-

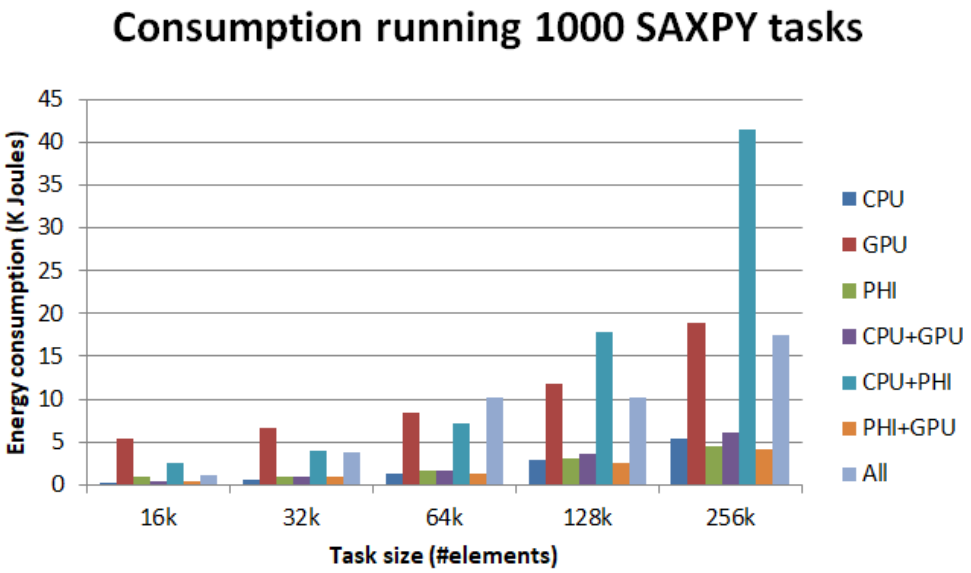


Figure 5.: Energy consumption while running 1000 SAXPY tasks with different worker combinations and task sizes

tion becomes the best one for sizes of 128k and 256k elements.

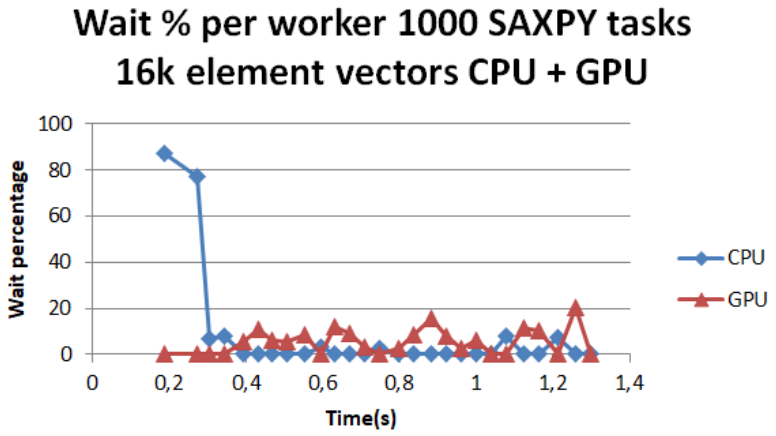


Figure 6.: Wait percentage of CPU-GPU combination for SAXPY tasks with 16k element vectors

Figures 6 and 7 show the wait percentage and the workload distribution (in number of tasks) for the CPU and GPU combination. Each marker in the graph represents a new iteration. The scheduler manages to find the best worker for running the tasks (in this case the CPU) within 3 iterations, by assigning more than 30 tasks to the CPU while giving just 2 tasks to the GPU. In this case the combination between the CPU and the GPU devices is always worse than the combination where only the CPU runs the tasks. This happens

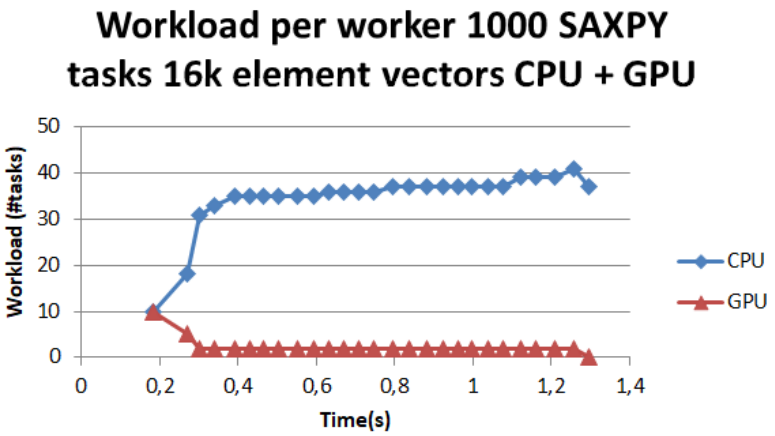


Figure 7.: Workload distribution of CPU-GPU combination for SAXPY tasks with 16k element vectors

because the performance of the GPU decreases the performance of the combination overall. The workload distribution function is not good enough to identify the terrible performance of the GPU when running SAXPY tasks so it keeps giving it new tasks, in spite of being just 2 each iteration.

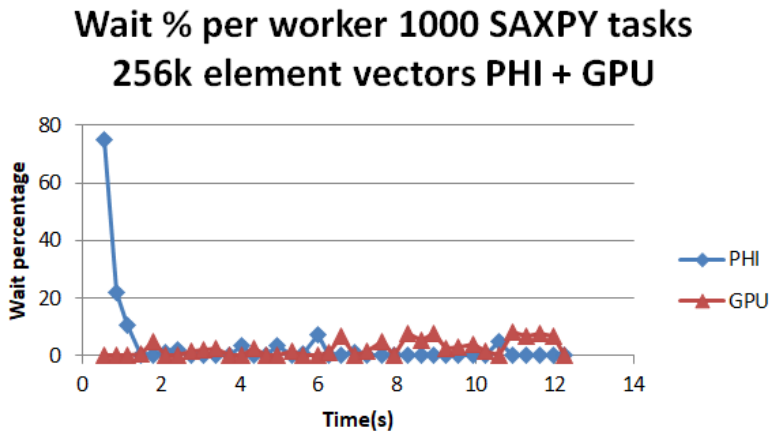


Figure 8.: Wait percentage of PHI-GPU combination for SAXPY tasks with 256k element vectors

