Accelerating the Unacceleratable: Hybrid CPU/GPU Algorithms for Memory-Bound Database Primitives

Michael Gowanlock[1], Ben Karsin[2], Zane Fink[1], and Jordan Wright[1]

[1]School of Informatics, Computing, and Cyber Systems at Northern Arizona University

[2]Department of Computer Science at Universite libre de Bruxelles

Abstract

Many database operations have a low compute to memory access ratio where achieving performance gains is percieved as insurmountable. We examine several of these memory-bound algorithms, including (i) batched predecessor searches; (ii) multiway merging; and, (iii) partitioning. We examine the performance of parallel CPU-only, GPU-only, and hybrid CPU/GPU approaches, and show that hybrid algorithms achieve respectable performance gains. We develop a model that considers main memory accesses and PCIe data transfers, which are two major bottlenecks for hybrid CPU/GPU algorithms. The model lets us analytically determine how to distribute work between the CPU and GPU to maximize resource utilization while minimizing load imbalance. We show that our model can accurately predict the fraction of work to be sent to each architecture, and consequently, confirms that these overlooked database primitives can be accelerated despite their memory-bound nature.

Introduction

- While compute-intensive operations have seen performance gains using GPUs, many database primitives, which perform many operations in-memory, have not been accelerated due to their perceived work-efficiency.
- One approach to improve the performance of memory-bound algorithms is to develop hybrid parallel algorithms that use both CPU and GPU resources, where each architecture performs part of the total computation.
- Most GPU research is dedicated to GPU-only approaches which solve the entire computation on the GPU. We utilize both the CPU and GPU to compute these database primitives.
- For data-intensive memory-bound algorithms, the CPU-GPU interconnect is the performance bottleneck, and if this bottleneck can be overcome, there is an opportunity to exploit the GPU's high memory bandwidth.
- We propose *accelerating the unacceleratable* — which we define as memory-bound database primitives that are well-suited to a hybrid CPU/GPU execution but not necessarily a GPU-only execution.
- As a demonstration of the potential improvement over CPU-only primitives, we develop hybrid CPU/GPU algorithms to efficiently solve the following problems: (i) batched predecessor searches; (ii) multiway merging; and, (iii) k-way partitioning.
- Our model for both CPU and GPU performance uses the well-known external memory (EM) model with the exception that we consider the total number of main memory *elements* loaded/stored as our performance metric instead of a fixed memory size M and block size B.
- We base our approaches on EM-optimal algorithms to minimize main memory accesses by both the CPU and GPU, and use benchmarks and experimental results to further improve performance on our hardware platform.

Problem Statement

For each of our database primitives we implement CPU-only, GPU-only, and hybrid CPU/GPU algorithms. We consider a platform with multi-core CPUs and a GPU, where the total response time includes all data transfers to and from the GPU and related overheads. The final result of each algorithm is stored in main memory. We assume that each algorithm can exceed the GPU's global memory capacity. However, each algorithm may not exceed main memory capacity, as we do not consider the impact of disk accesses in this work.

Hybrid Algorithms

- The three algorithms that we consider are parallelizable across architectures while minimizing memory accesses. We accomplish this by breaking up the total work into several *batches* of divisible workloads that can be computed independently on either architecture.
- We define n_b to be the number of batches. For batched predecessor search and partitioning, we arbitrarily select $n_b = 400$, while for multiway merge, we make n_b a function of k to ensure data transfers are sufficiently large to mitigate overheads.
- We model evenly splitting the queries based on PCIe and memory bandwidth to obtain low load imbalance. Let β be the unidirectional bandwidth over PCIe, and α be the memory bandwidth between the CPU and main memory when simultaneously reading and writing, where β and α are given in elements per second. For a given platform, β and α can be obtained through simple microbenchmarks.

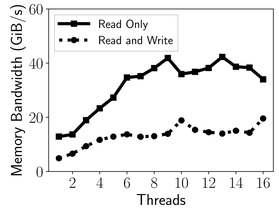


Figure 1. Main memory bandwidth vs. the number of threads. Multiple threads are needed to saturate memory bandwidth. Simultaneous reading and writing has lower throughput than reading only.

- We execute a simple microbenchmark which simultaneously reads and writes 5 GiB of 64-bit integers stored in main memory using 16 threads to saturate memory bandwidth. We obtain $\alpha = 19.56$ GiB/s on our platform. To obtain β, we transfer 10 GiB from pinned memory on the host to global memory on the GPU. We use the profiler to measure the bandwidth, and we obtain $\beta = 11$ GiB/s.
- Figure 1 shows the scalability of the microbenchmark. For comparison purposes, we plot the read only memory bandwidth (reading 10 GiB of 64-bit integers) and observe that the memory bandwidth is significantly higher than when performing both reading and writing.

