This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

Mapping streaming applications on commodity multi-CPU and GPU on-chip processors

Antonio Vilches*, Angeles Navarro*, Rafael Asenjo*, Francisco Corbera*, Rubén Gran†, María Garzarán‡

* Universidad de Málaga, Spain. E-mail: {avilches, angeles, asenjo, corbera}@ac.uma.es

† Universidad de Zaragoza, Spain. E-mail: rgran@unizar.es

‡ Dept. Computer Science, UIUC, USA. E-mail: garzaran@illinois.edu

Abstract

In this paper, we consider the problem of efficiently executing streaming applications on commodity processors composed of several cores and an on-chip GPU. Streaming applications, such as those in vision and video analytic, consist of a pipeline of stages and are good candidates to take advantage of this type of platforms. We also consider that characteristics of the input may change while the application is running. Therefore, we propose a framework that adaptively finds the optimal mapping of the pipeline stages. The core of the framework is an analytical model coupled with information collected at runtime used to dynamically map each pipeline stage to the most efficient device, taking into consideration both performance and energy. Our experimental results show that for the evaluated applications running on two different architectures, our model always predicts the best configuration among the evaluated alternatives, and significantly reduces the amount of information that needs to be collected at runtime. This best configuration has, on the average, 20% higher throughput than the configuration recommended by a baseline state of the art approach, while the ratio throughput/energy is 43% higher. We have measured improvements in throughput and throughput/energy of up-to 81% and 204%, respectively, when the model is used to adapt to a video that changes from low to high definition.

Index Terms

Heterogeneous CPU-GPU chips, pipeline pattern, adaptive mapping, analytical model, energy aware.

I. Motivation

Recently, we have seen a significant increase in the number of commodity multicore processors that include an on-chip GPU. Current desktops, ultrabooks, smartphones, tablets, and other embedded devices are powered by heterogeneous chips that comprise 2 to 8 CPU cores along with an integrated GPU.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

Examples of these are Intel Ivy Bridge and Haswell architectures, AMD APU, Qualcomm Snapdragon 800 and Samsung Exynos 5 Octa, to name a few. These heterogeneous chips can deliver significant speedups and low energy consumption compared to CPU-only systems on a large range of applications. However, issues such as the development of a suitable programming framework and runtime support for these architectures are in their infancy.

Most research in frameworks aimed at scheduling tasks on heterogeneous architectures, composed of CPU's and GPUs, has focussed on optimizing execution time without considering energy consumption [1], [2], [3], [4], [5]. However, a CPU core and a GPU exhibit different performance/energy trade-offs, this is, a workload can run faster on one device but consume less energy on the other one. Thus, in order to benefit from the potential energy efficiency that the accelerators can provide in these heterogeneous chips, the runtime scheduler also needs to consider the performance/energy asymmetry when making a scheduling decision [6].

In this paper, we focus on the problem of efficiently executing single streaming applications implemented as a pipeline of stages that run on heterogeneous chips comprised of several cores and one on-chip GPU, taking into consideration both performance and energy. Streaming applications are very common in today's computing systems, in particular mobile devices [7] where heterogeneous chips are the dominant platforms. To tackle the aforementioned problem, we study different choices such as: i) the granularity level at which the parallelism of each stage can be exploited (coarse or medium grain), ii) the mapping of the pipeline stages to the different computational devices and iii) the number of cores for which the application scales up. Our aim is to find the best configuration that considers all these factors. We also consider that the best configuration may change over time. This can happen because the number of operations performed by each pipeline stage changes. There are several reasons why this can take place. For instance, YouTube, Skype Video [8], or tele-operated robots [9] adjust the resolution of the video stream based on the bandwidth of the network connection. Also, the computation of a pipeline stage may depend on the characteristics of the input frame. In this situation, an off-line training may not be feasible, as the best configuration will depend on the runtime input.

As a motivating example to demonstrate the benefits of adapting the configuration of a pipeline we introduce ViVid, an application that implements an object (e.g., face) detection algorithm [10] using a "sliding window object detection" approach [11]. ViVid consists of 5 pipeline stages from which the first and the last one are the Input and Output stages. When applications like ViVid run on a heterogeneous on-chip architecture, many possible configurations are possible. To determine the best configuration, one needs to consider the granularity or number of items that should be simultaneously processed on each

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

stage, the device where each stage should be mapped, and the number of CPU cores that minimize the execution time, or the energy consumption, or both. As we will discuss in Section V, we have found that when ViVid runs on an Intel Ivy Bridge platform (also presented in Section V), the best configuration for videos with Low Definition (LD) is different from the best one for videos with High Definition (HD) and not adapting to an input change, can have a significant impact in both, execution time and energy. For instance, when the video resolution changes from LD to HD, not changing from the best configuration for LD to the new optimal for HD results in 0.55x of throughput loss (and 1.7x of more energy used). On the other hand, if we are using the optimal configuration for HD, an input change from HD to LD will result in a 0.76x of throughput degradation (and 1.1x of more energy used) if we do not change to the new best configuration. These results indicate that an approach that can predict the best configuration, out of all the possible ones, is desirable. This approach should have low overhead, so that it can be used when an input change is detected.

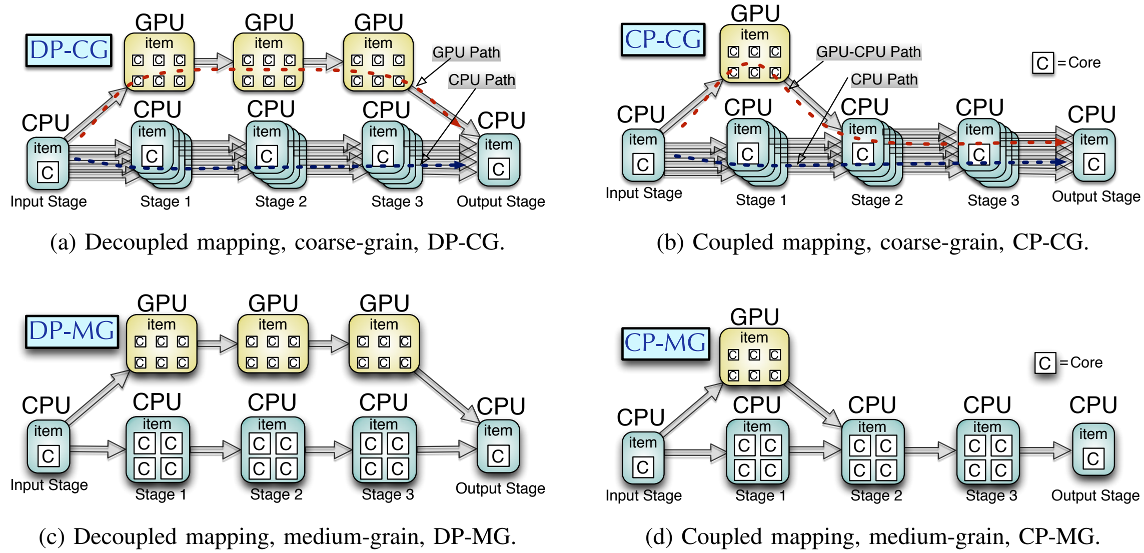
In this paper, we propose an adaptive framework that can dynamically adjust the configuration of the pipeline (granularity, mapping and number of cores). This framework is based on an analytical model that, by collecting a small number of runtime experiments (only 7 on a quad-core), can predict the optimal pipeline configuration. Our framework can be targeted at optimizing performance, or energy or a tradeoff metric that considers the ratio throughput/energy. Our analytical model can provide knobs so that the user can specify a desired throughput or power budget. For instance, if the user specifies a throughput of 33 fps for real time video streaming, the model can determine among the possible pipeline configurations, the one that minimizes the energy consumption and satisfies the user constraint. Similarly, given a power budget, the model can determine the fastest configuration. The information collected using runtime input data are used to dynamically adapt to input changes. Since this data collection phase can add some runtime overhead, our framework provides another knob so that the user can specify a threshold to limit the maximum overhead of this phase. We have evaluated our model using a set of streaming applications from vision and video analytic domain that are representative of the algorithms [12] that can benefit from the execution on these heterogeneous chips.

The contributions of this paper are the following ones:

- A taxonomy of the pipeline configurations for heterogeneous chips (section II).
- An adaptive framework that dynamically selects the best configuration while keeping the runtime overhead below an user-defined limit (section III).
- An analytical model that quantifies how the different implementation factors interplay. This model can be used to predict the optimal granularity and mapping of the pipeline stages to the different

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems



(a) Decoupled mapping, coarse-grain, DP-CG.

(b) Coupled mapping, coarse-grain, CP-CG.

(c) Decoupled mapping, medium-grain, DP-MG.

(d) Coupled mapping, medium-grain, CP-MG.

Fig. 1: Examples of the four different configurations for ViVid.

computational devices, as well as the appropriate number of threads (section IV).

- An evaluation of the accuracy of our analytical model. Our results demonstrate that the model accurately predicts, among the evaluated alternatives, the best pipeline configuration for all the applications and architectures studied (section V).

II. PIPELINE IMPLEMENTATION ALTERNATIVES

We use two axes to classify the different alternatives: (1) *granularity level*, that represents the level at which the parallelism is exploited on the CPU; and (2) *pipeline mapping*, that represents where the different stages of the pipeline can execute. We call pipeline configuration to each possible combination of granularity and mapping. Fig. 1 graphically depicts examples of the 4 possible configurations for the ViVid pipeline on an Ivy Bridge-like architecture with a GPU (6 computing units) and a CPU multicore (4 CPU cores). The figure shows the paths that traverse the in-flight items being processed. The pipeline stages are represented as rounded rectangles, while the device (GPU or CPU) on which each stage is processed, is depicted with the number of computing resources (small squares with the letter 'C') that collaborate on the computation of each item.

Granularity level: The vertical axis in Fig. 1 classifies the approaches based on the granularity level used to exploit parallelism on the CPU. Two levels of granularity are considered: Coarse Grain (CG) and Medium Grain (MG). If different items can be processed simultaneously and each CPU core (thread) can process one item through all the stages, then CG granularity can be exploited. On the other hand, if the pipeline stages exhibit nested parallelism (which can be exploited by using OpenCL, OpenMP or TBB

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

parallel_for), then a single item can be processed in parallel by several cores in the CPU, and MG granularity can be exploited. The CG granularity is shown in Figs. 1a and 1b. MG granularity is shown in Figs. 1c and 1d.

GPUs are not as flexible as the multicores regarding the granularity level of parallelism they can exploit. They excel at exploiting SIMT (Single Instruction Multiple Threads) type of parallelism. Thus, stages mapped onto a GPU only process a single item, with all the GPU processing units computing a portion of the item.

The MG granularity requires a barrier synchronization after each pipeline stage and before the next pipeline stage can start, to guarantee that all participating threads have finished processing the item. Therefore, MG can hurt performance when the load is imbalanced or there is not enough computational load per core. With MG, it is like having two devices, GPU and CPU, that can only work on two different items at a time. Thus, there is less pipeline parallelism when exploiting MG granularity. However, with the CG granularity, each CPU core (or thread) can process all the pipeline stages for a given item without intermediate synchronizations, i.e. each item traverses the pipeline at its own pace. Two drawbacks of the CG approach are that several items are in-flight at the same time, increasing the memory pressure, and that only applies to parallel pipeline stages (i.e. stateless pipeline stages). Notice that CPU cores can also exploit fine grain parallelism, due to the vector units of the processors, orthogonally to both, CG or MG granularities.

Pipeline mapping: The horizontal axis in Fig. 1 classifies the configurations based on whether all the stages execute on the GPU or only a few do. The first pipeline mapping is called decoupled (DP), while the other one is called coupled (CP).

