ABSTRACT

Heterogeneous systems are composed of a set of different processing units, each with its own performance capabilities and energy characteristics. The majority of current network packet processing frameworks utilize a separate computational device for each application, typically the most powerful, to cope with the constantly increasing network link speeds. However, this may result in the under-utilization of the device, as more applications could potentially execute at the same time. In this work, we propose an adaptive scheduling approach, tailored for demanding network packet processing applications, that carefully schedules applications across a diverse set of computational devices.

1. MOTIVATION

Current research has focused on building high-performance, scalable and energy efficient network packet processing systems. Some of these approaches tend to target only a single device (i.e. either a CPU or a GPU), since uniformly developing an application that utilizes all the available diverse compute devices effectively and consistently is considered challenging. These approaches dedicate a single device to accelerate a single application [5]. Others, propose device-specific scheduling approaches for concurrently running applications (targeting either GPUs or CPUs) [2, 4]. Recently, researchers have developed other approaches, to efficiently schedule a single network packet processing application among multiple heterogeneous hardware architectures [3].

However, there is a lack of support for concurrently running applications that fully utilize the multiple and heterogeneous hardware architectures (i.e. CPUs, integrated GPUs and discrete high-end GPUs), that are available in modern commodity systems. Moreover, the constantly increasing network link speeds require powerful packet processing systems, to effectively handle the input traffic that needs to be distributed appropriately among the available devices. Finally, energy efficiency is considered as a very important attribute in today’s systems.

In this work, we propose a scheduling approach, to address these requirements: (i) efficient scheduling of concurrently running network packet processing applications on heterogeneous systems, (ii) adaptive and effective network traffic management and distribution, and (iii) energy efficiency. The proposed scheduling approach will contribute to more advanced network packet processing systems with rich performance characteristics, like higher throughput, lower latency, better power consumption, and optimal resource utilization.

2. BACKGROUND

A typical commodity hardware architecture offers heterogeneity at three levels: (i) at the traditional x86 CPU architecture, (ii) at an integrated GPU packed on the same processor die, and (iii) at a discrete high-end GPU. Each one of these distinct processing units is characterized by different performance and energy features. Typically, a GPU is appropriate for data-parallel workloads, while a CPU is suitable for handling branch-intensive and latency-critical workloads. On the other hand, an integrated GPU is high energy efficient without depressing the processing throughput or latency.

The challenge to fully utilize a heterogeneous system, is to appropriately map computations to devices, as automated as possible. Previous works, assume that all devices perform equally, in order to automatically partition the workloads across the devices, or require a number of execution trials to determine their capabilities. Furthermore, many of these approaches have been designed for applications that take as input constant streaming data, and as a consequence, are unable to adapt when the input data stream varies, as in the case of network traffic.

3. APPROACH

3.1 Workloads

So far, we have implemented four simple, but typical, packet processing applications that involve both computational and memory intensive behavior (i) IPv4 packet forwarding, (ii) deep packet inspection using a DFA implementation of the Aho-Corasick string pattern matching algorithm, (iii) packet hashing using the MD5 algorithm, and (iv) AES-CBC encryption using a different 128-bit key for each connection. These applications are implemented using the OpenCL framework [1], to be uniformly executed across all the distinct devices in our system.

3.2 System Setup

Our base system is equipped with one Intel Core i7-3770 Ivy Bridge processor and one NVIDIA GeForce GTX 770 graphics card. The processor contains four CPU cores operating at 3.4GHz, with hyper-threading support, resulting in eight hardware threads, and an integrated HD Graphics 4000 GPU. Overall, our system contains three different, heterogeneous, computational devices: one CPU, one integrated GPU and one discrete GPU. Additionally, to accurately measure the power consumption we use four current sensors to monitor (real-time) the consumption of the CPU, GPU, DRAM and miscellaneous motherboard peripherals.

3.3 Performance Characterization

In this section we present the performance achieved by our applications. Specifically, we measure the sustained throughput and power consumption for each of the devices that are available in our base system. We use a synthetic packet trace and a different packet batch size each time. To accurately measure the power spent for
3.4 Adaptive Scheduler

The goal of our proposed scheduler is to be efficient in terms of throughput, energy consumption and latency, while utilizing all the available hardware resources. The scheduling algorithm allows an application to be configured regarding its needs (i.e. high throughput, low energy consumption or latency) and balance between them. Furthermore, the scheduler must be highly adaptive, since it is dedicated to network packet processing applications. It is essential that the scheduler is able to adjust to any input traffic amounts, since variable incoming rates affect significantly the performance of packet processing applications. Thus, to build the appropriate scheduling algorithm, we have to take into account the threefold heterogeneity of (i) the hardware, (ii) the applications and (iii) the network traffic. The most challenging part to provide efficient resource utilization for concurrently running network packet applications is to accurately classify and address any interference effects that arise, including (i) thermal constraints, (ii) contention for hardware resources (e.g. shared caches, I/O interconnects, memory controller, etc.), (iii) software resources, (iv) false sharing of cache blocks, and others.

The first step, to acquire a prior knowledge regarding our system characteristics is a coarse profiling of each new application on the available resources of our system. This initial training needs to profile all the possible combinations of the different configurations. This phase is very important since the scheduler learns the performance, latency and energy response of each device, in respect to the packet batching as well as the partitioning of each batch on every device (see Figures 1(a) and 1(b)). The next step for our scheduler is to decide the best combination of applications, devices and packet batch sizes, which will contribute to the desired performance target (e.g. low power consumption). Finally, the scheduler will periodically monitor the incoming traffic rate to adapt the batch partitioning or disable a power-hungry device that its use is redundant, for instance.

3.5 System Scalability

Our current system targets a single commodity machine that contains multiple heterogeneous hardware processing architectures. Likewise, our system can be extended accordingly, to target a distributed environment consisting of multiple commodity machines. Obviously, this will add an extra level of heterogeneity that needs to be considered carefully.

Acknowledgments

This work has been partially supported by the European Commission under the Horizon 2020 Program through the RAPID project under Grant Agreement 644312.

4. REFERENCES