Batched Predecessor Search

Let A be a set of keys sorted in non-decreasing order, where each key is denoted as a_i, where $i = 1, 2, \ldots, n$, and B be a set of queries sorted in non-decreasing order, where each query is denoted as b_j, where $j = 1, 2, \ldots, n$. For each query, $b_j \in B$, the batched predecessor search finds the largest value of i, such that $a_i \leq b_j$. While A and B can vary in size, for simplicity, we assume $|A| = |B| = n$.

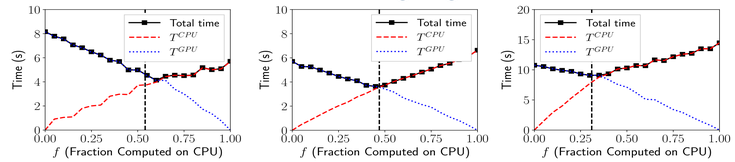
Multiway Merging

Given input array A consisting of k sublists, denoted as S_j, where $j = 1, 2, \ldots, k$, each of size $\frac{n}{k}$ and sorted in non-decreasing order, we wish to output the n total elements in sorted order. Furthermore, we assume that k is small enough that we can load elements from each sublist into memory without degrading CPU cache utilization.

Partitioning

Given an unsorted list, A, of n elements, we wish to partition A into k buckets $A_1, A_2, \ldots, A_k$ of roughly equal size such that each bucket is value-disjoint. That is, for any two elements $a \in A_i$ and $b \in A_j$, if $i < j$, then $a < b$. Partitioning involves (i) finding pivots for each bucket (k total); (ii) determining which bucket each element is in; and, (iii) moving each element into contiguous memory with other elements in the same bucket.

Research Highlights



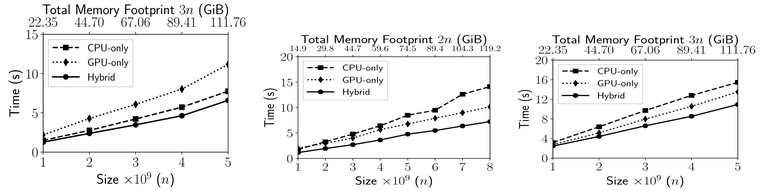
(a) Batched Predecessor Search, model: $f = 0.54$.

(b) Multiway Merge, model: $f = 0.47$. We use $k = 8$.

(c) Partitioning, model: $f = 0.31$. We use $k = 1024$ and $\mu = 32$.

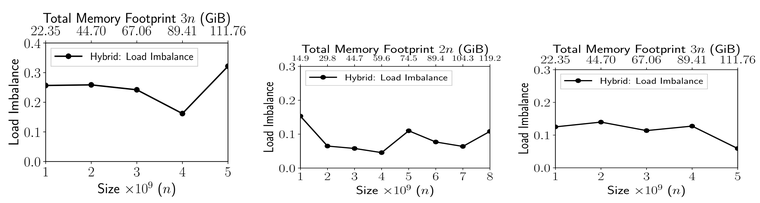
Figure 2. Hybrid model accuracy for all algorithms. The total response time, T^{CPU}, and T^{GPU} vs. f, are plotted. We show $n = 4.0 \times 10^9$ for all algorithms. The vertical dashed line in each plot denotes the modeled value of f.

- Figure 2(a) shows that even if the model can predict a good value for f, small differences in f can yield significant load imbalance for the batched predecessor search.
- Figure 2(b) demonstrates that the optimal value of f for multiway merging is almost exactly the value predicted by our model.
- Figure2(c) indicates that the model prediction is quite accurate, with the best performance achieved when $f = 0.35$ being only slightly faster than the predicted value of 0.3.



(a) (b) (c)

Figure 3. (a), (b), and (c) Response time vs. input size (n) comparing CPU-only, GPU-only, and hybrid batched predecessor, multiway merging, and partitioning algorithms, respectively.



(a) (b) (c)

Figure 4. (a), (b), and (c) The load imbalance of the hybrid algorithms in Figure 3(a), (b), and (c), respectively.

Future Work

- Investigating whether compression schemes or other memory transfer optimizations can alleviate some of the bottlenecks, despite the computational overhead.
- The study of other fundamental database operations that have not been considered for GPU acceleration.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant OAC-1849559 and Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.