Parallel Computing 44 (2015) 37–52

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures

João V.F. Lima [a,*], Thierry Gautier [d,b,c], Vincent Danjean [b,c,d], Bruno Raffin [d,b,c], Nicolas Maillard [a]

[a] Inst. of Informatics, UFRGS, CP 15064, 91501-970 Porto Alegre, RS, Brazil
[b] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
[c] CNRS, LIG, F-38000 Grenoble, France
[d] Inria, France

ARTICLE INFO

Article history:
Received 5 April 2014
Received in revised form 28 January 2015
Accepted 1 March 2015
Available online 6 March 2015

Keywords:
Parallel programming
Accelerators
Task parallelism
Data-flow dependencies
Work stealing

ABSTRACT

In this paper, we present a comparison of scheduling strategies for heterogeneous multi-CPU and multi-GPU architectures. We designed and evaluated four scheduling strategies on top of XKaapi runtime: work stealing, data-aware work stealing, locality-aware work stealing, and Heterogeneous Earliest-Finish-Time (HEFT). On a heterogeneous architecture with 12 CPUs and 8 GPUs, we analysed our scheduling strategies with four benchmarks: a BLAS-1 AXPY vector operation, a Jacobi 2D iterative computation, and two linear algebra algorithms Cholesky and LU. We conclude that the use of work stealing may be efficient if task annotations are given along with a data locality strategy. Furthermore, our experimental results suggests that HEFT scheduling performs better on applications with very regular computations and low data locality.

1. Introduction

With the recent evolution of processor design, future generations of processors will contain hundreds of cores. To increase the performance per watt ratio, the cores will be non-symmetric with few highly powerful cores and numerous, but simpler, cores. The success of these machines will rely on the ability to schedule the workload at runtime, even for small problem instances.

One of the main challenges is to design a scheduling strategy that may be able to exploit all potential parallelism on heterogeneous architectures composed of multi-CPUs and multi-GPUs. Previous works demonstrate the efficiency of strategies such as static distribution [1–4], centralized list scheduling with data locality [5], cost models [6,7] based on Earliest-Finish-Time scheduling [8], and dynamic for a specific application domain [9,10].

In our previous works, we state that the classic work stealing is cache-unfriendly and does not consider data locality [11,12]. We described a locality-aware work stealing with annotations that improves significantly the performance of compute-bound linear algebra problems [11]. However, it does not consider the processing power of available resources. Few studies have compared performance of different scheduling strategies for heterogeneous multi-CPU and multi-GPU platforms.

* Corresponding author.
E-mail addresses: jvlima@inf.ufsm.br (J.V.F. Lima), thierry.gautier@inrialpes.fr (T. Gautier), vincent.danjean@imag.fr (V. Danjean), bruno.raffin@inria.fr (B. Raffin), nicolas@inf.ufrgs.br (N. Maillard).

http://dx.doi.org/10.1016/j.parco.2015.03.001
0167-8191/

The purpose of this paper is to compare scheduling strategies based on dynamic scheduling and cost models for data-flow task programming on heterogeneous architectures. We designed and evaluated four scheduling strategies on top of XKaapi runtime: work stealing [12,13], data-aware work stealing [5,12], locality-aware work stealing [11,14], and Heterogeneous Earliest-Finish-Time (HEFT) [6,8].

We analysed our scheduling strategies with four benchmarks: a BLAS-1 AXPY vector operation, a Jacobi 2D iterative computation, and two linear algebra algorithms (Cholesky and LU). We conclude that the use of work stealing may be efficient if task annotations are given (such as `SetArch`) along with a data locality strategy. Experiments with LU showed that our locality-aware work stealing improved performance over HEFT by 13.24% and reduced data footprint by 17.29% on 4CPUs-8GPUs. Furthermore, our experimental results suggest that HEFT performs better on applications with very regular computations and low data locality. Jacobi 2D results showed that HEFT reduced iteration time by 53.19% and data footprint by 73.11% over locality-aware work stealing on 4 CPUs-8 GPUs.