Disregarding the Input/Output stages, DP mappings are illustrated in Figs. 1a and 1c, where we depict two "decoupled" paths: i) a *GPU path*, in which a thread (the GPU thread) offloads all stages to the GPU for processing one input item; and ii) a *CPU path*, in which a group of concurrent threads (the CPU threads) process all stages on the CPU. On the other hand, CP mappings are shown in Figs. 1b and 1d where we see two paths: i) a *GPU-CPU path* in which a thread (the GPU-CPU thread) offloads some stages to the GPU (stage 1 in the figures) for processing one input item, while the remaining stages are executed on the CPU; and ii) a CPU path, in which a group of concurrent threads (the CPU threads) process all stages on the CPU multicore. The difference between the CP's GPU-CPU thread and the DP's GPU thread is the following. In a CP mapping, when an item reaches the stage for which it has been decided that it will be processed on the GPU (stage 1 for the ViVid example), we first check if the GPU is idle, and in that case the thread becomes a GPU-CPU thread that launches the item's kernel to the

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

GPU and then waits for the GPU kernel to finish. Then, the same thread also processes the item through the remaining stages (in the example, stages 2 and 3 that are processed in the CPU). However, in DP, when an item reaches the first stage and finds the GPU is idle, the corresponding thread becomes a GPU thread that processes the item throughout all the stages on the GPU. Indeed, when we consider only 1 thread for the DP mapping, that thread becomes the GPU thread and therefore all the items traverse the GPU path. This is what we call a GPU homogeneous execution. In our example, for both CP and DP, if an item on the stage 1 finds that the GPU is already busy, then the item is directed through the CPU path. Although DP could be seen as a particular case of CP where all the stages happen to be mapped to the GPU, we distinguish both mappings because they have to be modelled differently as we will see in section IV.

CP mappings can be a good alternative when not all the stages are suitable for the GPU, or because it's not advisable to divert the GPU computing power from the stages where it is faster and/or more energy-efficient. This approach also has the advantage that not all the stages have to be implemented for the CPU and GPU. However, in the CP mapping, the GPU-CPU thread must orchestrate the "coupling" of the GPU and the CPU devices and the host-to-device/device-to-host communications, which results in some data movement and synchronization overheads. Also, note that DP mappings can be implemented only if all stages are parallel pipeline stages (stateless). If, on the contrary, all stages are serial, heterogeneity can be exploited by mapping some stages on the GPU and the rest on the cores, which is a particular case of the CP mapping in which all items are directed through the GPU-CPU path.

A. *Alternatives not considered*

Some additional alternatives not considered in this classification are the following:

- Splitting an item to be simultaneously computed on the CPU and GPU. As it was demonstrated by Totoni et al. [13], this possibility is not beneficial for our vision applications due to additional synchronization overheads between both devices.
- Having one stage exploiting both MG and CG granularities on the CPU. For example, a quad-core can be splited into two CPU devices with two cores each. This approach would combine CG and MG on the same stage: two CPU devices processing two items in parallel (CG), and each item running on two cores (MG). For that, we explored the OpenCL Device Fission function (`cl_ext_device_fission`) that can divide the CPU device into several subdevices with lower core count. However, we discarded this alternative due to we measured a 14% of overhead (for ViVid on Ivy Bridge) if the `device_fission` is called to change the subdevices configuration from one

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

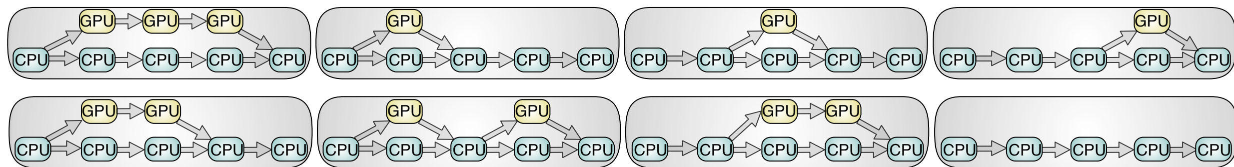


Fig. 2: All considered mappings of pipeline stages to CPU and/or GPU for ViVid.

pipeline stage to the next one. Thus, this approach is beneficial only if the optimum number of cores per device coincides for all the pipeline stages so the fission function is just called once.

- Exploiting stages with both MG and CG granularities on the GPU. The OpenCL fission feature is currently not able to split the GPU on Intel or AMD heterogeneous chips. Therefore, we do not consider this feature to evaluate additional pipeline configurations.
- Hybrid mappings in which some CPU stages exploit MG and the rest CG granularity. This is left for future work.

B. Accounting for all pipeline alternatives

Let's assume we have nC CPU cores (4 in Fig. 1), and 1 GPU in an heterogeneous chip (current commercial heterogeneous chips only contain a single GPU, so we overlook configurations with two or more GPUs in this work). In addition to the 4 pipeline configurations, DP-CG, DP-MG, CP-CG and CP-MG, there are two additional factors to consider: i) for CP mappings we have to find out the stages for which the GPU is more profitable; and ii) for the CG granularity we also have to explore the optimal number of threads. For this CG granularity the number of threads in the CPU multicore can go from 0 to nC. Additionally, since the GPU thread in DP-CG, or the GPU-CPU thread in CP-CG, will be mainly hosting the GPU (waiting for the GPU kernel to complete), the total number of threads, n, we explore goes from 1 to $nC+1$. This means that we allow oversubscription of one thread when $n = nC+1$. For the MG granularity, we always configure $nC+1$ threads because the constructors used to exploit nested parallelism (OpenCL, OpenMP or TBB `parallel_for`) by default use all the threads available in the multicore, nC, plus the GPU (or GPU-CPU) thread.

With all that, assuming that the pipeline consists of s parallel stages, there would be 2^s possible GPU/CPU mappings (this is shown in Fig. 2 for ViVid with $s = 3$). These mappings can be combined with $nC + 1$ different CG options, depending on the number of threads used and 1 MG option, i.e., $nC+2$ options. Thus, in total we have $2^s \cdot (nC+2)$. That results in 48 alternatives for ViVid with $s = 3$ and $nC = 4$.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

Our goal is to be able to predict the optimal pipeline configuration specifying the granularity, mapping (identifying the stages that should be mapped on the GPU), and the optimum number of threads for a given stream input. But first, the general framework is presented.

III. Framework

Our framework is particularly suitable for streaming applications that may exhibit a variation in the streaming characteristics. In these cases, we can adjust the pipeline configuration to optimize the desired metric (throughput, or energy, or a tradeoff). The interested reader can find more details about the API of our pipeline library in [14]. Our framework is designed as a two phase engine: first, a *training phase* followed by a *running phase*. The training phase carries out two steps: i) *a measurement collection step*, where some measurements of time and energy are performed on the GPU and CPU; and ii) *an evaluation step*, where our model (see next section) finds the optimal pipeline configuration using the collected measurements. During the training phase, runtime items are used, so no off-line training is necessary. Also, the runs to collect the measurements are conducted only on the CPU or on the GPU (homogeneous runs). Once the evaluation step finds the optimal configuration, the framework enters the running phase. In order to adapt to variations in the behavior of the applications, throughput is monitored during this running phase, so that any significant change can return the framework to the training phase. However, to limit the overhead of the the training phase, training is only performed when its associated overhead is less than a threshold value provided by the user (more details in subsection III-B).

Let's assume that the s parallel stages of our streaming application are S_1, S_2, .., S_s and that each item will be executed through all these stages. Our model is based on a set of equations that allow us to estimate the throughput and energy consumption per item for all possible pipeline alternatives. Let's suppose that our system consists of nC CPU cores and 1 on-chip GPU. Then, our framework invokes the model's equations for the $2^s \cdot (nC+2)$ possible pipeline configurations, and for each one computes the effective throughput, λ_e, and the effective energy per item, E_e. From the estimations, it selects the pipeline configuration for which the optimal is found: highest λ_e or lowest E_e, depending on the metric considered. We can also use any combination of these metrics to define a tradeoff metric and look for the configuration which obtains the optimal value.

A. Measurement Collection step

In this step, we carry out $nC+3$ experiments to obtain all the values needed by the model. Note that this number of experiments is usually much smaller than the $2^s \cdot (nC+2)$ possible alternatives, that thanks to the model we do not need to experimentally assess. For time measurements we use the clock

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

ticks hardware counter, while for the energy measurements, we use the energy counters available on the Ivy Bridge and the Haswell architectures [15], [16]. These counters measure three domains: P, C and G. P or Package means the consumption of the whole chip, including CPU, GPU, memory hierarchy, etc. C is CPU domain and G is the GPU domain. In our model we consider C, G and $U = P - C - G$. Therefore, this last component represents the Uncore energy consumption. For other architectures, energy information can be estimated by either relying on performance counters that can be read by using a library, such as PAPI [17]. Anyway, even if energy information is not accesible, our framework is still useful to minimize execution times.

The experiments and measurements we collect are always from homogeneous runs (only GPU or CPU execution). These experiments are:

- CG experiments: we perform 1 experiment in which all stages are executed by one thread in one CPU core. We collect time and energy per stage (see T_k^{CG} and $\left(E_{C_k}^{CG}, E_{G_k}^{CG}, E_{U_k}^{CG}\right)$, $k = 1 : s$, in Table I). For energy measurements we collect three components (C, G, U) as explained before. Next, we carry out nC additional experiments in the CPU multicore: on each one, n threads (with n changing from 2 to $nC + 1$) process n items (each thread processes one item) throughout all the pipeline stages, i.e. homogeneous CG executions. We collect the total time and energy per item (see $T^{CG}(n)$ and $\left(E_C^{CG}(n), E_G^{CG}(n), E_U^{CG}\right)$, $n = 2 : nC + 1$, Table I). Note that the case for one thread was already considered in the first experiment described above. Actually, $T^{CG}(1) = \sum_{k=1}^{s} T_k^{CG}$ and $E_*^{CG}(1) = \sum_{k=1}^{s} E_{*_k}^{CG}$, where $*$ takes the value C, G and U. With these measurements we implicitly incorporate to the model the impact that n threads processing n items have in the memory hierarchy as well as the scalability behavior in the CPU. To carry out these $nC + 1$ experiments, $(nC + 2) \cdot (nC + 1)/2$ items of the stream are processed.
- MG experiments: we conduct 2 additional experiments in which all stages are executed first by one thread on the GPU, and next by nC threads on the CPU multicore, i.e. homogeneous MG execution, where nC is the number of CPU cores. We collect time and energy per stage (see T_k^G and $\left(E_{C_k}^{G}, E_{G_k}^{G}, E_{U_k}^{G}\right)$, $k = 1 : s$, for the GPU and T_k^{MG} and $\left(E_{C_k}^{MG}, E_{G_k}^{MG}, E_{U_k}^{MG}\right)$, $k = 1 : s$, for MG on the CPU, in Table II). Now, 2 additional items of the stream are processed to carry out these 2 MG experiments.

B. Controlling the overhead of the measurement collection step

The cost of the training phase is mainly due to the measurement collection step, where items are processed inefficiently due to the homogeneous runs (only one device is used at a time) carried out during this step. After the measurement collection step and the subsequent model instantiation, we can

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

TABLE I: Measured time per item, T, and energy per item, E, for CG. Also time to collect them. Note that in practice, $T^{CG}(1) = \sum_{k=1}^{s} T_k^{CG}$, $E_*^{CG}(1) = \sum_{k=1}^{s} E_{*_k}^{CG}$, where $*$ takes the value C, G and U, and $t^{CG}(1) = t^{CG}$.

