Evaluating Orthogonality between Application Auto-Tuning and Run-Time Resource Management for Adaptive OpenCL Applications

Edoardo Paone Davide Gadioli Gianluca Palermo Vittorio Zaccaria Cristina Silvano
Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria
Email: {name.surname}@polimi.it

***Abstract*—The ever increasing number of processing units integrated on the same many-core chip delivers computational power that can exceed the performance requirements of a single application. The number of chips (and related power consumption) can thus be reduced to serve multiple applications — a practice which is called *resource consolidation*. However, this solution requires techniques to partition and assign resources among the applications and to manage unpredictable dynamic workloads.**

To provide the performance requirements in such scenarios, we exploit application auto-tuning, based on design-time analysis, of both application-specific *dynamic knobs* and computational parallelism. Such features are implemented in a software library, which is used to demonstrate the main contribution of this paper: a light-weight Run-Time Resource Management — RTRM — technique to improve resource sharing for computationally intensive OpenCL applications.

We evaluate how much the interaction between RTRM and application auto-tuning can become synergistic yet orthogonal. In the proposed approach, run-time adaptation decisions are taken by each application, autonomously. This has two main advantages: i) a non-invasive application design, in terms of source code, and ii) a very low run-time overhead, since it does not require any central coordination of a supervisor nor communication between the applications.

We carried out an experimental campaign by using a video processing application — an OpenCL stereo-matching implementation — and stressing out resource usage. We proved that, while RTRM is necessary to provide lower variance of the application performance, the application auto-tuning layer is fundamental to trade it off with respect to the computation accuracy.

I. Introduction

A new trend in programming data-parallel computationally intensive applications is using OpenCL not only for programming heterogeneous platforms (by exploiting GPU or FPGA accelerators) but also for homogeneous platforms. OpenCL follows an "offload" programming model, where an accelerator is accessed via the host system and is programmed as a co-processor, to speedup the execution of computationally intensive kernels. OpenCL API [1] is designed to make efficient use of the massive computational parallelism provided by modern accelerators. On the contrary, there is not yet support for efficient deployment of multiple OpenCL applications on the same platform. However, the increasing number of processing units integrated on the same chip delivers computational capabilities that can exceed the performance requirements of a single application. Thus, server consolidation is a common approach to reduce the number of machines (and therefore the power consumption) needed to provide some services. In this area, runtime adaptability represents a key technique for computing systems to adjust their behaviour with respect to operating environments, usage contexts, resource availability and even to faults, thus enabling close-to-optimal operation in the face of changing conditions.

In this work, we address the problem of resource sharing in server consolidation for adaptive and computationally intensive OpenCL applications. Our target application scenario is characterized by unpredictable, variable workloads, where applications have to serve concurrent requests and provide a best-effort service to the users. In this context, we are interested in evaluating the combination of: *i)* application auto-tuning and *ii)* Run-Time Resource Management (RTRM) techniques to improve resource sharing among computationally intensive workloads. The lack of auto-tuning and run-time adaptation capabilities at the application-level leads to sub-optimal power/performance trade-offs at the system level given by the underutilization of system resources. On one side, the auto-tuning mechanism allows to trade off performance and Quality-of-Result metrics directly acting on application-level knobs; on the other side, runtime system management needs to optimize the computing capabilities with respect to dynamic variations of the environment conditions, computational demands and resource availability.

We implemented an auto-tuning framework – the Application-Specific Run-Time Manager (AS-RTM) – which exploits design-time and run-time concepts to create an alternative and effective way of "self-aware" computing. Our framework aims at determining the relation between application parameters and performance metrics (such as throughput, accuracy) at design-time through an exploration phase based on code profiling. This information is used to create a model of the application behavior, characterizing the effect of the tunable parameters (also known as dynamic knobs [2]) on performance. Then, this model supports run-time decisions to tune the application behavior according to available resources and given constraints. Among the application knobs, we have included a parameter which controls, on a multi-core platform, the CPU quota used by the application. To this end, we exploit the *device fission* OpenCL API [1] to deploy OpenCL kernels on selected processing units of a multi-core CPU. The drawback of this API is that OpenCL dynamic compilation

This work was supported in part by the EC under the grants HARPA FP7-612069 and CONTREX FP7-611146.

is required each time the computing device is reconfigured. Thus, our analysis also takes into account the benefits of asynchronous compilation to reduce the reconfiguration overhead. Additionally, this paper introduces an innovative light-weight technique – called *resource-aware AS-RTM* – where the AS-RTM is enabled to take autonomous decisions on resource utilization. The *resource-aware AS-RTM* considers the information of system workload gathered by platform sensing for taking reconfiguration decisions, while minimizing the impact on other applications that share the same resources. However, differently from previous approaches (e.g. "invade and retreat" [3]), applications act as autonomous agents, without coordination among them. On the one hand, this solution has the advantage of being non-intrusive from a design point of view, since it does not require a communication infrastructure; on the other hand, it does not provide any guarantee of *fairness* nor *optimality* in resource allocation. To achieve system-level objectives such as *fairness*, we include in the experimental setup a configuration (called *Adaptive-RTRM*) which exploits a two-level run-time management: resource allocation delegated to a centralized resource manager and application-specific parameters controlled by the AS-RTM.

To summarize, the main contributions of this paper can be summarized as follows:

- We introduce an infrastructure that provides run-time support for application adaptivity, based on design-time analysis.
- We analyze the problem of resource allocation for computationally intensive OpenCL applications on multi-core OpenCL platforms.
- We evaluate different solutions for run-time management based on our auto-tuning framework, to exploit the orthogonality between application-specific knobs and resource allocation.
- We propose a light-weight technique for run-time resource management based on platform sensing at application level.

