Models and Algorithms under Asymmetric Read and Write Costs

Guy E. Blelloch[*], Jeremy T. Fineman[†], Phillip B. Gibbons[*], Yan Gu[*] and Julian Shun[‡]

[*]Carnegie Mellon University [†]Georgetown University [‡]U.C. Berkeley

Abstract—**In several emerging non-volatile technologies for main memory (NVRAM) the cost of reading is significantly cheaper than the cost of writing. Such asymmetry in memory costs leads to a desire for "write-efficient" algorithms that minimize the number of writes to the NVRAM. While several prior works have explored write-efficient algorithms for databases or for the unique properties of NAND Flash, our ongoing work seeks to develop a broader *theory* of algorithm design for asymmetric memories. This talk will highlight our recent progress on models, algorithms, and lower bounds for asymmetric memories [1], [2], [3]. We extend the classic RAM model to the asymmetric case by defining the (M, ω)-ARAM, which consists of a large asymmetric memory and a much smaller symmetric memory of size M, both random access, such that for the asymmetric memory, writes cost $\omega > 1$ times more than reads. We present algorithms for search trees, priority queues, sorting, dynamic programming, BFS- and DFS-related graph algorithms, and shortest paths. We also present novel lower bounds for several problems that reveal fundamental limits on obtaining write-efficient algorithms for these problems. Finally, we propose and study models for asymmetric memories that consider block transfers and/or parallel processing.**

I. Introduction

Fifty years of algorithms research has focused on settings in which reads and writes (to memory) have similar cost. But what if reads and writes to memory have significantly *different* costs? How would that impact algorithm design? What is the correct model to measure the cost of an algorithm on such memories? These questions are coming to the fore with the arrival of new non-volatile memory technologies, such as phase-change memory (PCM), spin-torque transfer magnetic RAM (STT-RAM), and memristor-based resistive RAM (ReRAM), which have the property that the cost of reading is significantly cheaper than the cost of writing, in terms of lower energy, lower latency, and higher (per-module) bandwidth. Such memories will appear in the marketplace in the first half of 2016 [4] as a block device, and after that as a DIMM off the processor's memory bus.

This talk highlights our recent results [1], [2], [3] providing a first step towards answering these fundamental questions about asymmetric memories. We introduce a simple model for studying such memories, and a number of new results.

Definition 1: We define the (M, ω)-*Asymmetric RAM (ARAM)*, comprised of a symmetric small-memory of size M, an asymmetric large-memory of unbounded size, and an integer write cost ω. The *ARAM cost* Q is the number of reads from large-memory plus ω times the number of writes to large-memory. The *time* T is Q plus the number of reads and writes to small-memory.

We focus on settings in which the problem input fits in the large-memory (and hence can be viewed as "unbounded" size) but not in the small-memory (i.e., $M \ll n$, where n is the input size and M is measured in units of the size of an input item). The large-memory models the NVRAM, while the small-memory models either DRAM, if present, or cache memory. The time T extends the time complexity on a RAM model to the asymmetric setting. The cost Q accounts for just the data transfers between the small-memory and the large-memory, charging ω for each write and 1 for each read. While T has the advantage of being most similar to the RAM model, it hides the performance gap between different levels of the memory hierarchy. Thus the Q metric may be more relevant in practice, as it focuses on reads and writes to the last relevant level of the memory hierarchy. We use a parameter ω to account for the higher cost of writes so that we can study the dependence of algorithms on the gap between writes and reads. We view ω as being significantly larger than 1, as factors up to 1–2 orders of magnitude have been reported in the literature (see [1] and the references therein).

Prior work focusing on asymmetric read and write costs for non-volatile memory has explored system issues (e.g., [5]), database algorithms (e.g., [6]) or the unique properties of NAND Flash (e.g., [7]) such as its limited endurance and large erase operation.

II. New Algorithms and Lower Bounds

Search Trees and Priority Queues. We consider the efficiency of balanced binary search trees on the (M, ω)-ARAM. Red-black trees with appropriate rebalancing rules require only $O(1)$ amortized time per update (insertion or deletion) once the location for the key is found. For a tree of size n, finding a key's location uses $O(\log n)$ reads but no writes, so the total amortized cost is $Q = T = O(\omega + \log n)$ per update. Since priority queues can be implemented with a binary search tree, insertion and delete-min have the same bounds. Note that these good bounds are obtained even for $M = O(1)$.