Parameter	Device	time col.	Description
$T_1^{CG}, \ldots, T_s^{CG}$	CPU	t^{CG}	time per item (and stage) on the CG exec. (1 thread)
$(E_{C_1}^{CG}, E_{G_1}^{CG}, E_{U_1}^{CG})$ ... $(E_{C_s}^{CG}, E_{G_s}^{CG}, E_{U_s}^{CG})$	CPU		(C,G,U) components of the energy per item (and stage) on the CG exec. (1 thread)
$T^{CG}(1), \ldots T^{CG}(n_m)$	CPU	$t^{CG}(1), \ldots t^{CG}(n_m)$	total time per item on the CG exec. (1, 2 ... $n_m = nC + 1$ threads)
$(E_C^{CG}(1), E_G^{CG}(1), E_U^{CG}(1))$ ... $(E_C^{CG}(n_m), E_G^{CG}(n_m), E_U^{CG}(n_m))$	CPU		(C,G,U) comp. of the total energy per item on the CG exec. (1, 2 ... $n_m = nC + 1$ threads)

TABLE II: Measured time per item, T, and energy per item, E, and per stage for GPU and for MG. Also time to collect them.

Parameter	Device	time col.	Description
$T_1^{G}, \ldots, T_s^{G}$	GPU	t^{G}	time per item (and stage) on the GPU exec. (1 thread)
$(E_{C_1}^{G}, E_{G_1}^{G}, E_{U_1}^{G})$ ... $(E_{C_s}^{G}, E_{G_s}^{G}, E_{U_s}^{G})$	GPU		(C,G,U) components of the energy per item (and stage) on the GPU exec.
$T_1^{MG}, \ldots, T_s^{MG}$	CPU	t^{MG}	time per item (and stage) on the MG exec. (nC threads)
$(E_{C_1}^{MG}, E_{G_1}^{MG}, E_{U_1}^{MG})$ ... $(E_{C_s}^{MG}, E_{G_s}^{MG}, E_{U_s}^{MG})$	CPU		(C,G,U) components of the energy per item (and stage) on the MG exec.

control the time when a new training can be performed to guarantee that the overhead due to the training is bounded. Suppose that after performing the training step, λ_c is the throughput of the current configuration and that $N_s = (nC + 2) \cdot (nC + 1)/2 + 2$ is the number of items processed during the measurement collection step (see Tables I and II). Then, Δt can be defined as the time penalty due to the training. It is computed as the time needed to carry out the collection step minus the time it takes to compute N_s items with the current λ_c throughput:

$$\Delta t = \left(t^{CG} + \left(\sum_{n=2}^{nC+1} t^{CG}(n) \right) + t^{MG} + t^{G} \right) - N_s/\lambda_c \tag{1}$$

The *overhead* of the last training with respect to the current throughput can be computed as,

$$ov = \frac{\Delta t}{t + \Delta t}$$

We can keep this overhead below a threshold value, ov_{thl}, if $\Delta t/(\Delta t + t) < ov_{thl}$, or in other words:

$$t > \frac{(1 - ov_{thl})}{ov_{thl}} \cdot \Delta t \tag{2}$$

For the ViVid application on the Ivy Bridge chip presented in section V, 5% of overhead is paid when the training takes place every 3.7 sec. for low resolution input video.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

IV. Analytical model: finding the optimal

We model the heterogeneous pipeline configurations as a closed network of logical queues where items arrive following a Poisson process [18]. This is pertinent in the context of streaming applications [19] where item arrivals can be considered independent and inter-arrival time can be viewed as following an exponential distribution. In these closed systems, items can be viewed as circulating continuously and never leaving the network of queues, because a new item can not enter until a previous one leaves. Fig. 3 shows our models for the Decoupled and Coupled configurations, where we can see that an item can follow one of two alternative paths before entering again in the system. In our models, we can find one or more queues on each path. In particular, any sequence of consecutive stages mapped to one device (the GPU or the CPU) is represented as a M/M/1 queue. This stands for a logical queue where a single server serves items that arrive according to a Poisson process and have exponentially distributed service times. Although there can be several concurrent threads on a device processing the sequence of stages represented by the queue, we have found that assuming one logical server on each queue still provides accuracy while keeping the equations of the model simple. In a closed network of queues, the following expressions define the *flow balance conditions* [20] at equilibrium,

$$\lambda_e = \sum_{path_j} \lambda_j \tag{3}$$

$$\sum_{path_j} p_j = 1 \tag{4}$$

$$p_j \cdot \lambda_e = \lambda_j \tag{5}$$

These equations allow us to relate the relative throughput of a path in a configuration with the effective throughput in that configuration. A path $path_j$ refers to one of the two possible paths defined in section II for each configuration: for DP configurations, it is either the GPU path or the CPU path (note the subindices for the parameters on each path of the model (GPU, CPU) in Fig. 3a); For CP configurations, it is either the GPU-CPU path or the CPU path (note the subindices for the parameters on each path of the model (GPU-CPU, CPUB) in Fig. 3b). In particular, equation 3 establishes that given the relative throughputs of the paths in the system, λ_j, then the effective throughput, λ_e, may be obtained as a sum (i.e. combining independent Poisson processes leads to a Poisson process). Equation 4 states that splitting a Poisson process probabilistically leads to Poisson processes, being p_j the probability of taking $path_j$. Equation 5 states that, in a M/M/1 queue at equilibrium, the average flow rate leaving the queue will also be the same as the average flow rate entering the queue.

We define two parameters for each queue Q_i: the *service rate*, μ_i, or average rate at which an item is processed, and the *energy rate*, $\vec{\varepsilon}_i$, or average energy consumed by an item in the corresponding device (GPU or CPU) where the queue works. This last parameter represents a vector with three components,

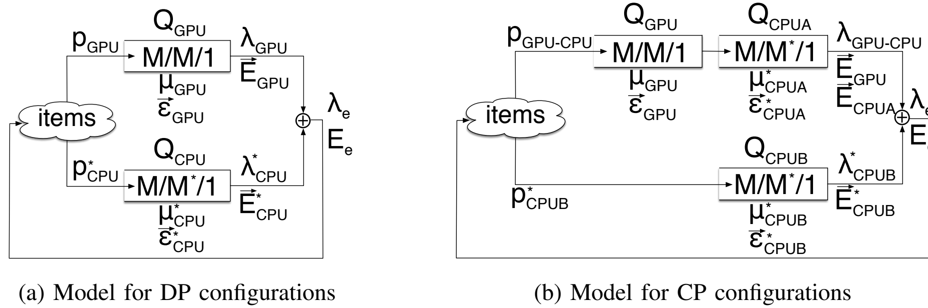
1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

one for each energy domain: $(\varepsilon_{i_C}, \varepsilon_{i_G}, \varepsilon_{i_U})$. They can be seen as the components of the average energy consumed by an item on a device due to the stages represented by the queue Q_i, when the device is the only one working in the system (homogeneous execution).

In any case, as the network is in equilibrium, each individual queue must be in equilibrium. This means that the utilization on the queue, ρ_i, is less than 100%, that is, the ratio between the relative throughput of the corresponding path, λ_j, and the queue's service rate, μ_i, is at most 1 [18],

$$\rho_i = \frac{\lambda_j}{\mu_i} \leq 1 \tag{6}$$



(a) Model for DP configurations (b) Model for CP configurations

Fig. 3: Closed network of queues.

Regarding the energy, as each individual queue Q_i is in equilibrium, we assume that the energy utilization on the corresponding device, $\rho_i^{\vec{E}}$, is proportional to the probability of items serviced on the corresponding queue, p_j, or in other words,

$$\rho_i^{\vec{E}} = p_j \leq 1, \quad \vec{E}_i = \rho_i^{\vec{E}} \cdot \vec{\varepsilon}_i \tag{7}$$

This expression allows us to estimate the *relative energy per item* consumed by queue Q_i on the corresponding device (GPU or CPU), $\vec{E}_i$. This parameter is also a vector that consists of three components: $(E_{i_C}, E_{i_G}, E_{i_U})$. In the case there were several logical queues mapped on a device, from Q_1 to Q_d, then the relative energy per item consumed by the queues in the device would be the sum of the relative energy per item for all the queues working in the device: $\sum_{i=1}^{d} \vec{E}_i = \left(\sum_{i=1}^{d} E_{i_C}, \sum_{i=1}^{d} E_{i_G}, \sum_{i=1}^{d} E_{i_U}\right)$. These components can be seen as the components of the energy consumed by the items that a device processes when the device is the only one working in the system (homogeneous execution).

However, the effective energy consumed by the GPU and CPU when both devices are working in the system (heterogeneous execution), is not the sum of the relative energies of the queues on each device. Let's define the *effective energy per item* consumed in the system, E_e, as,

$$E_e = min\left(\frac{TDP}{\lambda_e}, E_{e_C} + E_{e_G} + E_{e_U}\right) \tag{8}$$

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

where TDP is the power budget of the chip, λ_e the effective throughput, and $E_{e_C} + E_{e_G} + E_{e_U}$ the effective energy consumed when the TDP is not reached. For the heterogeneous chips studied, we have found that in case the TDP is not reached, then each component of the effective energy is given by the dominant component of the relative energy computed for each device. This is what we call the *energy balance condition*. Let's suppose that the relative energy per item for all the queues in the GPU device is given by $\overrightarrow{E}_{GPU} = (E_{GPU_C}, E_{GPU_G}, E_{GPU_U})$, and the relative energy per item for all the queues in the CPU device is given by $\overrightarrow{E}_{CPU} = (E_{CPU_C}, E_{CPU_G}, E_{CPU_U})$. The rationale for the energy balance condition is that the C-component of the effective energy is typically dominated by the C-component of the relative energy of the CPU device, E_{CPU_C}, while the C-component of the GPU device, E_{GPU_C} is just a "residual" or standby consumption when the CPU is idle. Remember that this last C-component of the relative energy of the GPU device is obtained with homogeneous runs (runs on the GPU where the CPU is idle) during the measurement collection step. On an heterogeneous run, however, the CPU is not idle, and so the standby consumption measured during the homogeneous run is already included in the C-component of the relative energy of the CPU, E_{CPU_C}, and does not need to be included again. A similar argument can be made for the G-component of the effective energy. With respect to the U-component, we have observed that the effective energy consumed is determined by the device (CPU or GPU) that processes a higher load.

Next, Sections IV-A and IV-B explain how we model Decoupled and Coupled configurations, respectively, and how we incorporate the granularity to the models.

A. Model for Decoupled configurations

These configurations are shown in Figs. 1a and 1c (DP-CG and DP-MG, respectively). Fig. 3a depicts our model for them. As explained in section II, in these configurations there is a GPU path in which a thread processes an item through all stages in the GPU, and also there is a CPU path in which n concurrent threads process other item/s through all the stages in the CPU device.

The GPU device is modeled with Q_{GPU} which is the M/M/1 queue that services all the stages for the items that go through the GPU path. This queue is characterized with two parameters: μ_{GPU}, the service rate of the GPU, and $\overrightarrow{\varepsilon}_{GPU}$, the energy rate consumed by the queue in the GPU device. These parameters are computed from the time and energy measurements taken in the collection step, as we show in Table III. For both parameters we consider the time and the energy per item of all the stages S_k that are mapped to the GPU (k from 1 to s, see Table II).

The CPU device is modeled with Q_{CPU} which is the M/M*/1 queue that services all the stages for the

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

items that go through the CPU path. The * stands for the different instantiations of the queue, depending on the granularity exploited. For the CG granularity, the queue is characterized with two parameters: $\mu_{CPU}^{CG}(n)$, the service rate of the CPU under CG granularity, and $\overrightarrow{\varepsilon}_{CPU}^{CG}(n)$, the energy rate consumed by the queue in the CPU device under CG granularity. Note that under the CG granularity the CPU device can run from 0 to nC concurrent threads. The $n = 0$ case represents in fact the GPU homogeneous execution, while the $n = nC$ represents the maximun number of threads in the CPU path. Therefore, for the CG granularity, both the service rate and the energy rate are computed for each possible number of threads. Table III shows how these parameters are computed, where we see that time and energy are taken from the measurements in Table I. Regarding the MG granularity, the queue is defined by μ_{CPU}^{MG} and $\overrightarrow{\varepsilon}_{CPU}^{MG}$. In Table III we show these parameters, where we notice that time and energy are taken from the measurements in Table II.