The remainder of this paper is organized as follows. Section II provides an overview of the related work, while Section III introduces the target architecture for the adaptive framework, explaining the reasons behind this solution. Use-cases and definitions are described in Section IV, while the experimental results are presented in Section V. Finally, Section VI concludes the paper.

II. Related Work

In [4], the OpenCL standard is extended to support computation offloading in the automotive industry, by exploiting IP-based in-car networks. However, computation offloading introduces the problem of resource sharing in server consolidation [5], thus it requires Run-Time Management (RTM) techniques.

The adaptive control technique proposed in [6], called *online architecture tailoring*, is based on control theory and provides for continuous self-adaptation of the application. However, this technique is demonstrated for a single application while we target more complex workloads that would require continuous adaptation at system level.

A distributed RTM approach for homogeneous many-core systems based on game theory is presented in [3]. While distributed approaches suffer from communication overhead and convergence time, centralized solutions [7] require heuristics for optimal resource allocation within a short decision time.

In [8], the authors combine design-time and run-time techniques in order to train a global resource manager. A step forward made on top of the previous approach has been done in [9] with a run-time management framework, called ARTE, supported by DSE. Even in this case, the run-time manager is a single one (system-wide) and, at the application level, it provides only the possibility to change the parallelization of the application.

More effective application-level tuning is leveraged by software-based approximate computing, a set of techniques to design applications to trade off accuracy with respect to increased performance. An example is given by the Speed-Press compiler [10], which allows to apply automatic *code perforation* to insert *dynamic knobs* [11] into the application code. Such knobs are application parameters that modify the computation, for example by changing the step in loop iterations, and impact on performance and approximation of the application output.

The work presented in [2] combines the concept of *dynamic knobs*, to support run-time adaptivity, and that of *heartbeat*, to provide a way to monitor application latency and throughput. Their framework, SEEC, uses a run-time manager with different levels of adaptation, from a simple closed-loop control scheme to a more refined machine learning manager. However, this solution lacks of design-time techniques support.

III. Target Adaptive Framework

The basic idea of the proposed adaptive framework consists of exploiting the orthogonality between application auto-tuning and runtime resource management for compute-intensive OpenCL applications. To this purpose, we have envisioned a twofold approach. On one side an application-oriented self-adaptive layer uses some knobs at the application level to trade off performance with Quality-of-Result metrics. On the other side, system resources are partitioned and assigned to the running applications by the resource management layer. The runtime decisions are taken based on profiling information gathered at design-time, which consists of a set of application configurations, namely the Operating Points (OPs) [12], which are optimal with respect to application performance metrics (e.g. throughput) and resource usage.

In this section, we first highlight the application-oriented self-adaptive layer (Section III-A), then the runtime resource management layer (Section III-B) and finally the proposed resource-aware extension to the Application-Specific Run-Time Manager (Section III-C).

A. Application adaptivity through dynamic knobs

In our approach, each application is linked to the framework library that provides an Application-Specific Run-Time Manager (the AS-RTM). The main purpose of the AS-RTM is to manage application adaptivity by tuning the *dynamic knobs* [11] – application parameters that can be changed at run-time without recompiling the application.

The AS-RTM is generic, however its behavior can be customized for each application given a different list of run-time configurations – the Operating Points (OPs). The OP data structure contains a set of parameters, which represent the values of the application *dynamic knobs* for a specific

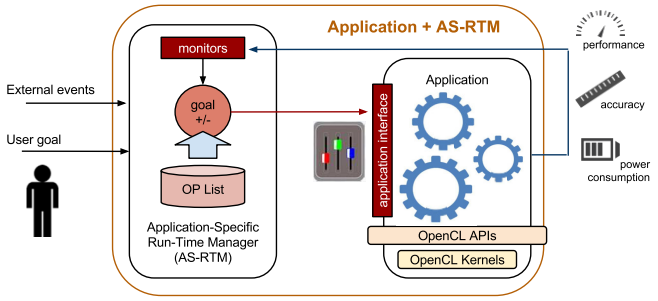


Fig. 1: Application adaptivity through the AS-RTM.

configuration, and the metric values profiled at design-time. As shown in Figure 1, the AS-RTM allows to define one or more application goals. A goal represents a soft-constraint on a certain metric (such as the frame-rate or the Quality-of-Service – QoS) or parameter (such as the number of threads) that can be dynamically set by the user or selected by the application itself depending on external events. Such constraints are assumed to be strictly ordered by their priority.

Similarly to the *Heartbeat* framework [13], our AS-RTM uses high-level monitors of the performance (such as a throughput monitor, a QoS monitor or a user-defined monitor) to sense the execution context and to react to any change in the application requirements.

The AS-RTM integration in a third-party application is straightforward: it does not require refactoring the application logic to meet an execution template (such as in [7] and [8]) but only wrapping the profiled code with the monitor calls.

B. Run-Time Resource Management

The proposed approach aims at analyzing run-time resource management for compute-intensive workloads, such as OpenCL applications, in multi-application scenarios.

In a plain OpenCL application, the platform resources are managed by the OpenCL run-time at application-level, so an application is enabled to use all devices available on an OpenCL platform and, by default, the entire quota of each device. On a multi-core CPU, for example, the OpenCL run-time binds each application to all compute units by default and relies on the OS scheduler to assign CPU time to all applications (seen as different processes by the scheduler). The drawback of this approach is that application performance is not predictable, as the number of deployed applications (processes) changes over time. Moreover, it is not possible to control the amount of resource quota for each application: while the OS scheduler is fair in allocating user time to processes, this is unfair from a point of view of the application, because applications might have different resource requirements, constraints or priority.

