RETROSPECTIVE: GraFboost: using accelerated flash storage for external graph analytics

Sang-Woo Jun*, Andy Wright†, Sizhuo Zhang†, Shuotao Xu†, and Arvind†
* Computer Science, University of California, Irvine.
swjun@ics.uci.edu
† Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
acwright@mit.edu, {*szzhang,shuotao,arvind*}*@csail.mit.edu*

The primary goal of GraFBoost, as well as many of our subsequent efforts, is to reduce the cost of large-scale data processing. We focus on hardware acceleration of out-of-core algorithms, replacing costly and power-hungry DRAM with cheap and efficient solid-state storage such as NAND Flash, without loss of performance. For GraFBoost, we targeted graph analytics because it is an egregious example of sparse and irregular data accesses, challenging ourselves to overcome one of the worst applications for out-of-core performance.

I. GraFBoost Overview

GraFBoost is a scalable out-of-core graph analytics platform which addresses the issue of randomly accessing algorithmic state of vertices in storage, during graph algorithm execution. If the capacity of data stored at each vertex is small relative to the size of a storage page, random accesses into the array of vertex data incurs heavy I/O amplification. For example, vertices during a shortest path algorithm only need to remember the current shortest distance to itself and the previous vertex in the path. This only requires, say, 8 bytes, which is minuscule compared to a 4 KB page.

Underneath, GraFBoost employs two noteworthy innovations to efficiently address this issue: The **(1) Sort-Reduce** algorithm, and **(2) near-storage FPGA acceleration**.

The ***Sort-Reduce*** algorithm efficiently sequentializes random updates into an array in secondary storage by first sorting a large number of requests before applying them ("*Sort*"). It reduces sorting overhead by merging pairs of consecutive update requests to the same index, whenever they are found during sorting ("*Reduce*"). This requires that the updates be expressible as an associative binary function (e.g., *add*).

GraFBoost is implemented as a ***near-storage FPGA accelerator***, to minimize the performance impact of the additional sort-reduce operation. Near-storage accelerators can use the high internal bandwidth of multiple storage devices without being limited by the host-side PCIe bandwidth.

Both approaches showed great benefits individually as well as together. The sort-reduce approach of interleaving reduction with sorting was able to typically reduce the amount of storage access by over 90%, resulting in even a purely software implementation of GraFBoost to outperform existing out-of-core software on large graphs, i.e., graphs exceeding the available memory capacity. A prototype accelerator on a Xilinx VC707 FPGA and 1 TB of NAND-flash storage was able to further improve performance by almost 2×. The prototype sort-reduce accelerator was able to process 4 GB/s per processing element, which benefits from the near-storage configuration because it exceeds the PCIe bandwidth of the prototype. It also almost eliminates the host processor and memory resource requirements, resulting in superior performance at a quarter of cost.

II. Insights Learned Since

Since the publication of GraFBoost, we have continued to explore accelerated out-of-core algorithms as a solution to scalability issues for a variety of applications.

One important discovery is that sort-reduce is a versatile solution to the out-of-core random access issue for many applications beyond graphs. The only restriction sort-reduce adds to generic array updates is that the update function has to be associative. This restriction is easily satisfied for many applications of interest, including sparse matrix multiplication for graph neural networks, constructing De Bruijn graphs from genome reads [79], hash table construction [81], and database joins [83].

Another interesting discovery is that, especially with hardware acceleration, the overhead of sort-reduce is low enough to improve the performance of even in-memory algorithms. Despite the name, DRAM can actually suffer an order of magnitude performance degradation with fine-grained random accesses compared to sequential. There are multiple reasons for this, including the overhead of opening a new bank and loading a new row into the row buffer, as well as minimum burst lengths and interface width. In our experience with bloom filters, the performance benefits of a sort-reduce accelerator was enough to raise the performance of a DDR3 module beyond a Hybrid Memory Cube directly handling random updates [81].

We also argue that the configuration of near-storage *reconfigurable* acceleration as employed by GraFBoost, is the desirable architecture for future out-of-core analytics systems. First, accelerators are necessary to minimize the performance impact of the additional sort-reduce operation. This means the user-defined update function also must be implemented in the accelerator to avoid the communication bottleneck. Second, the accelerator should be positioned near-storage to avoid the

bandwidth limitations of PCIe relative to storage performance. Even with reduced I/O amplification brought by sort-reduce, we noticed that still the most significant performance bottleneck is the storage performance. The storage system performance can readily be improved with more devices, in theory even to DRAM levels. But without near-storage acceleration, actual attainable performance quickly becomes limited by the host-side PCIe bandwidth. We note that even with near-storage acceleration, the I/O capabilities of individual accelerators can become a limiting factor at such high bandwidth. We describe a potential architectural solution in the following section.