TABLE III: Parameters of the DP-* configurations. * stands for CG or MG. s is the number of stages.

Parameter	Device / Gr.	Value	Description
μ_{GPU}	GPU	$\frac{1}{\sum_{k=1}^{s} T_k^G}$	service rate for the stages mapped to the GPU
$\overrightarrow{\varepsilon}_{GPU}$	GPU	$\left(\sum_{k=1}^{s} E_{C_k}^G, \sum_{k=1}^{s} E_{G_k}^G, \sum_{k=1}^{s} E_{U_k}^G\right)$	energy rate consumed by the stages mapped to the GPU
λ_{GPU}	GPU	μ_{GPU}	relative throughput of the GPU path
$\overrightarrow{E}_{GPU}$	GPU	$p_{GPU} \cdot \overrightarrow{\varepsilon}_{GPU}$	relative energy per item consumed by Q_{GPU}
$\mu_{CPU}^{CG}(n)$	CPU / CG	$\frac{1}{T^{CG}(n)},\ \ n = 0 : nC$	service rate for the stages mapped to the CPU under CG and n threads
$\overrightarrow{\varepsilon}_{CPU}^{CG}(n)$	CPU / CG	$(E_C^{CG}(n), E_G^{CG}(n), E_U^{CG}(n)),\ n = 0 : nC$	energy rate consumed by the stages mapped to the CPU under CG and n threads
μ_{CPU}^{MG}	CPU / MG	$\frac{1}{\sum_{k=1}^{s} T_k^{MG}}$	service rate for the stages mapped to the CPU under MG
$\overrightarrow{\varepsilon}_{CPU}^{MG}$	CPU / MG	$\left(\sum_{k=1}^{s} E_{C_k}^{MG}, \sum_{k=1}^{s} E_{G_k}^{MG}, \sum_{k=1}^{s} E_{U_k}^{MG}\right)$	energy rate consumed by the stages mapped to the CPU under MG
λ_{CPU}^*	CPU / *	μ_{CPU}^*	relative throughput of the CPU path
$\overrightarrow{E}_{CPU}^*$	CPU / *	$p_{CPU}^* \cdot \overrightarrow{\varepsilon}_{CPU}^*$	relative energy per item consumed by Q_{CPU}
λ_e	GPU + CPU	$\lambda_{GPU} + \lambda_{CPU}^*$	effective throughput of the system
E_e	GPU + CPU	$min\left(\frac{TDP}{\lambda_e}, E_{e_C} + E_{e_G} + E_{e_U}\right)$	effective energy per item consumed in the system. See eq. 9

Since our queues are in equilibrium, and we assume maximum utilization on each queue, by applying equation 6 we get $\rho_{GPU} = 1$ and $\rho_{CPU}^* = 1$. From this assumption, we find that the relative throughput for each path is given by the corresponding queue's service rate, that is, $\lambda_{GPU} = \mu_{GPU}$ and $\lambda_{CPU}^* = \mu_{CPU}^*$. Also, the flow balance conditions at equilibrium (equations 3-5) allow us to compute the effective

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

throughput of the system, $\lambda_e = \lambda_{GPU} + \lambda^*_{CPU}$, and the probability that an item goes through the GPU path, $p_{GPU} = \lambda_{GPU}/\lambda_e$, or the probability that it goes through the CPU path, $p^*_{CPU} = \lambda^*_{CPU}/\lambda_e$.

On the other hand, by applying equation 7 we get that the energy utilization of each queue on the corresponding device is proportional to the probability of items serviced on the queue, or in other words, $\rho^{\overrightarrow{E}}_{GPU} = p_{GPU}$ and $\rho^{\overrightarrow{E}^*}_{CPU} = p^*_{CPU}$. This assumption allows us to estimate the relative energy per item consumed by Q_{GPU} in the GPU device, $\overrightarrow{E}_{GPU} = p_{GPU} \cdot \overrightarrow{\varepsilon}_{GPU}$ and by Q_{CPU} in the CPU device, $\overrightarrow{E}^*_{CPU} = p^*_{CPU} \cdot \overrightarrow{\varepsilon}^*_{CPU}$ (for CG or MG granularities), respectively.

The effective energy per item consumed in the system, E_e, can be computed as the minimum of TDP/λ_e and the sum of three components, as defined in equation 8. If the TDP is not reached, then each component can be computed by the energy balance condition that establishes that each component of the effective energy is given by the dominant component of the relative energy computed for each device. In particular, this condition in the DP-* configurations means,

$$\begin{aligned}(E_{e_C}, E_{e_G}, E_{e_U}) &= max\left(\overrightarrow{E}_{GPU}, \overrightarrow{E}^*_{CPU}\right) = \\ &= \left(max(E_{GPU_C}, E^*_{CPU_C}), max(E_{GPU_G}, E^*_{CPU_G}), max(E_{GPU_U}, E^*_{CPU_U})\right) \end{aligned} \tag{9}$$

B. Model for Coupled configurations

These configurations are shown in Figs. 1b and 1d (CP-CG and CP-MG, respectively). Fig. 3b depicts our model for them. In these configurations there is a GPU-CPU path in which a thread processes a item through some stages in the GPU and other stages in a CPU core, and there can also be a CPU path in which other concurrent threads process items through all the stages in the remaining CPU cores. To model the service provided by a sequence of stages mapped to each device on each path, we use a logical queue. Thus, in the GPU-CPU path we can find at least a Q_{GPU} which is the M/M/1 queue that represents the sequence of consecutive stages that service an item in the GPU device, and at least a Q_{CPUA} which is a M/M*/1 queue that represents the rest of stages that service the item in the CPU device (* stands for the granularity studied). For simplicity, in the figure we have represented the case in which the item is first processed by some consecutive stages in the GPU, and later by the rest of stages in the CPU. In case of a mapping where the item is first processed by consecutive stages mapped to the CPU, then to the GPU, then to the CPU, and so on, the model would include more logical queues in the GPU-CPU path: first a Q_{CPUA}, followed by a Q_{GPU}, then another Q_{CPUA}, and so on.

Each Q_{GPU} queue is characterized with two parameters: μ_{GPU}, the service rate due to the consecutive stages mapped to the GPU, and $\overrightarrow{\varepsilon}_{GPU}$, the energy rate consumed by those stages in the GPU device. These parameters are computed as we show in Table IV. For both parameters we just consider the time and

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

the energy per item of the corresponding consecutive stages S_k that are mapped to the GPU ($S_k \in Q_{GPU}$). Also, each Q_{CPUA} queue is characterized with two parameters, depending on the granularity. For the CG granularity, the parameters are: μ^{CG}_{CPUA}, the service rate due to the consecutive stages mapped to the CPU under CG granularity, and $\overrightarrow{\varepsilon}^{CG}_{CPUA}$, the energy rate consumed by those stages in the CPU device under CG granularity. Table IV shows how these parameters are computed, where time and energy come from measurements in Table I. Regarding the MG granularity, the Q_{CPUA} queue is defined by μ^{MG}_{CPUA} and $\overrightarrow{\varepsilon}^{MG}_{CPUA}$. In Table IV we show these parameters, where we notice that time and energy are taken from measurements in Table II.

On the other hand, the stages mapped to the CPU in the CPU path, are modeled with Q_{CPUB} which is a M/M*/1 queue. Again, * stands for the different instantiations of the queue, depending on the granularity. For the CG granularity, the queue is characterized with: $\mu^{CG}_{CPUB}(n)$, the service rate of the CPU under CG granularity, and $\overrightarrow{\varepsilon}^{CG}_{CPUB}(n)$, the energy rate consumed by the queue in the CPU device under CG granularity. With CG, the CPU can run from 0 to nC concurrent threads, in addition to the coupled GPU-CPU thread that serves the GPU-CPU path. Therefore, for CG, the service rate is computed taking into account this additional coupled thread and we model it assuming that the GPU-CPU thread is interfering with the threads that are working concurrently on the CPU. We model this interference by subtracting to the service rate of $n+1$ concurrent threads running in the CPU (because the CPU consists of Q_{CPUA} and Q_{CPUB}), a virtual service rate of 1 thread that is executing in the GPU-CPU path (Q_{CPUA}, the coupled thread). The energy rate is computed for the n concurrent threads working on the queue. In any case, for CG, both the service rate and the energy rate are computed for each number of threads. Table IV shows how these parameters are computed, where time and energy come from measurements in Table I. Regarding the MG granularity, the queue is defined by μ^{MG}_{CPUB} and $\overrightarrow{\varepsilon}^{MG}_{CPUB}$. Under this granularity, all the CPU threads will be serving the Q_{CPUA}. Therefore, we assume that Q_{CPUB} will have a very low probability of serving new items, and so, $\mu^{MG}_{CPUB} = 0$ and $\overrightarrow{\varepsilon}^{MG}_{CPUB} = 0$.

In this configuration, we assume optimistic utilization on each queue. By applying equation 6 we get $\rho_{GPU} \leq 1$, $\rho^*_{CPUA} \leq 1$ and $\rho^*_{CPUB} \leq 1$. From these expressions we find that a solution for the relative throughput for each path is given by, $\lambda_{GPU-CPU} = min(\mu_{GPU}, \mu^*_{CPUA})$ and $\lambda^*_{CPUB} = \mu^*_{CPUB}$. In general, if there were more logical queues in the GPU-CPU path, then a solution for $\lambda_{GPU-CPU}$ could be the minimum of the corresponding service rates in the path. Again, the flow balance conditions at equilibrium (equations 3-5) lead to computing the effective throughput of the system as $\lambda_e = \lambda_{GPU-CPU} + \lambda^*_{CPUB}$, and the probability that an item goes through the GPU-CPU path as $p_{GPU-CPU} = \lambda_{GPU-CPU}/\lambda_e$, or through the CPU path as $p^*_{CPUB} = \lambda^*_{CPUB}/\lambda_e$.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

TABLE IV: Parameters of the CP-* configurations.* stands for CG or MG granularities.