To tackle the problem of resource sharing on multi-core platforms, we propose a distributed approach for run-time resource management (RTRM), by extending the AS-RTM described in Section III-A with a monitor for sensing system CPU usage. Since the so called *Resource-Aware AS-RTM* is enabled to take local decisions on resource allocation in multi-/many-core platforms, our solution falls into the research area of *invasive computing* [14]. However, our approach does not need communication between applications, thus each AS-RTM is completely autonomous and has only partial information about the platform state.

C. Proposed Resource-Aware AS-RTM

Our proposed methodology aims at extending the domain of application knobs, to manage resource-related parameters without the need of interacting with other actors. In this way it is possible to avoid any synchronization/communication delay and increase the portability of the application to a new platform, without any additional effort, but generating the OPs specific for that platform.

The AS-RTM introduced in Section III-A allows the user to set a soft-constraint on the application parameters or metrics in the OP configuration. The performance metrics in the OP list are obtained by profiling the application in isolation on the target platform. Thus, for computationally intensive applications, the performance metrics – such as throughput – are likely to benefit from increased computational parallelism, thus higher resource usage. On the contrary, our target scenario consists of multiple applications, with different performance and resource requirements, deployed on the same platform.

It is possible to treat any resource-related parameter (such as the computational parallelism) as a generic application-specific parameter. However, a plain management of such parameters could lead to system configurations where the total amount of computational parallelism required by the running applications exceeds the system resources. In turns, this would result in a degradation of application performance since the OS scheduler limits the process CPU usage.

To overcome this problem, we propose a resource-aware AS-RTM, which takes into account the CPU usage (as we target multi-core CPU platforms), for self-limiting the application parallelism (e.g. the number of working threads). At each application cycle, the AS-RTM monitors the goal status. Whenever a goal is not satisfied or has been changed (e.g. the required frame-rate has been decreased), in order to select the most suitable OP, the AS-RTM follows this procedure:

1) The AS-RTM updates the internal constraints on the OPs to meet the current goals.
2) If there is at least one OP that satisfies all the constraints, the AS-RTM chooses the one with the highest rank value. Otherwise, it chooses the OP closest to the valid region defined by the constraints, starting from the one with highest priority.

According to this decision policy, we add a constraint on the process CPU usage, on top of the application-specific ones. The value of this constraint is initialized to the maximum system CPU quota (Γ). At run-time, by monitoring the system CPU usage (γ) and the process CPU usage ($\pi_{measured}$), the AS-RTM selects an OP only if the profiled CPU usage of the OP ($\pi_{profiled}$) satisfies the following constraint:

$$\pi_{profiled} \leq \Gamma - \gamma + \pi_{measured} \quad (1)$$

If only one application is running, γ and $\pi_{measured}$ are equal, thus the application is allowed to use the entire CPU resource. Otherwise, if the platform is congested, Γ and γ have the same value, which forces the AS-RTM to select among the OPs whose profiled CPU usage fits the quota assigned by the OS scheduler.

In the experimental results, our solution is compared to a centralized approach, where resource allocation is delegated

and coordinated by a system-wide run-time resource manager (SW-RTRM), while the AS-RTM takes local decisions only on application-specific parameters. We will show that it is possible to reach the same average performance predictability with our framework, at the expenses of no guarantee on *fairness* nor *optimal* resource allocation.

IV. Experimental Setup

A. Use-case application

We consider a case study based on the Stereo-Matching application [15], implemented with OpenCL APIs [1] and designed to export a set of parameters which impact on both application-specific and platform metrics. Stereo-Matching belongs to a class of applications that allow to dynamically tune at run-time the trade-off between performance and Quality-of-Result (QoR) metrics, by means of *dynamic knobs* [11]. An example of such knobs is given by those parameters that change the number of loop cycles, by skipping some iterations with a given step (a technique known as *loop perforation* [10]). This generates a more approximate result, which we also refer to as *QoR loss*, but a faster application execution.

B. Definition of metrics

The Stereo-Matching application has two metrics of interest, namely the *frame-rate* (measured as [frames/s]) and the *disparity error*, which represents a measure of the average error associated to the application result (the pixel disparity [15]). However, in our tests we consider only normalized metrics that abstract from the specific application, defined as follows.

1) Normalized Actual Penalty (NAP): This metric measures the degree of user satisfaction, with respect to a frame-rate goal set at the application start. The frame-rate goal is a soft real-time constraint, which should be met independently from the machine workload and resource availability.

$$NAP = \frac{GOAL_{measured} - GOAL_{demanded}}{GOAL_{measured} + GOAL_{demanded}} \tag{2}$$

2) Normalized Error: This is a measure of the output quality normalized on the range of valid values, so that $ERR = 1$ when the application runs with the configuration that provides the lowest – but still acceptable, with respect to design requirements – output quality; while $ERR = 0$ when the quality is highest. It was obtained for Stereo-Matching from the *disparity error* (DErr) as follows:

$$ERR = \frac{DErr_{OP} - DErr_{MIN}}{DErr_{MAX} - DErr_{MIN}} \tag{3}$$

3) Difference w.r.t. to off-line profiling: Another metric of interest is the *deviation* (DEV) of the metrics (e.g. cycle period) observed at run-time with respect to the expected values, i.e. the OP metrics profiled at design-time.