Figures 8 and 9 show the wait percentage and workload distribution for the Xeon Phi and GPU combination for SAXPY tasks with 256k element vectors. This combination is the best one for the test of SAXPY tasks with 256k element vectors. Figure 9 shows that the scheduler finds the best number of tasks for the GPU in the second iteration, which is very good. In 4 iterations the scheduler manages to find a stable workload for the Xeon Phi with 22 tasks that take about as much time as the GPU takes to run 5 tasks. Figure 8 shows

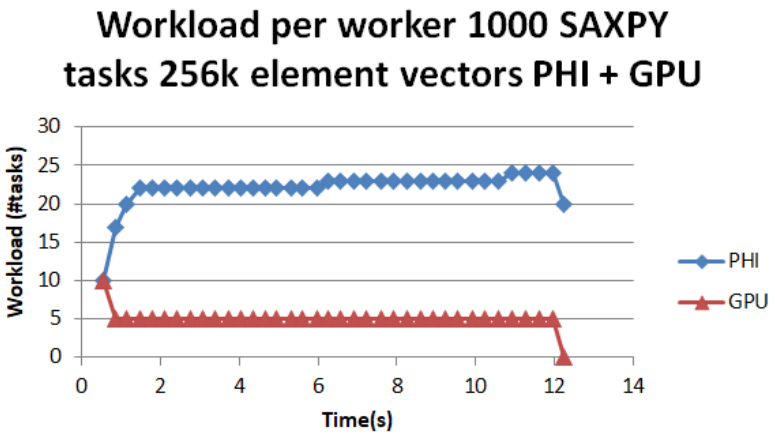


Figure 9.: Workload distribution of PHI-GPU combination for SAXPY tasks with 256k element vectors

that, starting from iteration 4, the wait percentage of the workers is always less than 10% and even in cases where the GPU waits about 10% of the time it is not enough to increase the workload since it only runs 5 tasks per iteration and a wait of 20% would be needed to increase the number of tasks to 6. This combination is faster than when only one worker is running the tasks (in this case the best single worker is the Xeon Phi) because the GPU can run some of the tasks, in spite of just being 5 tasks per iteration, increasing the overall performance.

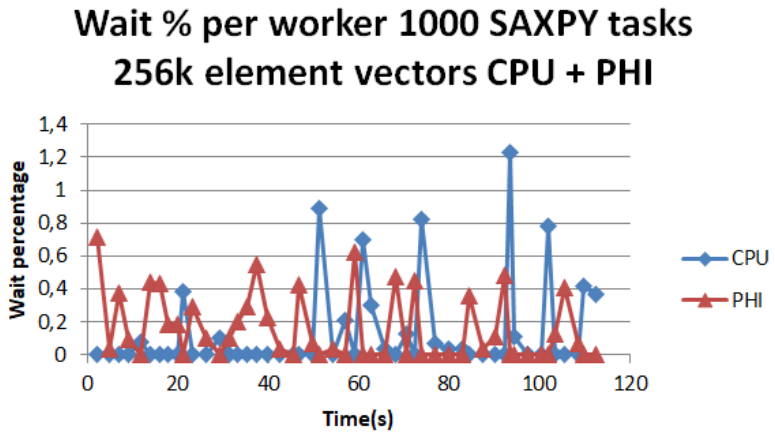


Figure 10.: Wait percentage of CPU-PHI combination for SAXPY tasks with 256k element vectors

The strangest case is the combination of CPU and Xeon Phi where it would expected to be the best since they are clearly the best single workers when running SAXPY tasks. Figures 10 and 11 show the wait percentage and the workload for the CPU and Xeon Phi

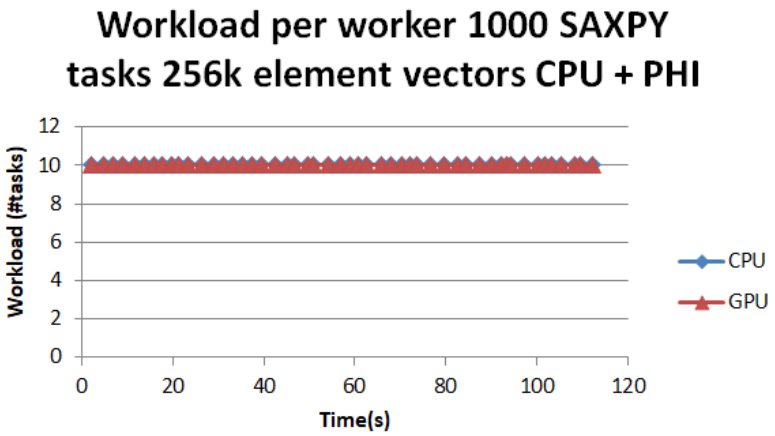


Figure 11.: Workload distribution of CPU-PHI combination for SAXPY tasks with 256k element vectors

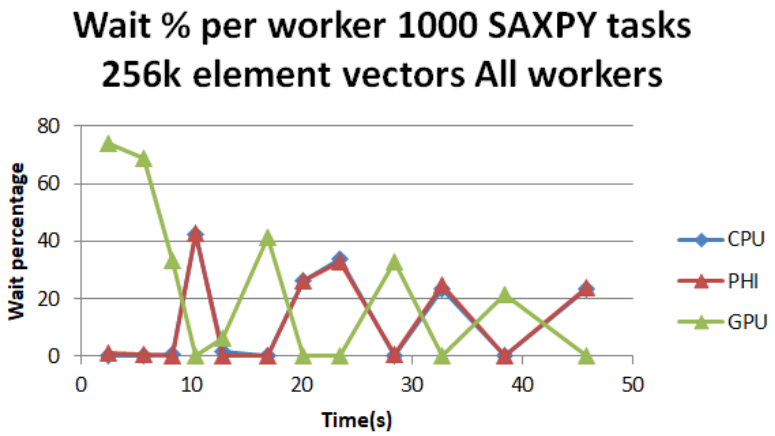


Figure 12.: Wait percentage of All worker combination for SAXPY tasks with 256k element vectors

combination for SAXPY tasks with 256k element vectors. The wait percentage of the workers is extremely low (reaching 1.2% maximum) so the workload never changes. Figures 12 and 13 show the wait percentage and workload for the combination with all workers, respectively. The lines for the CPU are barely visible since it behaves exactly the same way as the Xeon Phi. This low performance when combining the CPU and Xeon Phi device can be explained by the synchronous communication that is made between the Xeon Phi and the CPU, which forces the CPU to wait until the vectors are transferred to the Xeon Phi to continue the computation. Similarly the Xeon Phi has to wait for the CPU to finish the computation to be able to receive the new vectors for the next task.