The remainder of the manuscript is organized as follows. Section 2 presents the related work on runtime systems and scheduling algorithms for heterogeneous architectures. Section 3 gives an overview of XKaapi runtime and extensions from our previous works [11,12] for multi-GPU systems. Section 4 details the contribution of this paper on scheduling strategies for multi-CPU and multi-GPU architectures. Our experimental results are presented in Section 5. Finally, Section 6 and Section 7, respectively, present the discussion and conclude the paper.

2. Related work

In this section, we present runtime systems and scheduling algorithms for heterogeneous systems.

Charm++ has two extensions to support GPUs. Charm++ GPU Manager [15] allowed the overlap of chare objects (or computations) with transfers to GPUs. G-Charm [16] schedules at runtime chare objects between CPUs and GPUs based on the current loads and the estimated execution time provided by previous executions.

KAAPI [17] was specialized for multi-CPU and multi-GPU systems [10]. Each CPU or GPU implementation of a given task was encapsulated in a functor object, which provided a clear separation between a task definition and its various implementations, *i.e.*, one to each architecture.

StarPU is a runtime system providing a data management facility and an unified execution model over heterogeneous architectures including GPUs and Cell BE processors [6,7]. Its programming model relies on explicit parallelism by tasks with data dependencies and a memory layer to abstract transfers among disjoint address spaces.

StarSs proposes a programming model to exploit task-level parallelism by OpenMP-like pragmas [18] and a runtime system to schedule tasks while preserving dependencies. OmpSs [5] is a continuation of StarSs and extends SMPSs [19] and GPUSs [18] by providing simpler code annotations with the capacity to have recursive tasks. Recent versions of OmpSs include specific multi-versions of the same task [20] and regions of strided and/or overlapped data [21].

In the context of scheduling, many works in the literature propose strategies involving GPUs. There are related works in scheduling restricted to GPUs [22] and based on runtime systems [5,7,10,18,20]. Hermann et al. [10] propose a static and dynamic scheduling for iterative computations on multi-CPU and multi-GPU systems. The task graph partitioning (static phase) and work stealing (dynamic phase), respectively, enforce data locality and reduce the total number of steals.

Augonnet et al. [7] study strategies based on the Heterogeneous Earliest-Finish-Time (HEFT) scheduling algorithm [8]. They compare the impact of execution time (*heft-tm*) in addition to data transfers (*heft-tmdp*) and data prefetch (*heft-tmdp-pr*).

Ayguadé et al. [18] describe the GPUSs centralized list scheduling to minimize data transfers. Recent versions of GPUSs, called OmpSs [5], depict two scheduling policies: centralized with first-in first-out (FIFO) and locality-aware. The strategies proposed in GPUSs and OmpSs have the limitation of strict usage of GPUs, while CPUs are involved only in runtime routines. Planas et al. [20] study a scheduling strategy with task versioning similar to StarPU on multi-CPU and multi-GPU systems.

In our previous works, we state that the classic work stealing is cache-unfriendly and does not consider data locality [11,12]. We described a locality-aware work stealing with annotations that improves significantly the performance of compute-bound linear algebra problems [11]. However, it does not consider the processing power of available resources. In this sense, our strategy without any annotation is not aware of the efficiency in a certain architecture type. Since CPUs and GPUs are heterogeneous in computing power, a bad decision will affect tasks outside the critical path, which are candidates to be offloaded to accelerators, and may impact performance.

3. XKaapi runtime system overview

In this section, we introduce the XKaapi programming model and runtime system. We overview its software stack (3.1), programming model (3.2), scheduling algorithm (3.3), and multi-GPU runtime support (3.4). The aspects described in this section are previous contributions on XKaapi [10–12,17,23] and provide the basis for this paper's contribution on scheduling strategies for heterogeneous architectures.