$$DEV = \left| \frac{Tcycle_{measured}}{Tcycle_{OP}} - 1 \right| \tag{4}$$

Since our tests consider dynamic scenarios, for the NAP and ERR metrics we compute a synthetic value that takes into account the temporal dimension:

$$NAP_{AVG} = \frac{\int NAP(t)\, dt}{\Delta t}\ ,\ ERR_{AVG} = \frac{\int ERR(t)\, dt}{\Delta t} \tag{5}$$

C. Definition of dynamic workload

A dynamic workload, in this paper, consists of a set of applications with different schedules (start time), amount of data to process (number of frames in Stereo-Matching) and performance requirements (frame-rate). This use-case is aimed at mimicking the workload expected in server consolidation, specifically targeted to offloading computationally intensive OpenCL applications [4]. Although we use only one type of application (Stereo-Matching), we mimic a dynamic workload by exposing the following parameters:

- *Start delay*: each application instance is started upon user request, thus we use different start times.
- *Amount of input data*: each Stereo-Matching instance is required to process a different number of frames.
- *Frame-rate goal*: soft real-time constraint to guarantee a certain response time, as demanded by the user.

The above parameters are randomly chosen for each Stereo-Matching run, within a range of values shown in Table I.

D. Platform description

We ran our experiments on two multi-core platforms:

1) AMD platform: NUMA machine with four nodes, each a Quad-Core AMD Opteron Processor 8378 at 2.4 GHz, with 8 GB of RAM per node, running a Linux distribution based on kernel 3.9. OpenCL 1.2 run-time provided by AMD OpenCL SDK v2.8.1.

2) Intel platform: Workstation with Intel Xeon Quad-Core CPU E5-1607 at 3.0 GHz and 8 GB RAM, running a Linux distribution based on kernel 3.5. OpenCL 1.2 run-time provided by Intel OpenCL SDK 2013.

The device fission API can be used to partition a multi-core CPU device into sub-devices. The API defines several partition schemes. We use a partitioning by count [1] to create one sub-device of specific size, in this way controlling the CPU resource usage. However, the OpenCL program is bound to a context, so every time a new sub-device is selected it is necessary to create a new context and rebuild the OpenCL program. The OpenCL program build introduces overhead at run-time, which might limit the benefits of application auto-tuning if the reconfiguration rate is high. To reduce this reconfiguration overhead, we used asynchronous dynamic compilation, a feature of the `clProgramBuild` OpenCL API [1] supported by some OpenCL run-time implementations (e.g. Intel). By passing to `clProgramBuild` a function pointer to a notification routine, this routine is called when the program has been built, while the application can continue running in the previous configuration. Our measurements with Intel OpenCL SDK show that synchronous dynamic build of the Stereo-Matching kernels takes $624ms$ on average, while the reconfiguration overhead of asynchronous build is only $58ms$ (10x less).

TABLE I: Range of values for the random parameters of dynamic workload tests.

Parameter	AMD	Intel
Number of frames	10-840	
Frame-rate goal [frames/s]	1-7	
Start delay [s]	0-90	
Num. instances	1-6	1-4

E. Run-Time Management description

Five Run-Time Management configurations have been considered:

1) Plain-Linux: Baseline implementation without SW-RTRM and AS-RTM. Each application instance is deployed as a plain OpenCL application, thus it is binded by default to all processing elements available on the CPU. On the one hand, since there is no SW-RTRM, this configuration relies on the OS to schedule tasks from different applications. On the other hand, the application runs a fixed configuration, with 50% QoS.

2) Plain-RTRM: For this test we use an open source resource manager, the BarbequeRTRM [7], to allocate resources to the running applications. In BarbequeRTRM, application requirements are mainly defined by the set of Application Working Modes (AWMs), identified at design-time, each one corresponding to a given amount of required resources. However, if the resource requirement gets higher at run-time, an application can also request to the manager a higher AWM, through a specialized API. Any change in the application resource requirements or in the system resource availability generates an event that triggers a system reconfiguration. The SW-RTRM has a complete knowledge of the system state, including dynamic resource requirements of individual applications, which ensures optimal resource allocation w.r.t. system-level objectives – such as fairness, execution priority, reconfiguration overhead and congestion. Also in this case, the AS-RTM is not used, thus the application runs with QoS fixed to 50%.

3) AS-Linux: No SW-RTRM but the proposed AS-RTM is enabled. The AS-RTM can switch the Operating Point to trade-off between performance and QoS. Although the computational parallelism can be controlled by the AS-RTM through an application dynamic knob, the effective resource usage depends on the allocation of CPU user time by the OS scheduler.

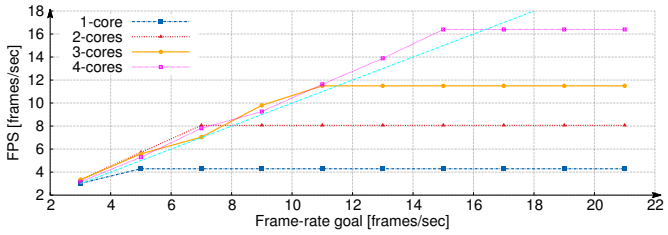
4) RA-AS-Linux: Configuration implementing the proposed Resource-Aware AS-RTM. Differently from the previous configuration, here the computational parallelism is used orthogonally with respect to the application-specific *knobs*. It implements the technique presented in Section III-C, based on monitoring of the system CPU usage for *smart* adaptation of the resource requirement.