Parameter	Device / Gr.	Value	Description
μ_{GPU}	GPU	$\frac{1}{\sum_{S_k \in Q_{GPU}} T_k^G}$	service rate of stages mapped to Q_{GPU} in the GPU-CPU path
$\overrightarrow{\varepsilon}_{GPU}$	GPU	$\left(\sum_{S_k \in Q_{GPU}} E_{C_k}^G, \sum_{S_k \in Q_{GPU}} E_{G_k}^G, \sum_{S_k \in Q_{GPU}} E_{U_k}^G\right)$	energy rate consumed by stages mapped to Q_{GPU} in the GPU-CPU path
μ_{CPUA}^{CG}	CPU / CG	$\frac{1}{\sum_{S_k \in Q_{CPUA}} T_k^{CG}}$	service rate of stages mapped to Q_{CPUA} in the GPU-CPU path under CG
$\overrightarrow{\varepsilon}_{CPUA}^{CG}$	CPU / CG	$\left(\sum_{S_k \in Q_{CPUA}} E_{C_k}^{CG}, \sum_{S_k \in Q_{CPUA}} E_{G_k}^{CG}, \sum_{S_k \in Q_{CPUA}} E_{U_k}^{CG}\right)$	energy rate consumed by stages mapped to Q_{CPUA} in the GPU-CPU path under CG
μ_{CPUA}^{MG}	CPU / MG	$\frac{1}{\sum_{S_k \in Q_{CPUA}} T_k^{MG}}$	service rate of stages mapped to Q_{CPUA} in the GPU-CPU path under MG
$\overrightarrow{\varepsilon}_{CPUA}^{MG}$	CPU / MG	$\left(\sum_{S_k \in Q_{CPUA}} E_{C_k}^{MG}, \sum_{S_k \in Q_{CPUA}} E_{G_k}^{MG}, \sum_{S_k \in Q_{CPUA}} E_{U_k}^{MG}\right)$	energy rate consumed by stages mapped to Q_{CPUA} in the GPU-CPU path under MG
$\lambda_{GPU-CPU}^*$	GPU-CPU / *	$\min(\mu_{GPU}, \mu_{CPUA}^*)$	relative throughput of the GPU-CPU path
$\overrightarrow{E}_{GPU}$	GPU	$p_{GPU-CPU} \cdot \overrightarrow{\varepsilon}_{GPU}$	relative energy per item consumed by stages mapped to the Q_{GPU} in the GPU-CPU path
$\overrightarrow{E}_{CPUA}^*$	CPU / *	$p_{GPU-CPU} \cdot \overrightarrow{\varepsilon}_{CPUA}^*$	relative energy per item consumed by stages mapped to Q_{CPUA} in the GPU-CPU path
$\mu_{CPUB}^{CG}(n)$	CPU / CG	$\frac{1}{T^{CG}(n+1)} - \frac{1}{T^{CG}(1)},\ n = 0 : nC$	service rate for the stages mapped to Q_{CPUB} in the CPU path under CG and n threads
$\overrightarrow{\varepsilon}_{CPUB}^{CG}(n)$	CPU / CG	$\left(E_C^{CG}(n), E_G^{CG}(n), E_U^{CG}(n)\right), n = 0 : nC$	energy rate consumed by the stages mapped in CPU path under CG and n threads
μ_{CPUB}^{MG}	CPU / MG	0	service rate for the stages mapped to Q_{CPUB} in the CPU path under MG
$\overrightarrow{\varepsilon}_{CPUB}^{MG}$	CPU / MG	0	energy per item rate consumed by the stages mapped in the CPU path under MG
λ_{CPUB}^*	CPU / *	μ_{CPUB}^*	relative throughput of the CPU path
$\overrightarrow{E}_{CPUB}^*$	CPU / *	$p_{CPUB}^* \cdot \overrightarrow{\varepsilon}_{CPUB}^*$	relative energy per item consumed by the stages mapped in the CPU path
λ_e	GPU + CPU	$\lambda_{GPU-CPU} + \lambda_{CPUB}^*$	effective throughput of the system
E_e	GPU + CPU	$min\left(\frac{TDP}{\lambda_e}, E_{e_C} + E_{e_G} + E_{e_U}\right)$	effective energy per item consumed in the system. See eq. 10

Similar to the DP-* configurations, we assume that the energy utilization of each queue on each device is proportional to the probability of items serviced on the corresponding queue, as defined in equation 7. This means $\rho_{GPU}^{\overrightarrow{E}} = p_{GPU-CPU}$, $\rho_{CPUA}^{\overrightarrow{E}^*} = p_{GPU-CPU}$ and $\rho_{CPUB}^{\overrightarrow{E}^*} = p_{CPUB}^*$. These expressions allow us to estimate the relative energy per item consumed on the GPU device, $\overrightarrow{E}_{GPU} = p_{GPU-CPU} \cdot \overrightarrow{\varepsilon}_{GPU}$ and on the CPU device, $\overrightarrow{E}_{CPUA}^* + \overrightarrow{E}_{CPUB}^*$ =$p_{GPU-CPU} \cdot \overrightarrow{\varepsilon}_{CPUA}^* + p_{CPUB}^* \cdot \overrightarrow{\varepsilon}_{CPUB}^*$ (for CG or MG

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

granularities), respectively. As we see, in the CP-* configurations we estimate the relative energy per item consumed in the CPU from the activity in Q_{CPUA} and in Q_{CPUB}. In general, if there were more logical queues in the GPU-CPU path, then all the resultant $\overrightarrow{E}_{GPU}$ for the different Q_{GPU} should be added to estimate the relative energy per item consumed in the GPU device. Similarly, the $\overrightarrow{E}_{CPUA}$ terms should be added to estimate the relative energy per item consumed in the CPU in that path. Finally, as in DP-* configurations, the effective energy consumed in the system, E_e, can be computed as the minimum of TDP/λ_e and the sum of three components (see eq. 8). If the TDP is not reached, then following the energy balance condition we get for CP-* configurations that,

$$(E_{e_C}, E_{e_G}, E_{e_U}) = max\left(\overrightarrow{E}_{GPU}, \overrightarrow{E}^*_{CPUA} + \overrightarrow{E}^*_{CPUB}\right) =$$

$$= \left(max(E_{GPU_C}, E^*_{CPUA_C} + E^*_{CPUB_C}), max(E_{GPU_G}, E^*_{CPUA_G} + E^*_{CPUB_G}), max(E_{GPU_U}, E^*_{CPUA_U} + E^*_{CPUB_U})\right) \quad (10)$$

C. Model extensions

Our model can be extended in two ways. First, so that our throughput and energy estimations also take into account the time and energy due to the data transfers between the CPU and the GPU. Second, to model the alternatives not considered in section II-A. Due to lack of space, these extensions are not described here, but details can be found in [14].

Notice that in the integrated GPUs that we use in our experiments and for our benchmarks, the transfer times are negligible and ignoring them does not affect the accuracy of the model. For discrete GPUs, we expect transfer times to have a higher impact, though.

V. Experimental results

In this section we present our experimental results. Section V-A discusses our evaluation methodology; Section V-B shows the benefit of the analytical model by comparing its performance with a state-of the art baseline approach as well as a study of the overhead due to the training phase and the profit due to the adaptive nature of our framework; Section V-C discusses our experimental results in detail and compares the throughput and energy predicted by the model with the values measured.

A. Evaluation methodology

Two Intel Quad-Core processors have been used in our experiments: a Core i5-3450, 3.1GHz, 77W TDP based on the Ivy Bridge architecture, and a Core i7-4770, 3.4GHz, 84W TDP based on the Haswell one. Both processors feature Advance Vector Extensions (AVX) and have an on-chip GPU, the HD-2500

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

and HD-4600, respectively. Although the Core i7 supports hyperthreading, we found that hyperthreading was not beneficial for our applications, maybe because our benchmarks implementations use the AVX vector units and they fully utilize the computational resources. Thus, only one thread per core was considered for all experiments, and so the upper value for n is 5 threads (4 cores plus 1 GPU). We rely on Intel Performance Counter Monitor (PCM) tool [16] to access the HW counters (energy, clock ticks, L2 and L3 misses, etc). Intel TBB 4.2 provides the core template to implement the pipeline [21]. Inside each pipeline stage, we use Intel OpenCL SDK 2014 for the stages that can be executed on the GPU, or AVX intrinsics for the computations conducted on the cores. For the MG results, we implement nested parallelism on each stage using TBB `parallel_for`. All versions have been compiled using Intel C++ Compiler 14.0 with -O3 optimization flag. We measured time and energy in 10 executions of the applications and compute the average. The reported metrics are throughput, λ, energy per item, E, and as a tradeoff metric, throughput/energy, λ/E. Therefore, λ is the number of frames per second, fps, E stands for the Joules per frame, and λ/E is the fps/Joule.

We validate our framework on Ivy Bridge and Haswell heterogeneous chips using four real applications: ViVid [10], with Low Definition (LD) videos (600×416 pixels) and High Definition (HD) videos (1920×1080 pixels), SRAD [1], Tracking [22] and Scene Recognition [23]. For all the benchmarks and the heterogeneous chips evaluated, the transfer times between GPU and CPU are negligible.

B. Baseline comparison and impact of adaptation

To assess the benefit of using our framework, we compare the pipeline configuration that our model predicts as best with the baseline configuration recommended by a previous work [13] that recommends a configuration based on the intuition that pipeline stages should be mapped to the device where they run more efficiently. This work also recommends exploiting parallelism using an approach similar to software pipelining where two frames are computed at the same time, one on the GPU and another one on the CPU. Therefore only MG granularity is exploited on the CPU cores by this baseline approach.

Table V shows the throughput in terms of frames per second, fps, and throughput/energy, fps/Joule, for homogenous runs, where only the CPU (with MG and CG granularities) or only the GPU is used: "CPU MG", "CPU CG" and "GPU"; and for two heterogenous executions, where CPU and GPU are used: "Baseline" that identifies the results of the aforementioned baseline configuration [13], and "Best" which correspond to the best configuration found by our framework. For both performance metrics, fps and fps/Joule, the higher the value, the better. The "Improv." column shows the percentage of improvement of "Best" with respect to "Baseline" (computed as (Best-Baseline)/Baseline). The last column shows the

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

TABLE V: Comparison of alternatives. For both λ and λ/E the higher the better.

			Homogenous Results			Heterog. Results			
Bench.	Architect.	Metric	CPU MG	CPU CG	GPU	Baseline	Best	Improv.	Best conf.
ViVid LD	Ivy Bridge	λ (fps)	40	62	10	51	65	27%	CP-CG (5)
		λ/E (fps/J)	46	83	8	66	92	40%	CP-CG (5)
	Haswell	λ (fps)	59	47	22	80	91	13%	CP-MG
		λ/E (fps/J)	61	43	24	116	134	15%	CP-MG
ViVid HD	Ivy Bridge	λ (fps)	3.7	3.1	1.1	5.6	5.9	5%	CP-MG
		λ/E (fps/J)	0.3	0.2	0.1	0.6	0.7	15%	CP-MG
	Haswell	λ (fps)	5.4	2.8	2.7	6.5	7.2	10%	CP-MG
		λ/E (fps/J)	0.5	0.1	0.3	0.78	0.9	12%	CP-MG
SRAD	Ivy Bridge	λ (fps)	82	62	72	114	132	16%	DP-MG
		λ/E (fps/J)	212	100	362	403	523	30%	DP-MG
	Haswell	λ (fps)	95	64	93	147	170	15%	DP-MG
		λ/E (fps/J)	182	79	673	499	673	34%	DP-CG (1)
Tracking	Ivy Bridge	λ(fps)	6.2	10	6.8	13	16	23%	CP-CG (4)
		λ/E (fps/J)	1.3	3.2	2.8	4.0	6.7	67%	CP-CG (4)
	Haswell	λ (fps)	6.3	11	9.2	13	19	46%	DP-CG (5)
		λ/E (fps/J)	1.1	2.8	4.0	3.7	8.4	127%	DP-CG (5)

best pipeline configuration and the optimum number of threads, between parenthesis, for the CG cases.

The table shows that the best configuration obtained using our model significantly outperforms the baseline, specially when energy is also considered. These data show that the intuition can result in the selection of a suboptimal configuration, whereas the model can evaluate all configurations and select the best. Also, the baseline only considers "CP-MG"-like mappings, whereas the model considers more alternatives. As the table shows, in 10 out of 16 cases, the best configuration is not CP-MG. In 6 cases (all appear in ViVid) the baseline uses the same mapping as the best (Stage 1 is mapped on the GPU). In these 6 cases, λ and λ/E of baseline and best differ because in the baseline the stages mapped to the GPU can only run in the GPU (Stage 1 only runs in the GPU), while in our implementation the stages mapped to the GPU can also execute on the CPU (Stage 1 runs on both GPU and CPU). Notice that even if we only consider CP-MG mappings, the approach we use as baseline may not find the best mapping of stages to CPU and GPU. This is the case in SRAD and Tracking. For instance, for Tracking the best CP-MG mapping would be to map filters 1 and 3 to the GPU, whereas the baseline approach would map filters 1 and 2 to the GPU. In addition, as the number of possible configurations increases, relying on the intuition to find the best one becomes increasingly difficult.

