Hardware Acceleration of Database Analytics

Evangelia Sitaridi
Amazon Web Services
e-mail: sitaridi@amazon.com

Advances in hardware strongly influence the database system design. The flattening speed of CPU cores makes many-core accelerators a vital alternative to explore for processing the ever-increasing amounts of data. Accelerators typically employ more and simpler cores with reduced instruction control overhead. For example, in GPUs, multiple execution units share the same instruction sequencing logic. As a result, GPU processors do not face the power constraints limiting the parallelism of CPUs.

Use of GPUs yields significant speed-ups for database processing [1], [2]. The increased availability of GPUs in the cloud opens new acceleration possibilities. To harness the power of GPUs, we adapt and redesign data analytics operators to exploit better their special memory and threading model. Due to the increasing memory capacities and also the users' need for fast interaction with data, as in the case of visual analytics, we focus on in-memory data processing.

We identify two key applications for hardware acceleration because of their impact on end-to-end system performance: a) Substring matching and b) Lossless data compression. Our techniques cover data preprocessing and algorithmic operator optimization taking the GPU control flow into consideration. We also discuss to what extent we can adapt our techniques for CPUs with SIMD extensions.

Substring matching is a necessary tool to explore the increasing volume of both unstructured and structured data, such as social media posts and log files. In the context of a database system, we are often interested only in the first match. If some threads locate a match early on, they can stop scanning the string. However, because of the warped execution of GPUs, these threads will remain idle while the remaining threads are still scanning their input string, so we suggest methods to reduce GPU thread underutilization.

We study the cache memory efficiency of single and multi-pattern string matching algorithms for conventional and novel pivoted string layouts in the GPU memory [3]. Our pivoted layout splits the input strings into similarly sized pieces and interleaves the string pieces in the GPU main-memory. As a result, threads of a warp processing contiguous strings will coalesce their accesses to the GPU memory. However, depending on the chosen string matching algorithm, some threads might fall behind and will end up accessing different string pieces, increasing the cache footprint. We define this effect as *memory divergence*. Because of the limited L2 capacity, memory divergence degrades the string matching performance, but we suggest how to eliminate memory divergence during substring matching.

We benchmark GPU substring matching performance and show 3× improved performance of GPUs against vectorized multi-core CPU state-of-the-art libraries. While a single instruction is enough to answer most queries with substring matching operators in CPUs with SIMD capabilities, more advanced predicates such as regular expression matching cannot be optimized as easily. Hence to offer regular expression capabilities in the DBMS, we follow a different approach by accessing the input strings non-sequentially and eliminate the inherent need for branching, whether the string matches or not a regular expression. Our evaluation on mainstream CPUs and co-processors shows our approach to be up to 5× faster compared to the scalar implementations.

In the second application, we parallelize Big Data decompression algorithms in our compression framework *Gompresso* [4]. Most research so far had focused on the speed of compressing data as it is loaded, but decompression speed can be more important for modern workloads – data is compressed only once when loaded into the database but repeatedly decompressed as it is read when executing analytics or machine learning jobs. We propose and evaluate two approaches to parallelize Inflate on GPUs efficiently. The first technique exploits the SIMD-like execution model of GPUs to coordinate the threads that are concurrently decompressing a data block. The second approach avoids data dependencies encountered during decompression by proactively eliminating performance-limiting dependencies during the compression phase. The resulting speed gain comes at the price of a marginal loss of compression efficiency. Gompresso is suitable for any massively parallel processor and is 2× faster in a head-to-head comparison with several multi-core CPU-based libraries while achieving a 17% energy saving with comparable compression ratios.

Motivated by the increased availability of high-end GPUs on the cloud we conclude outlining the challenges of managing heterogeneous hardware on the cloud and discuss future research directions to address them.

References

[1] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk, "GPU join processing revisited," in *DaMoN*, 2012, pp. 55–62.

[2] H. Wu, G. F. Diamos, T. Sheard, M. Aref, S. Baxter, M. Garland, and S. Yalamanchili, "Red fox: An execution environment for relational query processing on GPUs," in *CGO*, 2014.

[3] E. A. Sitaridi and K. A. Ross, "GPU-accelerated string matching for database applications," *VLDB J.*, vol. 25, no. 5, pp. 719–740, 2016.

[4] E. A. Sitaridi, R. Müller, T. Kaldewey, G. M. Lohman, and K. A. Ross, "Massively-parallel lossless data decompression," in *ICPP*, 2016, pp. 242–247.