5) AS-RTRM: Two-level run-time management, which uses both the centralized resource manager and the proposed AS-RTM. On the one hand, resource allocation is delegated to the BarbequeRTRM, which enforces a fair allocation of platform resources among the running applications; on the other hand, the AS-RTM is used to control at application-level the trade-off between performance and accuracy metrics, by tuning the parameters orthogonal w.r.t. resource-related parameters.

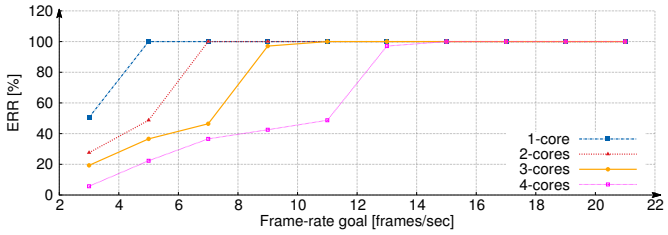
V. Experimental Results

The experiments described in this section have been carried out to assess the benefits of the proposed framework. In Section V-A, a single stereo-matching application, with some constraints on resource usage, has been used to assess the capability of the proposed AS-RTM framework of exploiting the available trade-offs between performance metrics. In Section V-B, we evaluate the orthogonality between the decision space of the AS-RTM, analyzed before, and different RTRM techniques. We consider first an approach where resource utilization is managed as an application parameter in a flat configuration (*AS-Linux*); then, we present the two-level approach based on a centralized resource manager (*AS-RTRM*); finally, we present the proposed technique for efficient resource sharing based on platform sensing (*Resource-Aware AS-Linux*). We conclude this section with a campaign of experiments (Section V-C) with random workloads to compare the different techniques analyzed individually in the previous experiments.

(a) Average measured frame-rate vs. frame-rate goal



(b) Average normalized disparity error vs. frame-rate goal



(c) Average Normalized Actual Penalty vs. frame-rate goal

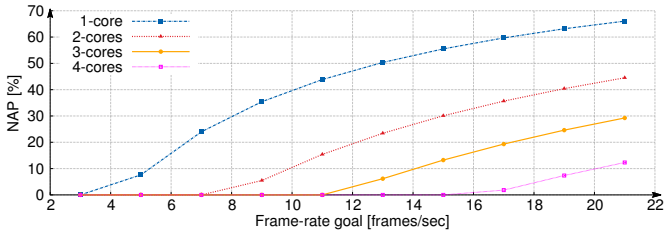
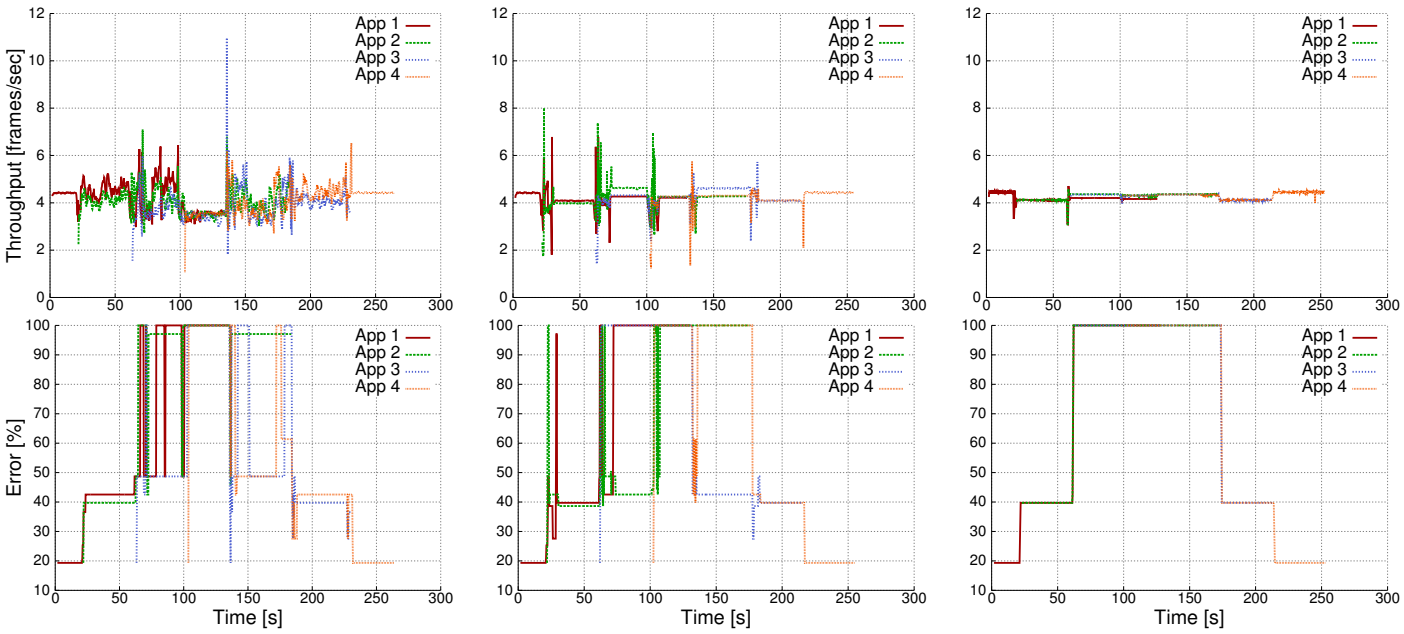


Fig. 2: Observed frame-rate, normalized error and NAP by varying the frame-rate goal and the number of cores.

A. Application Auto-Tuning Results

This experiment aims at assessing the benefits of application adaptivity. It consists of a single Stereo-Matching application deployed on the Intel platform, with 200 frames to process. The test is repeated for each possible number of cores (4 in total on the Intel platform), with the frame-rate goal incremented at each run from 3 to 21 frames/s.