The second test made was similar to the first one but, instead of running 1000 SAXPY tasks, 200 SGEMM tasks are run. The total number of elements per matrix is 64k, 256k,

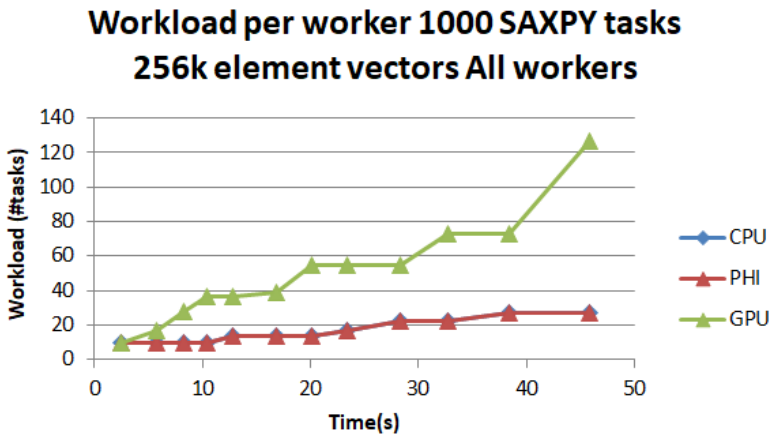


Figure 13.: Workload distribution of All worker combination for SAXPY tasks with 256k element vectors

1M, 4M and 16M. The different combinations used were the same as the ones used in the SAXPY task test.

In this test the GPU is the worst single worker for the task size with 64k element matrices but quickly becomes the best one for bigger sizes. The improvement in the performance of the GPU device is explained by the relation between the computation required for the task and the memory transfer, which is way more than the case of the SAXPY tasks. The biggest difference can be seen on the test with 16M element matrices, where both the time spent and the energy consumed are both really low (do not forget these graphs have a logarithmic Y axis) compared to the other combinations. The next best combination is the Xeon Phi and GPU combination, which is actually the best combination when running SGEMM tasks with 64k and 256k element matrices.

Figures 16 and 17 show the wait percentage and workload for the SGEMM task test with 64k element matrices. In this case the Xeon Phi is much better than the GPU so the scheduler reduces the workload of the GPU to 5 tasks and increases the workload of the Xeon Phi to 16 tasks in iteration 2. In iteration 3 the scheduler increases the number of tasks per iteration to 6 because the GPU waits more than 20% of the total iteration time. After that the work is stable until the end. In figures 18 and 19 the wait percentages and workload for the SGEMM task test with 16M element matrices. In this case the GPU is much better at computing the tasks than the Xeon Phi, unlike the case with 64k matrices. Once again the scheduler only takes 3 iterations to find the best balance between the workload of the devices.

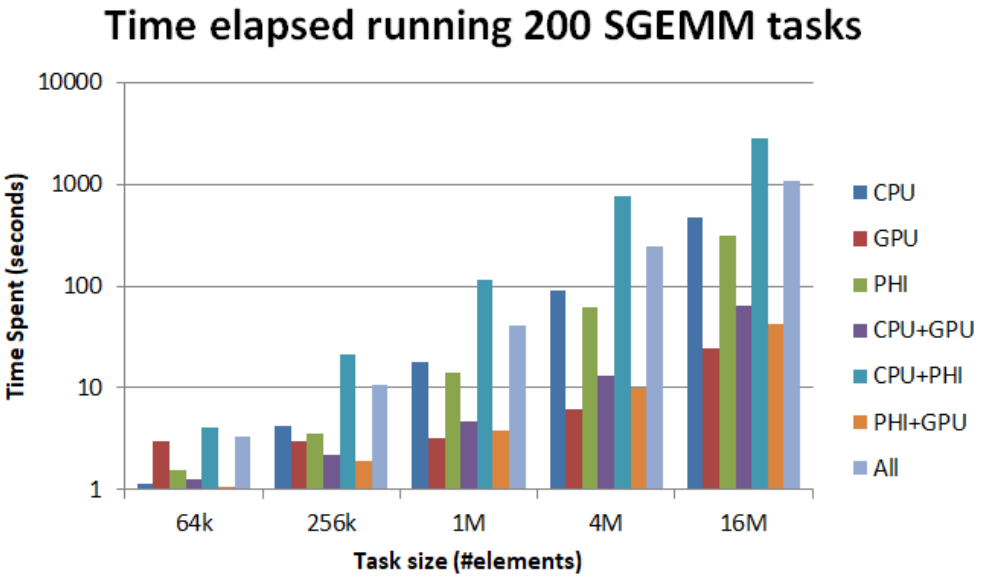


Figure 14.: Total time spent running 200 SGEMM tasks with different worker combinations and task sizes

The final test made combined the two previous tests by running 1000 SAXPY tasks and 200 SGEMM tasks. The worker combinations and the size of the tasks are the same as the ones used on the previous tests.

Figures 20 and 21 show the time elapsed and the energy consumption when running 1000 SAXPY tasks and 200 SGEMM tasks. The results are quite similar to the ones described above, in the second test made, since the time and energy required for running the SGEMM tasks are much higher than the required to run the SAXPY tasks. In all the tests made the CPU and Xeon Phi combination was the worst overall for the reasons explained when analysing the first test made (the fact that the data transfer between the CPU and Xeon Phi is done synchronously).