The table shows that, overall, throughput improvement ranges from 5% to 46% (20% on average), whereas the improvement in throughput/energy ranges from 12% to 127% (43% on average). Energy improvement ranges from 1% to 55% (18% on average). Interestingly, for ViVid on Ivy Bridge, the best

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

pipeline configuration depends on the resolution. CP-CG is the best configuration for LD, while CP-MG is the best for HD. Also, the best configuration can change based on whether the metric to be optimized is λ or λ/E. For instance, SRAD on Haswell obtains maximum throughput with a DP-MG configuration, whereas the maximum throughput/energy is obtained using DP-CG with a single thread. We will discuss each benchmark in more detail in the next subsection.

The previous work that we have considered as a baseline, can not adapt to changes in the input stream. We experimented with changes in the video stream feeding the ViVid application, from Low Definition to High Definition and viceversa. For instance, on Ivy Bridge, when changing from LD to HD, a change in the pipeline configuration from CP-CG to CP-MG results in an improvement of 81% in λ (204% in λ/E). Also, when changing from HD to LD, reconfiguring the pipeline from CP-MG back to CP-CG results in 30% improvement in λ (40% in λ/E). Since the training time for LD is 0.46 sec, and for HD is 6.47 sec, we can determine that the training is amortized (from the throughput point of view) when changes from LD to HD happen at most every 7.8 sec ($\sim$25 HD frames) and when changes from HD to LD happen at most every 0.85sec ($\sim$42 LD frames). More details can be found in [14].

C. Discussion of the results

In this section we validate the accuracy of the model. Figs. 4 to 8 show the results for all applications. In all of them we follow the same convention. On the left of each figure we see the CG evaluation (lines and marks) as the number of threads changes from 1 to 5, as shown on the x-axis. On the right of each figure we have the MG evaluation (three bars and two marks). The homogeneous CPU measurements collected in the training phase are represented by a dashed orange line for the CG execution (see Table I) and by a patterned orange bar for the MG execution (see Table II). Solid lines and bars represent model estimations for heterogeneous runs and marks represent experimental results. For CG predictions we use solid lines: in light-blue for the DP-CG configuration and in dark-brown for the CP-CG one. The square marks are the measurements obtained for both CG mappings: solid for DP-CG and hollow for CP-CG. The solid bars represent the model prediction for MG granularities for heterogeneous runs: in light-blue for the DP-MG configuration and in dark-brown for the CP-MG one. The x marks are the experimental results obtained for the CP-MG configuration whereas the solid triangles are the results for the DP-MG one. By comparing the measurements with the model estimates we can asses the accuracy of the model.

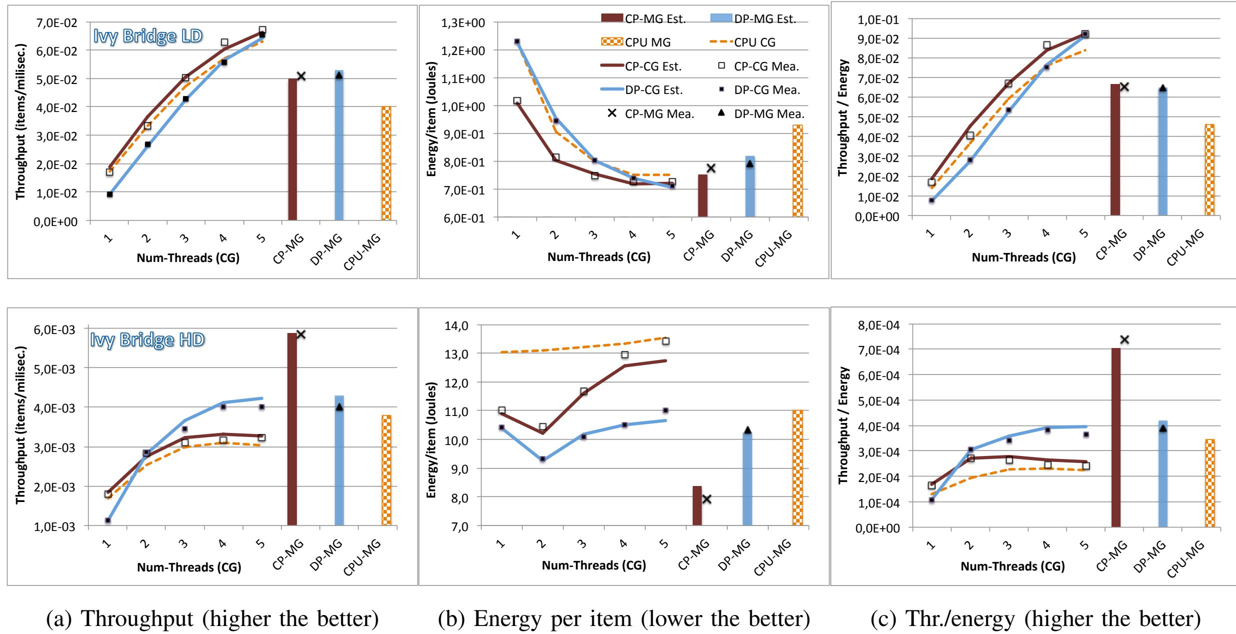
We have experimentally assessed all the evaluated pipeline configurations (48, 384, 48, and 5 for ViVid, SRAD, Tracking, and Scene Recognition, respectively). To facilitate readability, instead of cramming the results of all these experiments on a single chart, *amongst all the possible CP mappings, Figs. 4 to 8 only*

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

show the configuration that achieves the highest λ/E result. As we discuss in the next sections, for all the applications and architectures studied, the estimations of the model reasonably match the measured metrics. For all the cases, the model needs less than 10 microseconds to instantiate the equations for all the possibilities and determine the optimal granularity, mapping and number of threads.

1) ViVid: This application was introduced in section I. ViVid is comprised of 5 stages, being the first and last ones the serial Input and Output, while the three middle ones are parallel. Fig. 4 and Fig. 5 depict the estimated and measured Throughput (λ), Energy per item (E) and Throughput/Energy (λ/E) for LD and HD on Ivy Bridge and Haswell, respectively. Amongst all the CP mappings we only show the most performing one: when stage S_1 is the only one mapped on the GPU (as illustrated in Fig. 1 b) and d)), both for the CG and MG granularities.



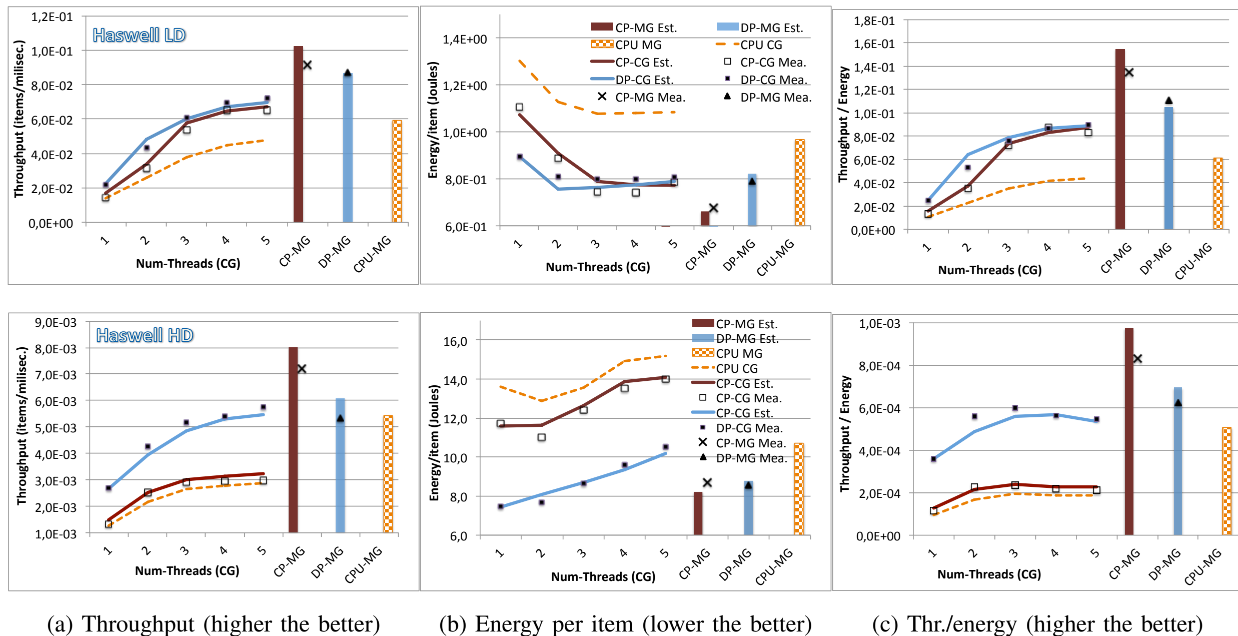
(a) Throughput (higher the better) (b) Energy per item (lower the better) (c) Thr./energy (higher the better)

Fig. 4: Performance metrics for ViVid when processing LD (up) or HD (bottom) video on Ivy Bridge. Solid lines/bars represent model predictions. Marks are the experimental results.

As Figs. 4 and 5 show, our model is able to give a good estimation of λ, E and λ/E. In general, it tends to slightly overestimate the throughput in the CP configurations, because for CP we always consider the ideal contribution of all the threads without considering the overheads. These overheads account for the synchronization costs of the GPU-CPU threads in the coupled GPU-CPU path, that we do not consider in our equations. The results show that our model fits the measured throughput reasonably well, specially on the Ivy Bridge architecture for which the estimated values are within 2% of the measured ones. On

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems



(a) Throughput (higher the better) (b) Energy per item (lower the better) (c) Thr./energy (higher the better)

Fig. 5: Performance metrics for ViVid when processing LD (up) or HD (bottom) video on Haswell.

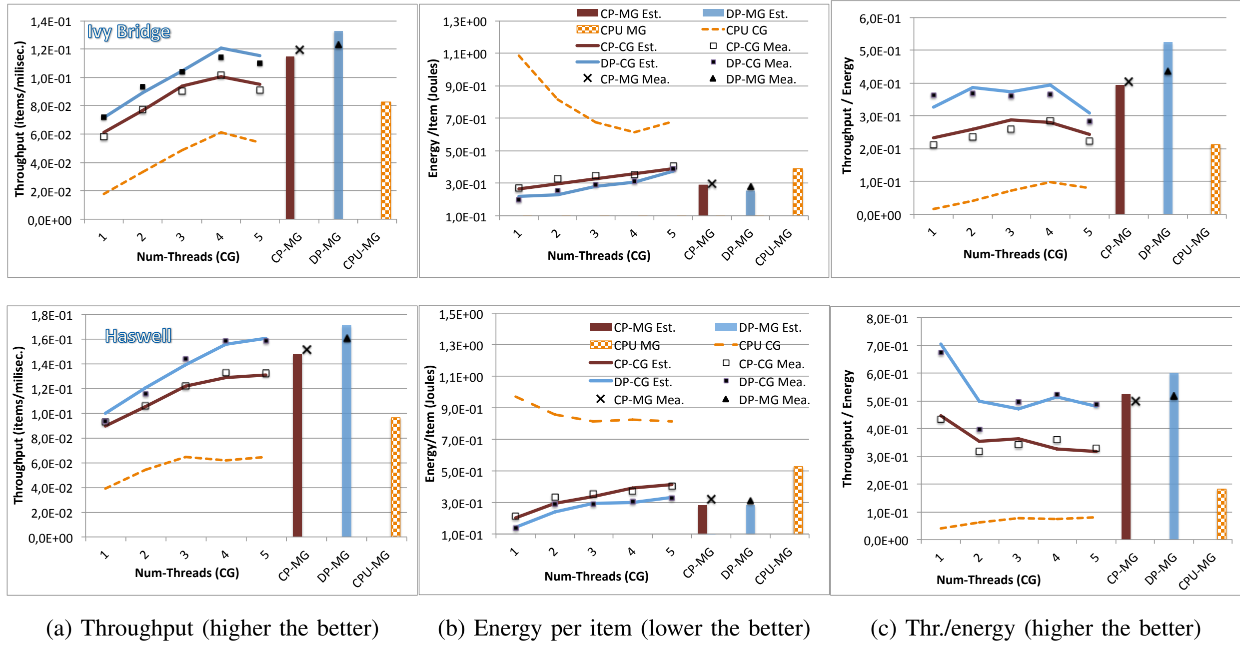
Haswell, our overestimation of the throughput is within 9%. Regarding the energy results, we can also see that, in general, our equations tend to slightly underestimate the energy, although deviation is always within 5% of the measured values. The deviation is more noticeable for Haswell, where our model predicts that CP-MG is better than DM-MG, though measures tell the contrary. Anyway, this imprecision is not significant because the differences between CP-MG and DP-MG are small, so there is not a big penalty to be paid by this error. In any case, the best configuration for energy optimization is DP-CG with 1 thread, that our model correctly predicts. In general, these results validate our initial assumption when deriving the simplified model for the energy consumption, which we introduced in section IV. Also, we can mention that the accuracy of the predicted values for our other metric of interest, Throughput/Energy (λ/E), are within -5% to 10% with respect to the measured values.