The results are shown in the three plots of Figure 2, where the x-axis is the goal value and y-axis represents, in order, the average measured frame-rate (2a), the average normalized error (2b), and the average NAP (2c). With the highest resource availability (4-cores) the AS-RTM can provide 3 frames/s without quality loss (ERR$\simeq$0%). On the contrary, configurations with lower resource availability, show a quality loss which ranges from 20% to 50%, depending on the number of cores. This means that there is a range of goal values, different for each amount of available resources, where the AS-RTM can trade off performance and computation error to meet the goal.



(a) AS-Linux (b) RA-AS-Linux (c) AS-RTRM

Fig. 3: Behavior of the Run-Time Management strategies, in terms of throughput and normalized error.

Figure 2 shows a similar behavior but with different thresholds for the maximum frame-rate that they can reach: the test with 1-core provides up to 4.3 frames/s, with 2-cores up to 8.1 frames/s, with 3-cores up to 11.5 frames/s and with 4-cores up to 16.4 frame/s. After these frame-rate thresholds, the AS-RTM (already in the OP with lowest quality of result) cannot find any suitable OP to meet the goal, thus the NAP value starts growing.

In conclusion, this test demonstrates that, for the selected case study, our AS-RTM allows to satisfy higher throughput demands by exploiting the possible trade-offs in the objective space in terms of performance versus computation error. The dynamic workloads presented in the next section will benefit from this feature, since in a multi-application deployment configuration each instance cannot use the full platform, but is constrained to a subset of resources.

B. Evaluating RTM Strategies

In this section, the three RTM strategies that use the proposed AS-RTM are compared in terms of predictability and fairness. The experiment analyzes application adaptivity in a sequential scenario. In such scenario, four Stereo-Matching instances are executed on the Intel platform, with a fixed delay between the start times. The number of frames to be processed by each instance has been chosen to let all the applications run together for approximately $30s$, then they complete their execution at different times. All instances have the same throughput goal (4 frames/s) and their AS-RTM is configured to minimize the disparity error. We can logically partition the experiment in two phases. In the first phase new applications are launched, so we can observe how already running applications react when the new applications steal resources. The second phase begins when the oldest instance has completed its execution. In this phase, one by one, all applications leave the execution context, so it is possible to see how the remaining instances exploit the resources that are released. Figure 3 represents the three evaluated RTM strategies: AS-Linux (3a), the proposed Resource-Aware AS-Linux (3b) and AS-RTRM (3c). For each configuration, the plots show the throughput and disparity error profiled at run-time, in a time window of 300 sec.

AS-Linux. When only one application is running, the throughput is stable and the disparity error is constant. As soon as the second application is started ($t = 20s$), the throughput of both instances starts oscillating but the error remains constant. The reason for this is that the AS-RTM does not change the OP (the throughput is above the goal) but, since the total amount of resources demanded doubles the number of cores, the throughput is strongly related to the scheduler policies. After $60s$, the third application is executed and even more resources are demanded, strengthening the relation between the OS scheduling and the throughput oscillation. In this case, the measured throughput can go below the goal value, forcing the AS-RTM to select a faster OP, which in turns boosts the oscillation. In conclusion, this configuration is not fair neither predictable.

Proposed Resource-Aware AS-Linux. Here the behavior is quite different: after an initial transitory period, the constraint on the CPU utilization forces the AS-RTM to use only OPs that match the available resources, preventing the throughput oscillations. Whenever a new application starts or ends, the AS-RTM waits until the CPU usage, of both the system and the application, becomes stable before updating the CPU usage constraint. During these periods, the number of threads might be greater than the number of cores for short periods, then some oscillations can be observed (e.g. $t = 20s$, $t = 70s$). Then, such undesired oscillations end once the resources are partitioned among the applications. The CPU monitor allows the AS-RTM to gain predictability, however – as the disparity error plot shows – this strategy is not fair because the resource allocation is not coordinated among applications.