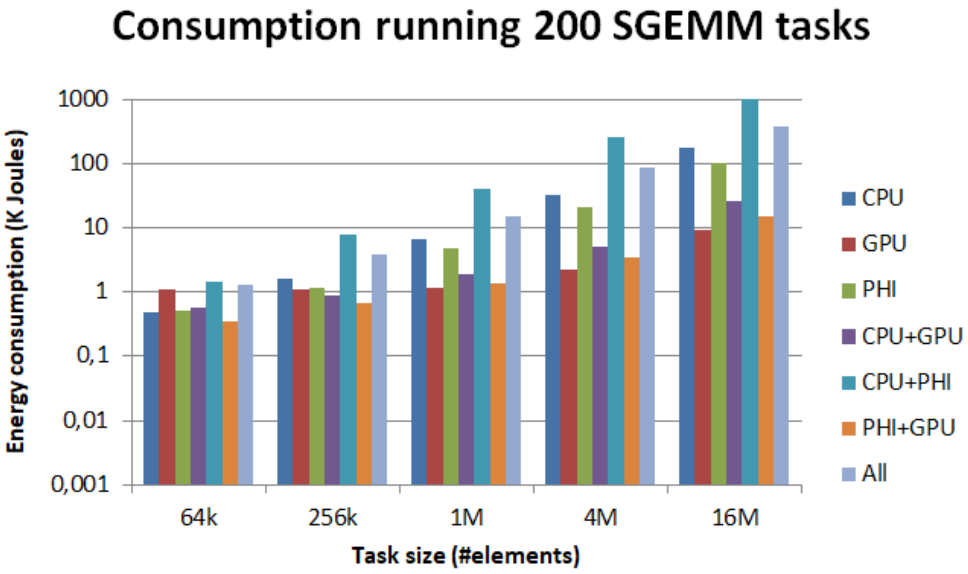


Figure 15.: Energy consumption while running 200 SGEMM tasks with different worker combinations and task sizes

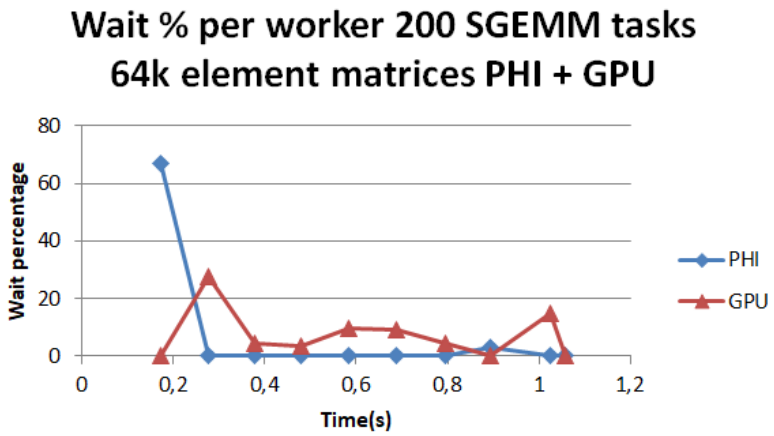


Figure 16.: Wait percentage of PHI-GPU combination for SGEMM tasks with 64k element matrices

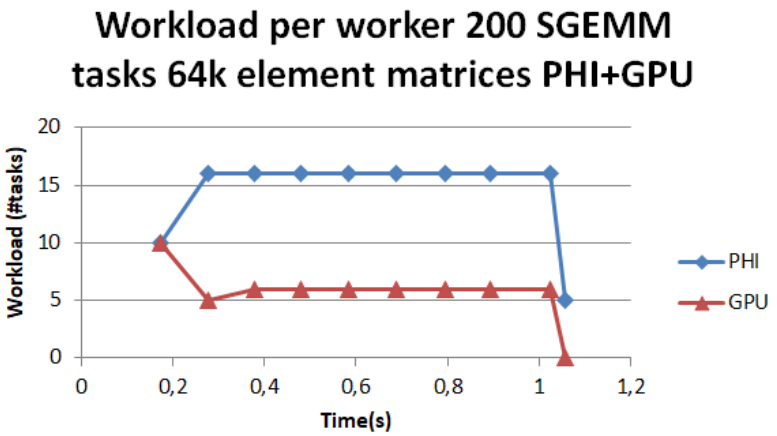


Figure 17.: Workload distribution of PHI-GPU combination for SGEMM tasks with 64k element matrices

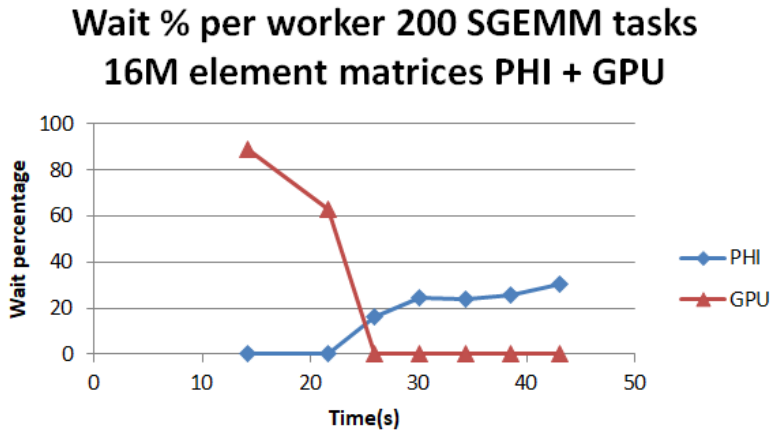


Figure 18.: Wait percentage of PHI-GPU combination for SGEMM tasks with 16M element matrices