Another observations is that in reality, out-of-core accelerator systems likely do not need to provision enough raw storage bandwidth to match the DRAM module it is aiming to replace. As long as I/O amplification is kept to minimal with approaches such as sort-reduce, we notice that the out-of-core system can outperform even fully in-memory systems with higher raw memory bandwidth, as long as the out-of-core system is employing accelerator optimizations such as efficient data-specific compression schemes, accelerating computation-intensive phases of execution, as well as reduced intermediate memory operations via a deep hardware pipeline [80], [84].

On the other hand, we also discovered a subset of graph applications which are not handled effectively with the GraFBoost approach. The first category of applications include those with relatively large vertex data structures, which would cause low I/O amplification when accessed directly. Examples we encountered include graph neural networks with large feature maps per vertex (small feature maps were very efficient with sort-reduce). For these examples, the I/O amplification from page-granuarity access is lower than that of making multiple sorting passes for sort-reduce. The solution to this issue would simply be to switch modes in such cases from sort-reduce to conventional in-memory caches, as we will describe in the next section. The second, rarer category of applications which are ineffective with sort-reduce, are those with complex, non-associative update functions. Examples we encountered include subgraph isomorphism with large query graphs. While hardware-accelerated sorting without reduction still results in significantly lower I/O amplification, these applications did not enjoy the order of magnitude storage access reduction enjoyed by other applications.

III. Moving Forward

Based on our success with GraFBoost and subsequent system designs, we have identified the additional advancements necessary for a general-purpose graph analytics system based on these ideas.

First, the system must be able to handle large, and/or variable-length data in each vertex. As mentioned in the previous section, large vertex data can be simply handled by excluding them from sort-reduce and instead applying them directly to storage, coupled with conventional in-memory caching. Variable-length vertex data requires an additional indirect index structure which can also be out-of-core, but its overhead is not expected to be high. This is because after sort reduce, index access would always be in an efficient, monotonically increasing pattern during updates.

Second, the near-storage accelerator must be able to handle the DRAM-level aggregate bandwidth of a large number of storage devices. However, individual accelerator packages will likely have bandwidth limitations due to pin count, while fast memory interfaces such as silicon interposers and 3D stacking are not feasible at this scale due to the sheer number of storage devices required to reach the desirable bandwidth. As a result, our target design will involve multiple near-storage accelerators communicating with an array of storage devices. For efficient use of distributed storage and its bandwidth by the near-storage accelerator, without being limited by host-side PCIe bandwidth, the accelerators will need to network over a non-blocking N-to-N fabric. A good choice may be a PCIe switch for P2P communication similar to the choice made by Samsung SmartSSD [82], or a sideband mesh network like the NVIDIA NVLink or BlueDBM [72].

Once the out-of-core analytics accelerator becomes as powerful as we envision above, it will become necessary to make this resource elastically allocatable and composable, to minimize resource fragmentation and achieve even better cost-effectiveness in a cloud environment. We expect low-latency communication fabric such as CXL to be extremely useful in facilitating this goal.

IV. Conclusion

While we are currently mainly focusing on NAND-flash fabric due to their cost-effective availability, emerging cost-effective memory technologies including PCM, STT-MRAM, and FeRAM all have similar restrictions in terms of page size and latency limitations. In light of these trends, coupled with the ever-increasing requirement for data collection and processing, we expect near-storage data management accelerators akin to sort-reduce and GraFBoost to become a critical component of future systems for the continued scalability of large, data-intensive applications.

References

[79] N. Cadenelli, S.-W. Jun, J. Polo, A. Wright, D. Carrera, and Arvind, "Enabling genomics pipelines in commodity personal computers with flash storage," *Frontiers in genetics*, vol. 12, p. 615958, 2021.

[80] S. Kang, J. An, J. Kim, and S.-W. Jun, "Mithrilog: Near-storage accelerator for high-performance log analytics," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 434–448.

[81] S. Kang, T. S. G. Nerella, S. Uppoor, and S.-W. Jun, "Bunchbloomer: Cost-effective bloom filter accelerator for genomics applications," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 9–16.

[82] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "Smartssd: Fpga accelerated near-storage data analytics on ssd," *IEEE Computer architecture letters*, vol. 19, no. 2, pp. 110–113, 2020.

[83] G. Sun and S.-W. Jun, "Columnburst: a near-storage accelerator for memory-efficient database join queries," in *roceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2020.

[84] G. Sun, S. Kang, and S.-W. Jun, "Burstz: a bandwidth-efficient scientific computing accelerator platform for large-scale data," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.