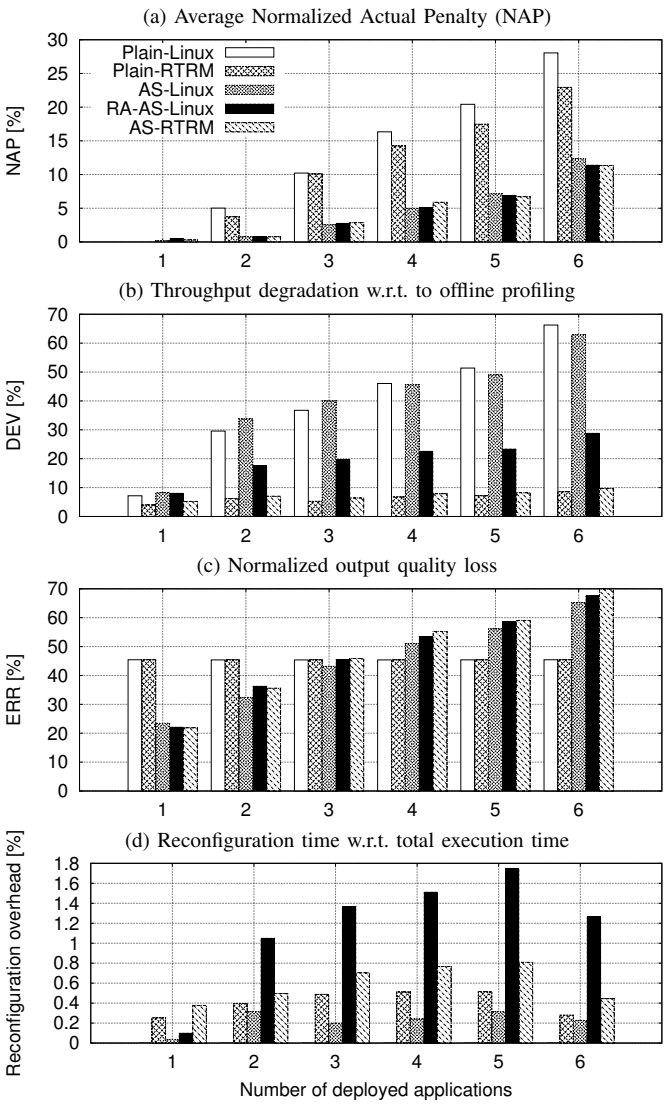


Fig. 4: Dynamic workload analysis on the AMD platform.

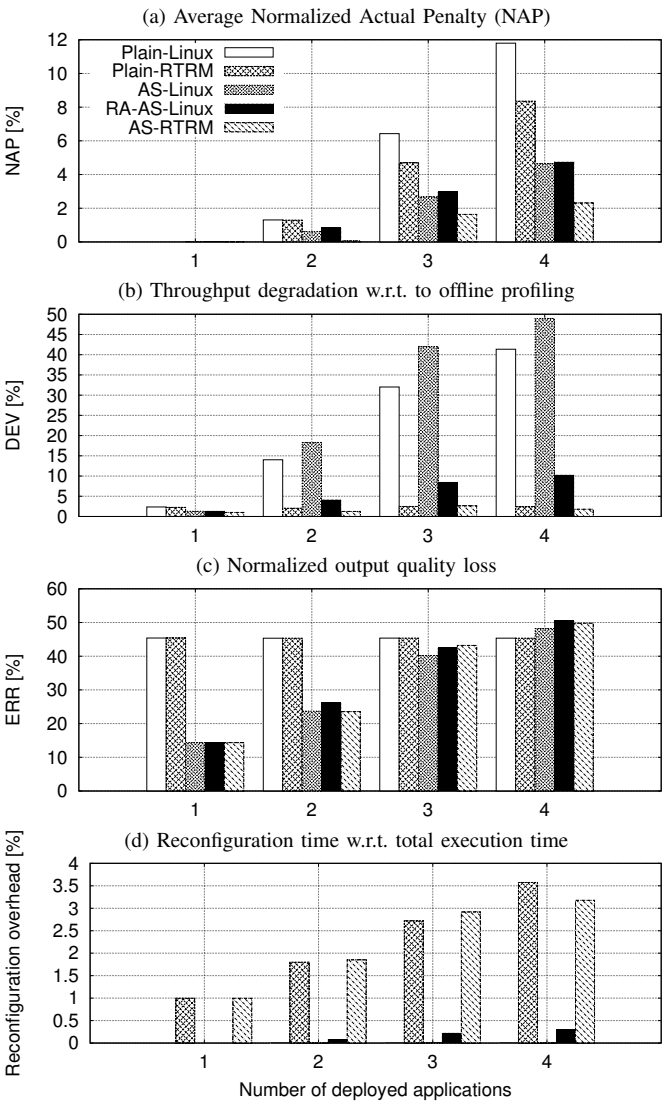


Fig. 5: Dynamic workload analysis on the Intel platform.

AS-RTRM. Thanks to the centralized coordination, the transitory periods – whenever an application starts or ends – are drastically reduced. This configuration provides the best performance predictability and allocation fairness. However, except for the transitory periods, the performance achieved by the proposed *Resource-Aware AS-Linux* and the *AS-RTRM* are similar (see the throughput plots).

C. Dynamic Workload Results

This section describes the results obtained by deploying a multi-application configuration on both reference platforms. The maximum number of instances and the maximum frame-rate goal are shown in Table I.

Figure 4 reports the results for the test on the AMD platform, while Figure 5 for the Intel one. As shown in Figure 4a and Figure 5a, *Plain-Linux* has the worst NAP metric: although the single application can reach all throughput demands, concurrent execution of applications with different resource demands introduces high penalties on the performance metrics. In this configuration, all applications use by default the entire CPU (device fission is disabled). This introduces a high rate of context-switches, which degrades the measured frame-rate. As a consequence, the difference between the design-time and run-time profiling is highest for this configuration and such deviation continues increasing as we deploy more concurrent applications. This is an expected result since the OpenCL library relies on the OS scheduler to allocate user time to different applications.

In *Plain-RTRM* the system-wide coordination of resource allocation has the most significant impact on predictability of application performance: indeed, the metric of performance deviation is the lowest for this configuration. The NAP benefits from the execution in a controlled environment, since the allocation of CPU cores has the effect of reducing the number of context-switches. However, the QoS of the application output is still fixed so the reconfiguration options are limited to the computational parallelism.

This is not the case of *AS-Linux*, where the QoS metric (Figure 4c and Figure 5c) is tuned at run-time to react to variations in the system workload. The error associated to the application output is below *Plain-RTRM* in scenarios with 1-3 instances, above for scenarios with 4-6 instances. The NAP

benefits from the wider range of trade-offs, which is lower than in *Plain-RTRM*; however, the predictability of performance metrics is low, as shown by the performance deviation bar (Figure 4b and Figure 5b).