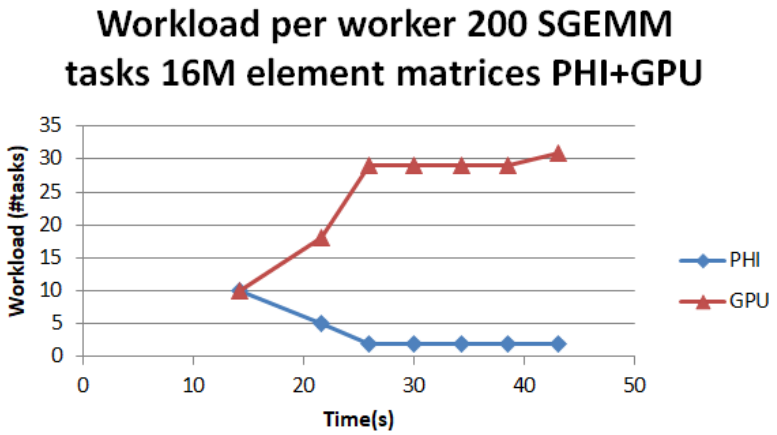


Figure 19.: Workload distribution of PHI-GPU combination for SGEMM tasks with 16M element matrices

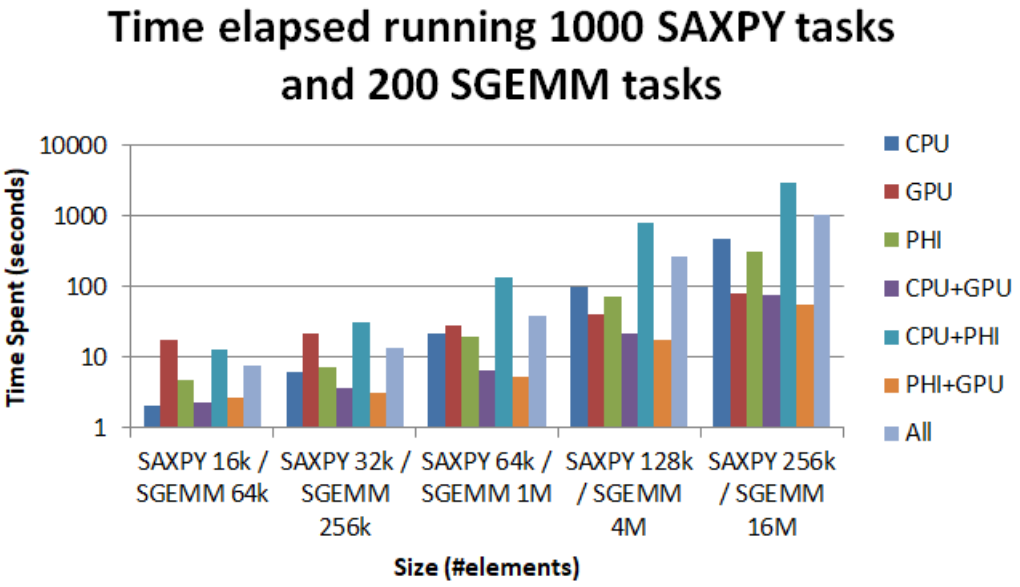


Figure 20.: Total time spent running 1000 SAXPY tasks and 200 SGEMM tasks with different worker combinations and task sizes

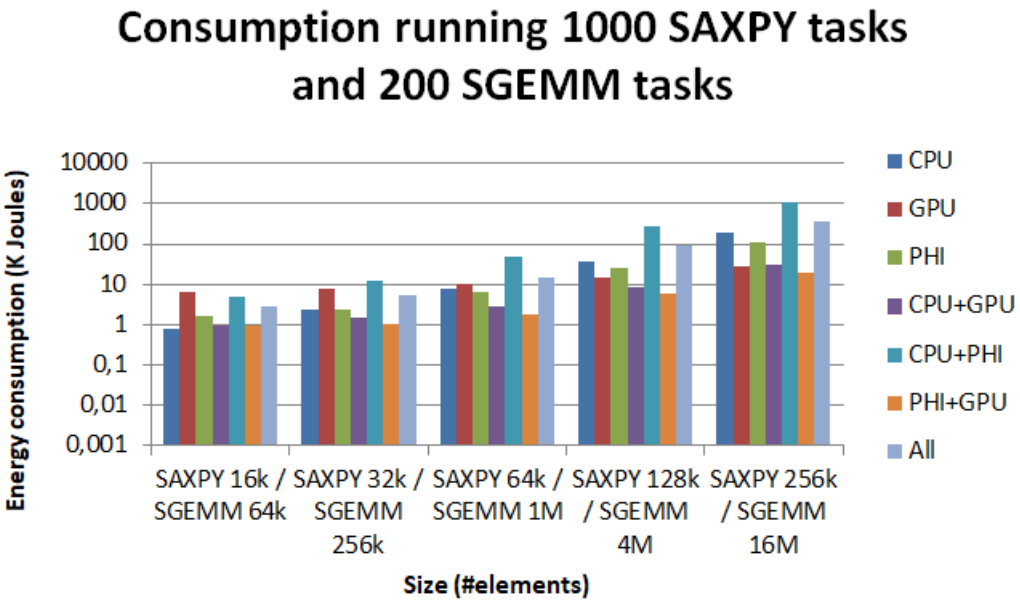


Figure 21.: Energy consumption while running 1000 SAXPY tasks and 200 SGEMM tasks with different worker combinations and task sizes

5

CONCLUSIONS

The power readers provided in this dissertation can be used as a standalone library to help the programmer identify time and power consumption hotspots within the code. These power readers can also be used by existing schedulers and frameworks to make decisions about the power consumption based on real-time readings instead of generalising the consumption based on design-time readings.

The framework created can easily evaluate the performance of different devices when running tasks of different types. Through tests made to the framework and the scheduler it is clear that the scheduling algorithm does not reduce the energy consumption of the system when running the tasks because it tries to find the best task type for each worker, instead of considering the total energy consumption of the system. Because of this, the scheduler always tries to assign tasks to the workers in spite of sometimes the idling of the workers is actually better. The workload distribution part of the scheduler works as expected by finding a balanced workload within 3 or 4 iteration. Since the framework has all of it's components built as blocks it is easy to remove the current scheduler and replace it with a better one. With no memory control it is still not possible to implement dependent tasks in this framework.