Overall, and despite these small inaccuracies, the model successfully predicted the best pipeline configuration (granularity, mapping) as well as the appropriate number of threads, for each type of input and architecture. On Ivy Bridge, for LD videos the optimal is found with the CP-CG configuration and 5 threads (although DP-CG is very close), whereas for the HD input the optimal is provided by the CP-MG configuration. However, on Haswell, the best option for LD and HD is always CP-MG.

The figures also show an important result: a configuration with a higher throughput can consume more energy than a lower throughput one. This can be observed on Haswell, in Figs. 5a and 5b, for the HD

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems



(a) Throughput (higher the better) (b) Energy per item (lower the better) (c) Thr./energy (higher the better)

Fig. 6: λ, E and λ/E for SRAD on Ivy Bridge (up) and Haswell (bottom).

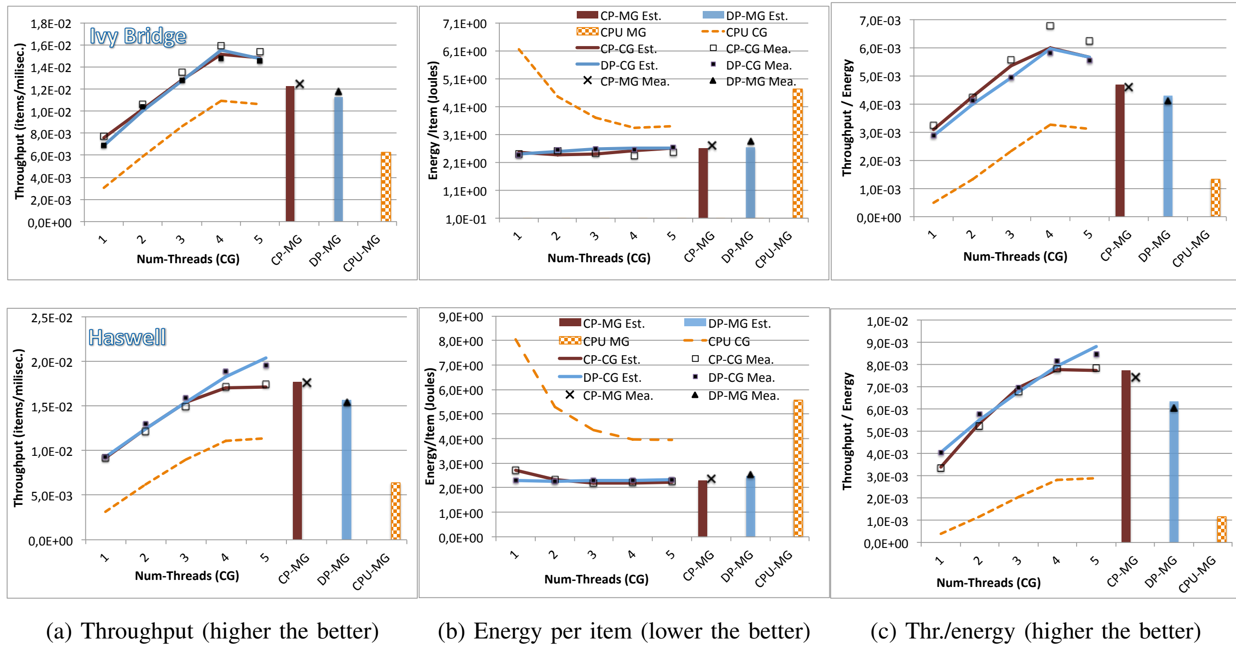
input and the CP-CG configuration, for which the highest throughput is obtained with $n = 5$ threads. However, the energy consumption is also higher for that number of threads. In fact, for this configuration the optimal λ/E for HD is found for $n = 3$, solution that our model correctly predicts as we see in Fig. 5c.

2) SRAD: The SRAD (Speckle Reducing Anisotropic Diffusion) benchmark is part of the Rodinia benchmark suite [1]. This benchmark implements a diffusion method for ultrasonic and radar imaging applications based on partial differential equations (PDEs) [24]. SRAD has 8 pipeline stages: a serial Input and Output, and 6 parallel stages. In our experiments we ran these stages over a stream of images (200). Each stage can implement a CG or MG granularity. Figure 6 shows all metrics for Ivy Bridge and Haswell. From all the CP mappings we only show the most efficient one, that happens to be when the GPU is mapped on all but the second stage, for both the CP-CG and CP-MG configurations, in both machines.

Results in Fig. 6 show that for SRAD our model also provides a reasonable estimation of the throughput and energy on each pipeline alternative. For this application, our equations tend to overestimate the throughput of the DP configurations, especially for the DP-MG configuration, where 7% of deviation over the measured throughput was found. Also, as pointed out for ViVid, a slight underestimation of the energy consumption was registered, in this case always below 8%. These inaccuracies are the reason of

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems



(a) Throughput (higher the better) (b) Energy per item (lower the better) (c) Thr./energy (higher the better)

Fig. 7: λ, E and λ/E for Tracking on Ivy Bridge (up) and Haswell (bottom).

the 16% of overestimation for λ/E in the DP-MG configuration. All in all, our model correctly predicts that the optimal configuration for Ivy Bridge is DP-MG, whereas for Haswell is DP-CG with $n = 1$ if we optimize λ/E. Notice, that DP-CG with $n = 1$ implies that the only thread is the GPU one, which corresponds to an homogeneous execution on the GPU. This is another example of a case in which the highest throughput does not result in the lowest energy. For instance, here we find the maximum λ with DP-CG for $n = 5$ (see Fig. 6a for Haswell). However, since the minimum energy consumption is found for $n = 1$ (Figure 6b), the optimal λ/E is also for DP-CG $n = 1$ (Fig. 6c for Haswell). Our model correctly captures this fact.

3) Tracking: Tracking calculates the movement of a set of features over the image-flow of a video stream. The implementation is based on the Kanade Lucas Tomasi (KLT) [22] algorithm of the San Diego Visual Benchmark Suite [12]. The pipeline of this application has 5-stages: the first and fifth ones are Input and Output, whereas the middle ones are parallel and can be mapped on both CPU and GPU. Each parallel stage can implement a CG or MG granularity. Again, from all the CP mappings we only show the most efficient one: stages 1 and 3 on the GPU for both the CP-CG and CP-MG configurations for Ivy Bridge and Haswell. In the experiments for tracking, we have used a video stream with 200 frames (1080x1920).

Fig. 7 shows the computed and estimated λ, E and λ/E on Ivy Bridge and Haswell. The model's

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

deviation for both platforms is always below 5%, 7% and 11% of measured throughput, energy and throughput/energy, respectively. Again, the model predictions are accurate enough to guess that the appropriate configuration is CP-CG with 4 threads with the GPU used on stages 1 and 3 for Ivy Bridge and DP-CG with 5 threads for Haswell.

4) Scene Recognition: This application performs generic visual categorization, ie., it identifies the object content of natural images while generalizing across variations inherent to the object class (view, imaging, lighting, occlusion, etc). This code is based in the algorithm proposed in [23]. The code is implemented as a 4-stages pipeline. The first and last stages take care of the sequential Input, and Output. The two middle stages are parallel. The input to this code are 200 images of 256×256 pixels from a database containing images from 8 different classes (forest, street, coast, etc). In this benchmark, only the CP-CG configuration is feasible. DP mapping is not an option because the branchy nature of the second parallel stage makes it not suitable for the GPU. Besides, MG granularity does not scale. The first parallel stage can execute on both the CPU and GPU. However, this stage just represents the 18% of the pipeline execution (on both architectures).

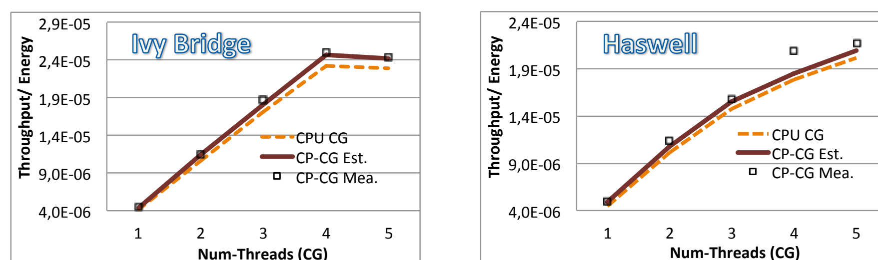


Fig. 8: λ/E for Scene Recognition (higher the better).

Fig. 8 shows the computed and estimated λ/E for the CP-CG configuration when executing the application with 1 to 5 threads on Ivy Bridge and Haswell. Our model accurately predicts the measured values. The higher difference between predicted and measured values is found in Haswell with 4 threads. In this case, λ is underestimated 7% whereas the energy is overestimated 5.5% which turns into 11% of underestimation for λ/E. We can note that the improvement of the CP-CG execution with respect to the homogeneous multicore is small, as we are just affecting the 18% of the application, being the improvement factors (the ratio of the stage's throughput/energy on the CPU vs the GPU) also small. Anyway, one interesting finding is that the Ivy Bridge CP version reaches the point of diminishing returns with 4 threads (although the throughput is slightly higher with 5 threads, the energy is also higher). On Haswell, the optimal solution is for $n = 5$, and our model finds it.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

D. Lessons learned

One relevant result of our model is that it helps us to identify the appropriate granularity for each problem. In the quest of choosing the right granularity for each problem, we have found one important piece of experimental evidence that helps us to understand how the granularities affect performance: the throughput and energy values reported by the multicore homogeneous execution for the two type of granularities are key to predict when one type of granularity will perform better than the other. We can see this result in Table V by comparing the "CPU MG" and "CPU CG" columns of the homogeneous results: if λ or λ/E is larger for "CPU MG" than for "CPU CG", then the recommended granularity for the best configuration (see "Best conf." column in that table) is MG, and viceversa.

In addition to granularity, the mappings also play an important role. DP mappings work well if the GPU thread obtains better values for the metric of interest in all the stages of the pipeline than a CPU thread for CG granularities (or better efficiencies in all stages than nC threads for MG granularities). If this is not the case, then CP can potentially exploit better the heterogeneity of the system, as long as the CP mapping ensures that the pipeline stages are mapped to the device where they execute most efficiently.