This limitation is overtaken by the proposed *Resource-Aware AS-Linux*, where an application is allowed to use resources only if these are available. *RA-AS-Linux* performance is better than *Plain-RTRM* in our tests, because the AS-RTM is more reactive than a centralized resource manager; on the other hand, the performance predictability is slightly worse in high contention scenarios. It must be noticed that *RA-Adaptive-Linux* cannot provide any guarantees on fairness in the resource allocation, nor can support applications with different priority levels whereas the BarbequeRTRM can do.

Finally, the configuration that performs best in all scenarios is the *AS-RTRM*. By combining the benefits of system-wide resource management and application-level auto-tuning, it is possible to achieve the best performance. However, this configuration requires a more complex software framework, which might not be needed for best-effort applications.

The average Normalized Actual Penalty (NAP) (Figure 4a and Figure 5a) is the metric that better summarizes this analysis. We can observe, as expected, an increasing NAP for all configurations as the workload is increased. Nevertheless, the adaptive configurations (supported by the AS-RTM) always reduce the NAP with respect to the plain configuration, which means that the frame-rate goal is met much more frequently. However, the NAP metric considers only the processing time and not the run-time management overhead, e.g. the time spent in reconfiguration. Thus Figure 4d and Figure 5d show also the reconfiguration overhead, with respect to the total execution time of each experiment. The average overhead in the configurations with BarbequeRTRM is 0.4% on the AMD platform and 2% on the Intel platform. In both platforms, we use synchronous OpenCL program build (see Section IV-D), because the application execution context is not aware of the system state, thus it cannot control the rescheduling events. Therefore, the difference in reconfiguration overhead depends on the OpenCL run-time libraries: the build of Stereo-Matching kernels takes $154ms$ on AMD and $624ms$ on Intel platform, respectively. On the other hand, the overhead of *AS-Linux* and *RA-AS-Linux* is different between the two platforms for another reason: the Intel platform supports asynchronous OpenCL program build, while AMD does not. This feature can be exploited in the configurations with decentralized resource management, because reconfiguration is completely managed by the AS-RTM. On our Intel platform, this results in a 10x reduction of the reconfiguration overhead.

VI. Conclusions

In this work, we have considered the problem of managing multiple OpenCL applications for server consolidation on multi-core platforms. The application we targeted in our tests, Stereo-Matching, can achieve different performance (frame-rate) depending on the computational capabilities of the platform, however more fine-grained control of the resource usage is done by means of the OpenCL *device fission* API.

We have evaluated different Run-Time Management strategies, in terms of adaptability and predictability in the OpenCL context, reproducing some approaches proposed in literature ([11], [7]). Moreover, we have introduced a light-weight Run-Time Management technique, which extends the trade-off space of an Application-Specific Run-Time Manager to resource-usage control. This technique, targeted to compute-intensive applications, allows to take local decision on resource utilization at application level, for efficient resource sharing. Differently from other distributed approaches, applications act as autonomous agents, without coordination among them.

Our tests show that the average performance of the proposed AS-RTM is very close to the performance achieved with a combined approach based on a centralized resource manager; at the same time, our approach is more portable and less intrusive from an application design point of view.

References

[1] Khronos Group, "OpenCL Specification," 2012.

[2] H. Hoffmann, J. Holt, G. Kurian, E. Lau, M. Maggio, J. Miller, S. Neuman, M. Sinangil, Y. Sinangil, A. Agarwal, A. Chandrakasan, and S. Devadas, "Self-aware computing in the Angstrom processor," pp. 259–264, 2012.

[3] S. Wildermann, T. Ziermann, and J. Teich, "Game-Theoretic Analysis of Decentralized Core Allocation Schemes on Many-Core Systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. New Jersey: IEEE Conference Publications, 2013, pp. 1498–1503.

[4] H. Endt and K. Weckemann, "Remote Utilization of OpenCL for Flexible Computation Offloading using Embedded ECUs, CE Devices and Cloud Servers." in *PARCO*, 2011, pp. 133–140.

[5] Y. Wang, S. Chen, H. Goudarzi, and M. Pedram, "Resource allocation and consolidation in a multi-core server cluster using a Markov decision process model," in *International Symposium on Quality Electronic Design (ISQED)*. IEEE, Mar. 2013, pp. 635–642.

[6] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 1, pp. 239–246, Jan. 2013.

[7] P. Bellasi, G. Massari, and W. Fornaciari, "A RTRM proposal for multi/many-core platforms and reconfigurable applications," in *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, Jul. 2012, pp. 1–8.

[8] C. Ykman-Couvreur, P. A. Hartmann, G. Palermo, F. Colas-Bigey, and L. San, "Run-time resource management based on design space exploration," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*. New York, USA: ACM Press, Oct. 2012, p. 557.

[9] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "ARTE: An Application-specific Run-Time managEment framework for multi-cores based on queuing models," *Parallel Computing*, vol. 39, no. 9, pp. 504–519, Sep. 2013.

[10] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*. New York, USA: ACM Press, Sep. 2011, p. 124.

[11] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," *ACM SIGPLAN Notices*, vol. 47, no. 4, p. 199, Jun. 2012.

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

[13] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats," in *Proceeding of the 7th international conference on Autonomic computing - ICAC '10*. New York, USA: ACM Press, Jun. 2010, p. 79.

[14] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schrder-Preikschat, and G. Snelting, "Invasive computing: An overview," in *Multiprocessor System-on-Chip*, M. Hbner and J. Becker, Eds. Springer New York, 2011, pp. 241–268.

[15] K. Zhang, J. Lu, and G. Lafruit, "Cross-Based Local Stereo Matching Using Orthogonal Integral Images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1073–1079, Jul. 2009.