Sorting. Many classic sorting algorithms like quicksort and merge-sort require $\Theta(n \log n)$ reads and writes, and hence would cost $O(\omega n \log n)$ on the (M, ω)-ARAM. However, it is well known that sorting can be implemented by first inserting each key into a balanced search tree, and then in-order traversing the tree to write out the final sorted order. Using the above-mentioned binary search tree, the sorting algorithm based on this method requires only $O(n)$ writes with no increase in reads ($O(n \log n)$). This algorithm is asymptotically optimal, since comparison sorting n element requires at least $\Omega(n \log n)$ comparisons and n writes are necessary to produce the sorted output in contiguous memory. On the (M, ω)-ARAM, $Q = T = O(n \log n + \omega n)$, even for $M = O(1)$.

It is also interesting to consider sorting in parallel on a shared-memory multicore. In [1] we discussed a parallel version of the sorting algorithm, which has asymptotically the same numbers of reads and writes as the above sequential version. The algorithm is highly parallel, with a critical path length of only $O(\omega \log n)$.

Dynamic Programming. With regards to dynamic programming, some problems are reasonably easy and some harder. The standard Floyd-Warshal algorithm for the all-pairs shortest-path (APSP) problem uses $O(n^3)$ writes, and hence would cost $Q = T = O(\omega n^3)$. Kleene's divide-and-conquer algorithm can be used to reduce the ARAM cost. Each recursive call makes two calls to itself on problems of half the size, and six calls to matrix multiply over the semiring $(\min, +)$. Here we analyze the algorithm in the (M, ω)-ARAM. The matrix multiplies on two matrices of size $n \times n$ can be done in the model in $Q_M(n) = O(n^2(\omega + n/\sqrt{M}))$ [1]. This leads to the recurrence $Q_{Kleene}(n) = 2Q_{Kleene}(n/2) + O(Q_M(n)) + O(\omega n^2)$, which solves to $Q_{Kleene}(n) = O(Q_M(n))$ since the cost is dominated at the root of the recurrence. This is a factor of n fewer writes than the Floyd-Warshal algorithm. A similar approach can be used for several other problems, including sequence alignment with gaps, optimal binary search trees, and matrix chain multiplication [2].

Longest common subsequence and Edit distance are more challenging. If the answer is directly computed using the classic recursion formulation, the dependency structure of the computation is a **diamond DAG**, whose properties have been well-studied in the symmetric memory setting. For our asymmetric setting, we proved a new lower bound showing that the ARAM cost Q cannot be asymptotically reduced for the diamond DAG, even when only selectively storing intermediate values of the diamond DAG and allowing for recomputation of partial results—well-known techniques for reducing the number of writes. To overcome this fundamental limitation, we must violate the classic evaluation method for a DAG that requires a node's execution must wait until all its immediate predecessors in the DAG have executed. Instead, we show a new approach that performs partial accumulation at a node based on values computed at only *some* of its predecessors at a time, combined with selective recomputations, which overcomes this lower bound. Using this new technique, called a "path sketch", Q and T can be decreased by approximately a factor of $\omega^{1/3}$.

BFS, DFS, topological sort, biconnected components, strongly connected components. Breadth-first and depth-first search on graphs of n nodes and m edges can be performed in only $Q = T = O(\omega n + m)$. In particular, each vertex requires a constant number of writes when it is first added to the frontier (the stack or queue) and when it is finished (removed from the stack or queue). Searches along an edge to an already visited vertex require no writes. This implies that several problems based on BFS and DFS also require only $Q = T = O(\omega n + m)$, such as topological sort, biconnected components, and strongly connected components.