5.1 SUGGESTIONS FOR FUTURE WORK

The framework was created as a proof of concept to test the feasibility both of using the power sampler measurements to evaluate the behaviour of tasks when running on each of the devices connected to the system as well as a simple scheduling technique that tries to find the best type of work for each device based on decisions made by the user before the scheduler starts.
To keep improving the framework, here are some suggestions for future work:

- **Create a memory control for the framework** - with memory control, it is possible to add dependent tasks to the scheduler. These tasks can only run after the data they have to handle becomes available, after being used by another task. Having dependent tasks allows for much more scheduling possibilities like taking advantage of mutual exclusivity of tasks. With memory control the number of bytes transferred from and to the devices can be automatically obtained.

- **Assign a weight to the tasks** - by assigning a weight to the tasks that corresponds to the total number of operations done, it is much easier to balance the workload between different devices. The simplest task of each type would have a weight of 1 while a task that has twice the operations of the simplest one would have a weight of 2. Currently the tasks do not have a weight associated to them so each task is viewed as equal by the scheduler, so there is a great room for improvement on the load balancing here.

- **Create different schedulers and decisions** - create different scheduling decisions like decide based on the number of tasks on each task type queue. Create an entirely different scheduler like, for example, one that uses the sorting done on the currently available scheduler to order the best workers for each task type instead of ordering the best task types for each worker, as it is currently being done.

- **Make the scheduler capable of adding tasks in real time** - alter the scheduler so tasks can be added in runtime and not only at the beginning as it is currently implemented.

- **Test the scheduling decision combinations presented on section 3.1.1** - since there was not enough time to properly test the scheduling decision combinations, it is suggested they are done.

- **Compare the results obtained using this scheduler against ILP based techniques** - compare the effectiveness of the solutions found by the scheduler with the solutions obtained when scheduling based on an ILP technique.

- **Create a way to save and load the information about the devices and task types** - with a way of saving a loading the current state of the relations and history table it is possible to avoid the tuning phase and start the scheduling where the previous one ended.

BIBLIOGRAPHY

OpenMP, "Openmp homepage," http://www.openmp.org/, 2016, [Online; accessed 27-July-2017].

R. Ribeiro, J. Barbosa, and L. P. Santos, "A framework for efficient execution of data parallel irregular applications on heterogeneous systems," *Parallel Processing Letters*, vol. 25, no. 2, 2015. [Online]. Available: https://doi.org/10.1142/S0129626415500048

S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 92–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486801

C. Brabrand, M. Ribeiro, T. Tolêdo, and P. Borba, "Intraprocedural dataflow analysis for software product lines," in *Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development*, ser. AOSD '12. New York, NY, USA: ACM, 2012, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2162049.2162052

E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

F. Gruian and K. Kuchcinski, "Low-energy directed architecture selection and task scheduling for system-level design," in *Proceedings of the 25th Euromicro Conference*, 1999, pp. 296–302.

M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

Y. Wang, F. Li, R. Jiao, and X. Zhang, "A survey on energy-efficient task scheduling in multicore/multiprocessor real-time systems," in *IJACT*, vol. 4, 2012, pp. 469–476.

A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc 50th Annual Design Automation Conf*, ser. DAC '13, vol. 1. New York, USA: ACM, 2013, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488734

Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 620–625. [Online]. Available: http://dl.acm.org/citation.cfm?id=367072.367835

N. Beldiceanu, E. Bourreau, P. Chan, and D. Rivreau, "Partial search strategy in chip," in *In 2nd International Conference on Metaheuristics*, ser. MIC 97, 1997.

M. Chiesi, L. Vanzolini, C. Mucci, E. F. Scarselli, and R. Guerrieri, "Power-aware job scheduling on heterogeneous multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 868–877, March 2015.

C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *IET Computers Digital Techniques*, vol. 5, no. 2, pp. 123–135, March 2011.

P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx, and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *15th International Symposium on System Synthesis, 2002.*, Oct 2002, pp. 112–119.

TACC, "Stampede user guide," https://portal.tacc.utexas.edu/user-guides/stampede, 2017, [Online; accessed 23-July-2017].

Intel, "Xeon e5-2680 intel ark specifications," https://ark.intel.com/products/64583/Intel-Xeon-Processor-E5-2680-20M-Cache-2_70-GHz-8_00-GTs-Intel-QPI, 2017, [Online; accessed 23-July-2017].

CPU-World, "Intel xeon phi se10p specifications, available at www.cpu-world.com," http://www.cpu-world.com/CPUs/Xeon_Phi/Intel-Xeon%20Phi%20SE10P.html, 2017, [Online; accessed 23-July-2017].

CNET, "Nvidia k20 gpu specifications, available at www.cnet.com," https://www.cnet.com/products/nvidia-tesla-k20-gpu-computing-processor-tesla-k20-5-gb-series/specs/, 2017, [Online; accessed 23-July-2017].

A

LIST OF SCHEDULING DECISIONS ANALYSED AND RESPECTIVE PAPERS

When searching for the state of the art scheduling algorithms, several papers were analysed. Starting from several surveys about the current state of the art in power aware scheduling, a table containing 49 different scheduling techniques and their properties was created. 7 different properties were defined for the scheduling techniques:

- **Task reorder** - if the scheduler changes the order of the tasks being ran to decrease the energy consumption. In spite of not being used on the scheduler presented on this dissertation, being able to reorder the tasks is an advantage.
- **Data locality** - if the scheduler takes into account the locality of the data being handled when assigning tasks. Taking advantage of data locality is a great way to decrease the energy consumption by minimising the memory transfers, decreasing the energy consumption of the busses. In the scheduler presented in this dissertation, the tasks are independent from all others so there is no use for data locality.
- **Voltage control** - if the scheduler can increase or decrease the frequency of the devices when scheduling the tasks. By decreasing the frequency of a device, the power consumption of that device can be minimised. Since the Stampede supercomputer does not allow programs to change the frequency of the devices, schedulers with this technique are not wanted.
- **Cores on/off** - if the scheduler can turn individual cores from the CPU on or off. Since it is not possible to do this on the Stampede supercomputer, it is an undesirable property.
- **Communication improvement** - If there are scheduling improvements done based on reducing communications.
- **Scheduling** - When the scheduling is done. The scheduling can be done in static time, run time or both. Since the scheduler presented on this dissertation does the

scheduling in run time, schedulers that run in run time (even if they also have part of the scheduling done in static time) are preferred over static ones.