Other interesting result is that higher throughput does not always imply a lower energy consumption. This is most noticeable in the CG plots of previous figures, mainly for SRAD and ViVid HD. For example, in Fig. 6, the CG configurations for SRAD have the highest throughput for 4-5 threads, whereas the minimum energy consumption is achieved for one thread.

As summary, we have discussed some of the main trade-offs that affect throughput and energy in the DP and CP mappings under different granularities, and how our reasonable simple model is able to correctly predict all these trade-offs.

VI. Related works

One approach for coding streaming applications is to use a programming language with support for streams, as for example StreamIt [25]. But currently these approaches do not provide support for heterogenous CPU-GPU executions. By using both CPU cores and GPUs, simultaneous computation on heterogenous platforms delivers higher performance than CPU-only or GPU-only executions [26]. However, programming frameworks that provide support for computing in heterogeneous architecture such as Qilin [2], OmpSs [3], XKaapi [4] or StarPU [5] just consider execution time when deciding task distribution among CPU cores and GPU accelerators. The difference between these related works and ours is that they focus on data parallel patterns, while we center on streaming applications.The work by Totoni

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

et al. [13] is perhaps the closest to ours. In this paper, we propose several other pipeline configurations they do not consider. In Section V-B we have used their approach as a baseline and compared with the configuration that our model finds to be the best.

Research on power and energy aware heterogeneous computing started to draw attention several years ago. Most works try to model power or energy specifically for the GPU: some works analytically model GPU power with architecture level instructions [27], [28], or hardware performance events [29], [30]. However, they model the execution of applications in a GPU or a cluster of GPUs, without considering the simultaneous execution on the CPU multicores, which is central in our approach. Those works try to model the power consumption of specific GPU micro-architecture components (such as global memory accesses, texture cache accesses, bank conflicts, etc.) to identify the power bottlenecks in a kernel and suggest power aware optimization strategies. We are concerned in how the different computational resources (CPU cores and GPUs) interact when working in parallel and how to dynamically select energy and performance aware mapping configurations in streaming applications.

There have been other research efforts, such as [31], [32], that have tried to define analytical models to optimize the scheduling of pipeline applications, considering energy and throughput as an objetive or a constraint of the problem. However, these works focus on optimizing the concurrent execution of multi-programed workloads that consist of independent pipeline applications, whereas we are interested in optimizing single streaming execution. In addition, they model energy as a sum of system level components (processor, network, disk, ...) where the energy consumed on each component is the product of the execution time in that component and the dynamic power in the component (measured or estimated using microbenchmarks and supposed constant for the benchmarks evaluated). Our approach, on the other hand, uses the accurate hardware energy counters available on the architectures we study, that allow us to measure at runtime the consumption on the CPU, GPU and Uncore components for the specific application. In contrast with previous static approaches, we use this information to guide the scheduler at runtime to find the optimal granularity, mapping and number of threads that optimize the throughput or the energy (or a trade-off metric) of our application.

VII. Conclusions

To the best of our knowledge, this is the first work proposing an analytical model that can be used to efficiently map the different stages of a pipeline application onto an heterogeneous chip (integrated CPU-GPU processor). The model can use throughput, energy, or a tradeoff such as throughput/energy to predict the best pipeline setting. The model was validated with four applications, finding that the accuracy

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

of our estimations are within 2% to 16%, that suffices to find out the optimal pipeline configuration.

We have also compared the best configuration predicted by the model with a state of the art approach. Our results show that the configurations selected by the model produce, on the average, 20% higher λ and 43% higher λ/E. We have measured improvements in λ and λ/E of up-to 82% and 204%, respectively, when the model is used to adapt to an input video that changes its resolution. Our framework guarantees that the runtime overhead due to the training required to adapt to a changing input is always kept below a user-defined limit.

REFERENCES

[1] S. Che *et al.*, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IISWC*, 2010, pp. 1–11.
[2] C.-K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *MICRO 42*. ACM, 2009, pp. 45–55.
[3] J. Planas *et al.*, "Self-adaptive OmpSs tasks in heterogeneous environments," in *Proc. of IPDPS*, 2013.
[4] T. Gautier, J. Lima, N. Maillard, and B. Raffin, "XKaapi: A runtime system for data-flow task programming on heterogeneous architectures," in *IPDPS 2013*, 2013, pp. 1299–1308.
[5] C. Augonnet *et al.*, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, Feb. 2011.
[6] J. Cong and B. Yuan, "Energy-efficient scheduling on heterogeneous multi-core architectures," in *ACM/IEEE Intl. Symp. on low power electronics and design*, 2012, pp. 345–350.
[7] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "Mevbench: A mobile computer vision benchmarking suite," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, Nov 2011, pp. 91–102.
[8] L. De Cicco, S. Mascolo, and V. Palmisano, "Skype video responsiveness to bandwidth variations," in *18th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2008, pp. 81–86.
[9] J. Fernandez-Madrigal, E. Cruz-Martin, A. Cruz-Martin, J. Gonzalez, and C. Galindo, "Adaptable web interfaces for networked robots," in *Intelligent Robots and Systems, (IROS 2005)*, Aug 2005, pp. 3441–3446.
[10] M. Dikmen, D. Hoiem, and T. S. Huang, "A data driven method for feature transformation," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3314–3321.
[11] M. Jones and P. Viola, "Fast multi-view face detection," *Mitsubishi Electric Research Lab TR-20003-96*, vol. 3, 2003.
[12] S. K. Venkata and et al., "SD-VBS: The San Diego Vision Benchmark Suite," in *IISWC*, 2009, pp. 55–64.
[13] E. Totoni *et al.*, "Easy, fast and energy efficient object detection on heterogeneous on-chip architectures," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, 2013.
[14] A. R. et al., "Productive interface to map streaming applications on heterogeneous processors," Comput. Architecture Dept., Tech. Rep., 2015, http://www.ac.uma.es/˜asenjo/research/.
[15] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *ISPLED*, 2010.
[16] R. Dementiev, T. Willhalm, O. Bruggeman, P. Fay, P. Ungerer, A. Ott, P. Lu, J. Harris, P. Kerly, and P. Konsor, *Intel Performance Counter Monitor*, 2012. [Online]. Available: www.intel.com/software/pcm
[17] H. McCraw, J. Ralph, A. Danalis, and J. Dongarra, "Power monitoring with PAPI for extreme scale architectures and dataflow-based programming models," in *Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications, IEEE Cluster 2014*, sep 2014.
[18] W. J. Gordon and G. F. Newell, "Closed queuing systems with exponential servers," *Operations Research*, vol. 15, no. 2, pp. 254–265, 1967.
[19] A. Navarro, R. Asenjo, S. Tabik, and C. Cascaval, "Analytical modeling of pipeline parallelism," in *Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on*, Sept 2009, pp. 281–290.
[20] S. K. Bose, "Open and closed networks of M/M/m type queues," Indian Inst. of Technology Guwahati, Tech. Rep., 2002.
[21] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*. O'Reilly, 2007.
[22] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI*, 1981, pp. 674–679.
[23] G. Csurka *et al.*, "Visual categorization with bags of keypoints," in *ECCV*, 2004, pp. 1–22.
[24] Y. Yu and S. Acton, "Speckle reducing anisotropic diffusion," *Image Processing, IEEE Tran. on*, vol. 11, no. 11, pp. 1260–1270, 2002.
[25] R. Soule, M. I. Gordon, S. Amarasinghe, R. Grimm, and M. Hirzel, "Dynamic expressivity with static optimization for streaming languages," in *Intl Conf on Distributed Event-Based Systems*, Jun. 2013.
[26] C. Yang *et al.*, "Adaptive optimization for petascale heterogeneous CPU/GPU computing," in *CLUSTER*, 2010, pp. 19–28.
[27] J. Pool, A. Lastra, and M. Singh, "An energy model for graphics processing units," in *ICCD*, 2010, pp. 409–416.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2432809, IEEE Transactions on Parallel and Distributed Systems

[28] J. Leng *et al.*, "GPUWattch: Enabling energy optimizations in GPGPUs," in *ISCA*, 2013, pp. 487–498.
[29] K. Kasichayanula *et al.*, "Power aware computing on GPUs," in *SAAHPC*, 2012, pp. 64–73.
[30] S. Song, C. Su, B. Rountree, and K. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *IPDPS*, 2013, pp. 673–686.
[31] A. Benoit, P. Renaud-Goud, and Y. Robert, "Performance and energy optimization of concurrent pipelined applications," in *IPDPS*, 2010, pp. 1–12.
[32] C.-S. Lin *et al.*, "Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems," *Journal of Systems Architecture*, vol. 59, no. 10, Part C, pp. 1083 – 1094, 2013.

Antonio Vilches earned a M.S. in Computer Sciences with Honors in 2012 from the University of Málaga, Spain. Later, he spent 1 year working in industry, where he was optimizing on demand web apps with millions of requests per minute. He is currently working towards his PhD at the department of Computer Architecture in the University of Málaga as well. His research interests are in heterogeneous systems, parallel computing, runtime optimizations and machine learning methods.

Angeles Navarro received the engineering degree in telecommunications in 1995 and the PhD degree in computer science in 2000, both from the University of Málaga, Spain. She is an associate professor in the Computer Architecture Department at University of Málaga since 2001. She lectures on computer organization and architecture. She has served as a program committee member for PPoPP'11, ICPP'11, ISPA'12, IPDPS'13, IPDPS'14, PACT'14, ICS'14. She is member of the committee of experts of the European Commission in the field of "Embedded Systems and High Performance Computing". Her research interests are in programming models for heterogenous systems, analytical modeling, compiler and runtime optimizations.

Rafael Asenjo received the engineering degree in telecommunications in 1993 and the PhD degree in telecommunication engineering in 1997, both from the University of Málaga, Spain. From 1994 to 2001, he was an assistant professor in the Computer Architecture Department at University of Málaga, and has been an associate professor in the same department since 2001. He is serving as General Chair for PPoPP'16. He has also served as a PC member SBAC-PAD'12, IPDPS'13, IPDPS'14 and SC'15. His research interests are in programming models, parallel programming, heterogeneous architectures, parallelizing compilers and multiprocessor architecture. He is ACM Member.

Francisco Corbera received the BS and MS degrees in computer science in 1994, from the University of Granada, Spain, and the PhD degree in computer science in 2001, from the University of Málaga, Spain. From 1996 to 2001, he was an assistant professor in the Computer Architecture Department at University of Málaga, and has been an associate professor in the same department since 2002. He lectures on computer technology and architecture. His research interests are in parallelizing compilers and multiprocessor architectures.

Rubén Gran graduated in Computer Science from the University of Zaragoza (Spain) an hold his PhD in 2010 from the Polytechnic University of Catalonia (UPC, Spain). Since then, he is assistant professor in the Department of Computer Science and Systems Engineering at the University of Zaragoza. His research interests are hard real-time systems, hardware for reducing worst-case execution time and energy consumption, microarchitecture and effective programming for parallel and heterogeneous systems.

María J. Garzarán received in 1993 a B.S. degree in computer science from Universidad Politecnica de Valencia, Spain and in 2002 a Ph.D. degree from Universidad de Zaragoza, Spain. She is also the recipient of the 2002 Best PhD Thesis Award from Universidad de Zaragoza. She is a Research Associate Professor at the Computer Science Department of the University of Illinois at Urbana-Champaign. She was Program Co-Chair of the 2007 Workshop on Languages and Compilers for Parallel Computing (LCPC) and of the Programming Systems track of Supercomputing 2014. Her research focuses on thread-level speculation, automatic performance tuning, parallel programming, compilation, and reliability. She is IEEE Member and Senior Member of ACM.

1045-9219 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.