Single-source shortest-path. For directed graphs with non-negative edge weights, the most efficient single-source shortest-path algorithm in the classic RAM model is Dijkstra's algorithm. Unfortunately, this algorithm is not write-efficient because it keeps a global priority queue in the memory, and each update operation corresponds to at least one write, for a total cost of $O(\omega(m + n \log n))$. We proposed a new algorithm called "phased Dijkstra" [2] such that the priority queue is limited to only $O(M)$ elements so that it fits in the small-memory. To achieve this, the algorithm must use n/M phases, resulting in $Q = T = O(n(\omega + m/M))$. For certain settings, namely, when $M < n < M\omega$, this improves upon classic Dijkstra.

Lower bounds for FFT and sorting networks. We have new lower bounds for FFT and for sorting networks on the (M, ω)-ARAM. As with the diamond DAG lower bound discussed above, our lower bounds hold for these computation DAGs (the classic DAG for FFT, any possible DAG of pairwise comparators that sorts for sorting networks), even when only selectively storing intermediate values of the DAG and allowing for recomputation of partial results. Under the reasonable assumption in practice that $M \geq \omega$ (recall that M is a scalar measured in units of an input data item size), our bounds show that it is *not possible* to have an algorithm that is an asymptotic cost improvement over the best algorithms for a two-level *symmetric* memory model. I.e., one cannot take advantage of reads being cheaper than writes by a factor of ω, resulting in $T = \Theta(\omega n \log n / \log(M))$ on the (M, ω)-ARAM. Interestingly, when comparing this lower bound to the $O(n \log n + \omega n)$ upper bound for'the above comparison sorting algorithm, we see a *large asymptotic gap* between sorting networks and more general comparison sorting, in contrast to classic RAM results, for which the two have the *same* asymptotic complexity.

III. Other Models and Results

"External" memory sorting. We considered explicitly modeling the transfer of data between the two memories in units of size $B > 1$. We adapt the well-studied External Memory model (a.k.a., the I/O model or the Disk model used in databases) to charge ω times more for writing a block of size B to the asymmetric memory than for reading such a block. We examine the three general approaches for sorting on the External Memory model—multi-way mergesort, sample sort, and buffer-tree sort—and showed that each of these algorithms can be adapted to use fewer writes. Interestingly, each of the original algorithms have the same optimal cost on the External Memory model, and we show that each can be adapted to the asymmetric setting to have the same $Q = O(\omega n/B \log_{\omega M/B}(n/B))$ cost. While the new multi-way mergesort algorithm is similar to prior work [6] and the new samplesort algorithm is readily achieved, the new buffer-tree sort requires a number of new techniques [1].

Cache models. It is also interesting to understand how the cache models and cache policy are affected by read-write asymmetry. More specifically, we extended important results for the Ideal-Cache model and thread schedulers to the asymmetric case—namely, the Asymmetric Ideal-Cache that we proposed can be (constant factor) approximated by an asymmetric-LRU cache, and it can be used in conjunction with a work-stealing (parallel-depth-first) scheduler to obtain good parallel cache complexity bounds for machines with private caches (a shared cache, respectively). We used this model to design write-efficient cache-oblivious parallel algorithms.

IV. Conclusion

This talk highlights our recent work providing a first step towards understanding the questions about algorithm design and computational models under asymmetric read and write costs. We introduced a simple model called the (M, ω)-ARAM for studying such memories, and new algorithms and lower bounds for a number of classic problems, as well as extensions to the model. We hope that our work inspires more creative algorithmic research on this topic, in conjunction with the forthcoming emergence of non-volatile memory technologies (such as the Intel/Micron 3D XPoint [4]) that will enable experimental studies of the proposed algorithms.

References

[1] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, and J. Shun, "Sorting with asymmetric read and write costs," in *ACM SPAA*, 2015.

[2] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, and J. Shun, "Efficient algorithms with asymmetric read and write costs," *arXiv preprint*, arXiv:1511.01038.

[3] N. Ben-David, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, C. McGuffey and J. Shun, "Parallel algorithms with asymmetric read and write costs," in *ACM SPAA*, 2016.

[4] http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology, 2015.

[5] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *ACM ISCA*, 2009.

[6] S. Viglas, "Write-limited sorts and joins for persistent memory," *PVLDB*, vol. 7, no. 5, 2014.

[7] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, no. 2, 2005.