- **Heterogeneous** - If the scheduler can handle devices of different types. Since the scheduler created needs to be able to handle workers of different types at the same time, schedulers able to handle heterogeneous devices are preferred.

Scheduler	Task Reorder	Data Loc	Voltage Control	Cores on/off	Comm imprv	Scheduling	Heterogeneous
Shin and Kim (2003)	Y	Y	Y	N	N	sta, run	Y
Xie and Wolf (2001)	Y	Y	N	N	N	sta, run	Y
Zhang et al. (2002)	Y	N	Y	N	N	run	Y
Shin and Kim (2003)	Y	Y	Y	N	N	sta	Y
Luo and Jha (2007)	Y	N	Y	Y	N	run	Y
Lee and Lee (2006)	N	N	Y	Y	N	sta	Y
Qiu et al. (2008)	N	N	Y	Y	N	sta	Y
Andrei et al. (2004)	Y	N	Y	Y	Y	sta	Y
Yan et al. (2005)	N	N	Y	Y	Y	sta	Y
Lee (2009)	Y	Y	Y	Y	N	run	N
Kim et al. (2009)	N	N	Y	Y	N	run	N
Hua et al. (2007)	Y	N	Y	Y	N	run	N
Niu and Quan (2006)	Y	N	Y	N	N	sta, run	N
Seo et al. (2008)	Y	N	Y	Y	N	run	Y
Pandey et al. (2006)	N	N	Y	N	Y	run	N
Xu et al. (2011)	Y	N	N	N	Y	sta	Y
Hsu et al. (2011)	Y	Y	Y	N	Y	sta	Y
Madsen et al. (2003)	N	N	N	N	N	sta	N
Hu and Marculescu (2005a)	N	N	N	N	Y	sta, run	Y
Hu and Marculescu (2005b)	Y	N	Y	N	Y	sta	Y
Wang et al. (2010)	Y	N	Y	N	Y	sta	Y
Watanabe et al. (2007)	Y	N	Y	N	Y	sta	Y
Gruian and Kuchcinski (1999)	Y	Y	N	N	Y	sta	Y
Hsieh and Pedram (2002)	N	N	N	N	Y	sta	Y
Ruggiero et al. (2006)	N	N	N	N	Y	sta, run	Y
Chiesi et al. (2015)	Y	N	N	N	N	run	Y

List of schedulers analysed during the thesis

Scheduler	Task Reorder	Data Loc	Voltage Control	Cores on/off	Comm imprv	Scheduling	Heterogeneous
Murali et al. (2006)	N	N	N	N	Y	sta	N
Rhee et al. (2004)	N	N	N	N	Y	sta	N
Chen et al. (2008)	Y	Y	N	N	Y	sta	N
Marcon et al. (2008)	N	N	N	N	Y	sta	N
Ascia et al. (2004)	N	N	N	N	Y	sta	N
Wu et al. (2003)	N	N	Y	N	Y	sta	Y
Chou and Marculescu (2008)	Y	N	N	N	Y	run	N
Mehran et al. (2008)	Y	Y	N	N	Y	run	N
Briao et al. (2008)	Y	N	Y	N	Y	run	N
Qi et al. (2010)	N	N	Y	N	N	run	N
Chou and Marculescu (2011)	N	N	N	N	Y	run	N
Smit et al. (2004)	Y	N	N	N	Y	run	Y
ter Braak et al. (2010)	Y	N	N	N	Y	run	Y
Schranzhofer et al. (2009)	Y	N	N	N	Y	run	Y
Singh et al. (2010)	Y	N	N	N	Y	run	Y
Mariani et al. (2010)	Y	N	N	N	Y	sta, run	N
Beltrame et al. (2010)	Y	N	N	N	N	sta, run	N
Ykman-Couvreur et al. (2011)	Y	N	N	N	N	sta, run	Y
Yang et al. (2002)	Y	N	N	N	N	sta, run	Y
Zamora et al. (2007)	N	N	N	N	Y	sta, run	Y
Schranzhofer et al. (2010)	N	N	N	N	Y	sta, run	Y
Huang et al. (2011)	Y	N	Y	N	N	sta, run	Y
Singh et al. (2013)	Y	N	N	N	Y	sta, run	Y

List of schedulers analysed during the thesis (2)

BIBLIOGRAPHY

D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, ser. ISLPED '03. New York, NY, USA: ACM, 2003, pp. 408–413. [Online]. Available: http://doi.acm.org/10.1145/871506.871607

Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 620–625. [Online]. Available: http://dl.acm.org/citation.cfm?id=367072.367835

Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings of the 39th Annual Design Automation Conference*, ser. DAC '02. New York, NY, USA: ACM, 2002, pp. 183–188. [Online]. Available: http://doi.acm.org/10.1145/513918.513966

J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, 2007. [Online]. Available: https://doi.org/10.1109/TCAD.2006.885736

W. Y. Lee and H. Lee, "Energy-efficient scheduling for multiprocessors," *Electronics Letters*, vol. 42, no. 21, pp. 1200–1201, Oct 2006.

M. Qiu, J. Wu, J. Hu, Y. He, and E. H. M. Sha, "Dynamic and leakage power minimization with loop voltage scheduling and assignment," in *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 1, Dec 2008, pp. 192–198.

A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 10 518–. [Online]. Available: http://dl.acm.org/citation.cfm?id=968878.969002

L. Yan, J. Luo, and N. K. Jha, "Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1030–1041, July 2005.

W. Y. Lee, "Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors," in *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, ser. DS-RT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 216–223. [Online]. Available: http://dx.doi.org/10.1109/DS-RT.2009.12

J. Kim, S. Oh, S. Yoo, and C. M. Kyung, "An analytical dynamic scaling of supply voltage and body bias based on parallelism-aware workload and runtime distribution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 4, pp. 568–581, April 2009.

