Data & Knowledge Engineering 93 (2014) 60–79

Contents lists available at ScienceDirect

Data & Knowledge Engineering

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

Editorial

Load-aware inter-co-processor parallelism in database query processing ☆

Sebastian Breß [a,*], Norbert Siegmund [b], Max Heimel [c], Michael Saecker [d,c], Tobias Lauer [e], Ladjel Bellatreche [f], Gunter Saake [a]

[a] Otto von Guericke University Magdeburg, P.O. Box 4120, D-39016 Magdeburg, Germany
[b] University of Passau, Innstraße 41, D-94032 Passau, Germany
[c] Technische Universität Berlin, Straße des 17. Juni 135, D-10623 Berlin, Germany
[d] ParStream GmbH, Große Sandkaul 2, D-50667 Cologne, Germany
[e] Jedox AG, Bismarckallee 7a, D-79098 Freiburg im Breisgau, Germany
[f] LIAS/ISAE-ENSMA, 1 avenue Clément Ader BP 40109, F-86961 Futuroscope, France

ARTICLE INFO

Article history:
Received 31 December 2013
Received in revised form 31 March 2014
Accepted 3 July 2014
Available online 10 July 2014

Keywords:
Co-processing
Query processing
Query optimization

ABSTRACT

For a decade, the database community has been exploring graphics processing units and other co-processors to accelerate query processing. While the developed algorithms often outperform their CPU counterparts, it is not beneficial to keep processing devices idle while overutilizing others. Therefore, an approach is needed that efficiently distributes a workload on available (co-)processors while providing accurate performance estimates for the query optimizer. In this paper, we contribute heuristics that optimize query processing for response time and throughput simultaneously via inter-device parallelism. Our empirical evaluation reveals that the new approach achieves speedups up to 1.85 compared to state-of-the-art approaches while preserving accurate performance estimations. In a further series of experiments, we evaluate our approach on two new use cases: joining and sorting. Furthermore, we use a simulation to assess the performance of our approach for systems with multiple co-processors and derive some general rules that impact performance in those systems.

1. Introduction

In recent years, processing devices became more and more heterogeneous, and will likely become even more heterogeneous in the future [16,31]. Using such co-processors to accelerate database query processing became increasingly popular over the last years. Many efficient algorithms for data processing were developed to exploit the processing capabilities of modern co-processors, such as Graphics Processing Units (GPUs) [4,10,14], Accelerated Processing Units (APUs) [15] or Field Programmable Gate Arrays (FPGAs) [24].

Most of the aforementioned approaches aim at improving the efficiency of database operations. Only few solutions address the challenge of utilizing multiple processing devices efficiently (i.e., using the processing device that promises the highest gain w.r.t. an optimization criterion while keeping all processing devices busy). There are two major classes of solutions in this field: (1) heterogeneous task-scheduling approaches and (2) tailor-made co-processing approaches. With (1), we do not know the specifics of

☆ This paper is a substantially extended version of an earlier work [9].

* Corresponding author.
E-mail addresses: bress@ovgu.de (S. Breß), siegmunn@fim.uni-passau.de (N. Siegmund), max.heimel@tu-berlin.de (M. Heimel), michael.saecker@parstream.com (M. Saecker), tobias.lauer@jedox.com (T. Lauer), bellatreche@ensma.fr (L. Bellatreche), saake@ovgu.de (G. Saake).

http://dx.doi.org/10.1016/j.datak.2014.07.003
0169-023X/

database systems (e.g., the set of operations and data representations, access structures, optimizer specifics and concurrency control). Additionally, task scheduling approaches typically require a system to use task abstractions of a certain framework (e.g., Augonnet and others [3] or Ilić and Sousa [18]). Since DBMS have their own task abstractions, a large part of code would have to be rewritten. With (2), we are bound to operations in a specific DBMS (e.g., He and others [14] or Malik and others [21]).

1.1. Problem statement

There is no approach that exploits DBMS-specific optimizations while being independent of an algorithm's implementation details or the hardware in the deployment environment. To close this gap, we presented in prior work a decision model that distributes database operations on available processing devices [7]. The model uses a learning-based approach that is independent of implementation details and hardware while providing accurate cost estimations for database operations. We implemented our decision model in a hybrid query-processing engine (HyPE) [6], which is designed to be applicable to any hybrid DBMS.[1] The problem is that in real life systems, a machine may contain several heterogeneous co-processors besides a few CPUs. Each processing device has its own load and in case they are not homogeneous, different processing speed. However, there is no state-of-the-art approach that is capable to distribute a workload of database operators on such a system while taking into account the load and relative speed of each processing device.

1.2. Research question

In a more general context, we have to answer the following research question: How can we distribute a workload of database operators on processing devices with different speeds and load factors efficiently?

1.3. Contributions

In this paper, we make the following contributions:

1. We introduce heuristics that allow us to handle operator streams and efficiently utilize inter-device parallelism by adding new optimization heuristics for response time and throughput.
2. We provide an extension to HyPE, which implements the heuristics.
3. We present an exhaustive evaluation of our optimization heuristics w.r.t. varying parameters of the workload using micro benchmarks (e.g., to identify the most suitable heuristic).

This is a revised version of a previous paper [9]. Compared to this earlier work, we make the following novel contributions:

1. We introduce a new optimization heuristic called probability-based outsourcing, which selects devices at random with a probability that corresponds to their expected performance.
2. Then, we extend our evaluation by two additional use cases: joining and sorting to validate our approach on the most common operations in a column store, a central aspect of a GPU-accelerated DBMS [8].
3. Finally, we investigate how the best optimization heuristic scales with an increasing number of co-processors and an increasing speed difference between processing devices, thus proving the overall applicability of our load-aware scheduling in databases.

1.4. Major findings

We find that our approaches can reliably balance a workload not a priori known on all available (co-)processors. The (co-)processors may have significantly different processing speeds for certain operations, which are automatically learned by our system. In a further series of experiment in a simulator, we find that the dominating performance bottleneck in a multi co-processor system is to transfer result data back to the CPU.

1.5. Outline

The paper is structured as follows: in Section 2, we present our preliminary considerations. We discuss operator-stream-based query processing as well as HyPE's extensions in Section 3. We introduce our optimization heuristics in Section 4 and provide an exhaustive evaluation using micro benchmarks in Section 5. We conduct additional experiments with our simulator in Section 6 and discuss related work in Section 7.

2. Preliminary considerations

In this section, we provide a brief background on GPUs, because we will evaluate our approach on a CPU/GPU system. Then, we discuss our decision model [7], which we extend to support operator-stream-based scheduling.

[1] A DBMS that implements for each operation at least an operator for a CPU and a co-processor, such as *GDB* [14], Ocelot [16], or MapD [23].