G. Hua, M. Wang, Z. Shao, H. Liu, and C. Xue, "Real-time loop scheduling with energy optimization via DVS and ABB for multi-core embedded system," in *Embedded and Ubiquitous Computing, International Conference, EUC 2007, Taipei, Taiwan, December 17-20, 2007, Proceedings*, 2007, pp. 1–12. [Online]. Available: https://doi.org/10.1007/978-3-540-77092-3_1

L. Niu and G. Quan, "Energy minimization for real-time systems with (m, k)-guarantee," *IEEE Trans. VLSI Syst.*, vol. 14, no. 7, pp. 717–729, 2006.

E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1540–1552, Nov 2008.

S. Pandey, T. Murgan, and M. Glesner, "Energy conscious simultaneous voltage scaling and on-chip communication bus synthesis," in *2006 IFIP International Conference on Very Large Scale Integration*, Oct 2006, pp. 296–301.

C. Q. Xu, C. J. Xue, and E. H. Sha, "Energy-efficient joint scheduling and application-specific interconnection design," *IEEE Trans. VLSI Syst.*, vol. 19, no. 10, pp. 1813–1822, 2011.

C. H. Hsu, S. C. Chen, and C. T. Yang, "Dynamic voltage adjustment for energy efficient scheduling on multi-core systems," in *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops*, May 2011, pp. 197–202.

J. Madsen, S. Mahadevan, K. Virk, and M. Gonzalez, "Network-on-chip modeling for system-level multiprocessor simulation," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, ser. RTSS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 265–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956418.956621

J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, April 2005.

——, "Communication and task scheduling of application-specific networks-on-chip," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 5, pp. 643–651, Sept 2005.

Y. Wang, D. Liu, M. Wang, Z. Qin, and Z. Shao, "Optimal task scheduling by removing inter-core communication overhead for streaming applications on mpsoc," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2010, pp. 195–204.

R. Watanabe, M. Kondo, M. Imai, H. Nakamura, and T. Nanya, "Task scheduling under performance constraints for reducing the energy consumption of the gals multi-processor soc," in *2007 Design, Automation Test in Europe Conference Exhibition*, April 2007, pp. 1–6.

F. Gruian and K. Kuchcinski, "Low-energy directed architecture selection and task scheduling for system-level design," in *Proceedings of the 25th Euromicro Conference*, 1999, pp. 296–302.

C.-T. Hsieh and M. Pedram, "Architectural energy optimization by bus splitting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 4, pp. 408–414, Apr 2002.

M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006*, 2006, pp. 3–8.

M. Chiesi, L. Vanzolini, C. Mucci, E. F. Scarselli, and R. Guerrieri, "Power-aware job scheduling on heterogeneous multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 868–877, March 2015.

S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. D. Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proceedings of the Design Automation Test in Europe Conference*, vol. 1, March 2006, pp. 1–6.

C.-E. Rhee, H.-Y. Jeong, and S. Ha, "Many-to-many core-switch mapping in 2-d mesh noc architectures," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, Oct 2004, pp. 438–443.

G. Chen, F. Li, S. W. Son, and M. Kandemir, "Application mapping for chip multiprocessors," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 620–625. [Online]. Available: http://doi.acm.org/10.1145/1391469.1391628

C. A. M. Marcon, E. I. Moreno, N. L. V. Calazans, and F. G. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," *IET Computers Digital Techniques*, vol. 2, no. 6, pp. 471–482, November 2008.

G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," in *Proceedings of the 2Nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '04. New York, NY, USA: ACM, 2004, pp. 182–187. [Online]. Available: http://doi.acm.org/10.1145/1016720.1016765

D. Wu, B. M. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems," *IEE Proceedings - Computers and Digital Techniques*, vol. 150, no. 5, pp. 262–73–, Sept 2003.

C. L. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip," in *2008 Design, Automation and Test in Europe*, March 2008, pp. 1232–1237.

A. Mehran, A. Khademzadeh, and S. Saeidi, "Dsm: A heuristic dynamic spiral mapping algorithm for network on chip," *IEICE Electronics Express*, vol. 5, no. 13, pp. 464–471, 2008.

E. W. Briao, D. Barcelos, and F. R. Wagner, "Dynamic task allocation strategies in mpsoc for soft real-time applications," in *2008 Design, Automation and Test in Europe*, March 2008, pp. 1386–1389.

X. Qi, D. Zhu, and H. Aydin, "Global reliability-aware power management for multiprocessor real-time systems," in *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2010, pp. 183–192.

C. L. Chou and R. Marculescu, "Farm: Fault-aware resource management in noc-based multiprocessor platforms," in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

L. T. Smit, G. J. M. Smit, J. L. Hurink, H. Broersma, D. Paulusma, and P. T. Wolkotte, "Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, Dec 2004, pp. 421–424.

T. D. ter Braak, P. K. F. Hölzenspies, J. Kuper, J. L. Hurink, and G. J. M. Smit, "Run-time spatial resource management for real-time applications on heterogeneous mpsocs," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 357–362.

A. Schranzhofer, J. J. Chen, and L. Thiele, "Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2009, pp. 151–160.

A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *Journal*

of Systems Architecture, vol. 56, no. 7, pp. 242 – 255, 2010, special Issue on HW/SW Co-Design: Systems and Networks on Chip. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762110000330

G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria, "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 196–201.

G. Beltrame, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1083–1095, July 2010.

C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *IET Computers Digital Techniques*, vol. 5, no. 2, pp. 123–135, March 2011.

P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx, and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of the 15th International Symposium on System Synthesis*, ser. ISSS '02. New York, NY, USA: ACM, 2002, pp. 112–119. [Online]. Available: http://doi.acm.org/10.1145/581199.581226

N. H. Zamora, X. Hu, and R. Marculescu, "System-level performance/power analysis for platform-based design of multimedia applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 1, pp. 2:1–2:29, Feb. 2007. [Online]. Available: http://doi.acm.org/10.1145/1188275.1188277

A. Schranzhofer, J. J. Chen, and L. Thiele, "Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 692–707, Nov 2010.

L. Huang, R. Ye, and Q. Xu, "Customer-aware task allocation and scheduling for multi-mode mpsocs," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 387–392. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024816

A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, pp. 9:1–9:29, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2390191.2390200