Accelerating BFS and SSSP on a NUMA machine for the Graph500 Challenge

Tanuj Kr Aasawat
Kazuki Yoshizoe
{tanujkr.aasawat,kazuki.yoshizoe}@riken.jp
RIKEN, Japan

Tahsin Reza
Matei Ripeanu
{treza,matei}@ece.ubc.ca
University of British Columbia, Canada

1 INTRODUCTION

Modern shared-memory systems embrace the NUMA architecture which has proven to be more scalable than the SMP architecture. The NUMA architecture is a representative of increasing memory hierarchy in recent shared-memory systems and the move towards vertical scaling. In many ways, a NUMA system resembles a shared-nothing distributed system: physically distinct processing units and memory regions. Memory memory accesses to remote NUMA domains are more expensive than local accesses.

Graphs are notorious for having data-dependent, highly irregular memory accesses, which amplifies the short coming in NUMA machines - expensive remote memory accesses. In this work we explore how data locality and communication minimization can be achieved by exploiting "distributed" shared-memory of NUMA machines to develop shared-memory graph processing solutions optimized for NUMA systems.

Our designs exploit the distributed "shared" memory nature of NUMA architecture. We introduce a novel hybrid design for memory accesses that handles the burst mode in traversal based algorithms, like BFS and SSSP, and reduces the number of remote accesses and updates. All the designs and optimizations are generic, independent of the graph algorithms, and are part of our NUMA-aware graph processing framework.

Through comprehensive experimentation, using the Graph500 R-MAT graphs, we demonstrate that our designs offer up to 94% speedup over our NUMA-oblivious framework Totem [5] and 2.5× over shared-nothing distributed design, for BFS and SSSP.

2 DESIGN OVERVIEW OF THE NUMA-AWARE COMMUNICATION SUBSTRATE

Our system adopts the Bulk Synchronous Parallel (BSP) processing model. It creates explicit partition for each NUMA domain and enables communication between remote partitions though dedicated message buffers. Each partition has a set of local vertices and edges. In the BSP model, processing consists of sequence of rounds or supersteps. Each superstep consists of three phases (executed in order): computation, communication, and synchronization. This sequence of supersteps continues until all the processing units have finished processing their respective partitions. Finally, the result is aggregated from all the partitions.

We partition the graph and place one partition per NUMA node, which enables us to process partitions in parallel (with the opportunity to serve all the memory accesses from the local partition). For inter-partition communication, since NUMA systems are essentially a shared-memory system, we have the opportunity to explore optimization techniques to reduce the communication overhead. Following are the three designs we explore:

2.1 2-Box Design

Here we fully embrace the design philosophy of a shared-nothing distributed system. It has two message buffers: outbox and inbox. In the computation phase, each partition updates its local state buffer for local vertices; while updates for remote vertices are *aggregated and stored locally* in respective outboxes. In the communication phase, each partition transfers the contents of respective outbox to the corresponding remote inbox , and applies the remote updates from the inbox to the local state buffer.

The **advantages** of this design are: (i) *Zero remote memory accesses* in the computation phase and the inter-partition copying of message buffer is explicit, mostly sequential accesses, which enables better bandwidth utilization, and (ii) *Message aggregation* for the remote vertices is done locally which results in *only one message per remote vertex during the communication phase, regardless of the number of remote edges.* This drastically decreases the inter-partition traffic and the communication time. The **limitation** of this approach is high *communication overhead.* Since remote vertices are marked and counted during the partitioning step, the size of the message buffers remain unaltered during algorithm execution. Though *message aggregation* reduces the number of messages sent, there is still communication overhead when the message buffers have few vertices, which is often the case for the majority of the supersteps for algorithms like BFS and SSSP, where in each superstep, communication happens over only a selected set of edges.

2.2 0-Box Design

In this design, we consider the fact that NUMA is a shared-memory system. We do explicit partitioning as if NUMA is a distributed system, but we access the state buffers as if we are in a shared-memory system. During computation, if a remote vertex is visited, the value is updated directly in the local state buffer of the respective remote partition, thereby *it overlaps computation with communication.* To ensure consistency, atomic operation is used to update shared states. Note that, in this design, the state of *all the boundary edges* is accessed remotely.

Since this design *overlaps computation with communication*, it **performs better** for algorithms like Direction-optimized BFS and SSSP, where communication happens only via a selected set of edges in every superstep. However, for algorithms like PageRank and BFS Top Down, where the communication density is high, we do not see any performance gain. *Lack of message aggregation* increases the number of remote memory accesses severely.

2.3 Hybrid Design

The idea behind this design stems from a number of key observation: In the initial few supersteps in traversal based algorithm, the message density will be very high (known as burst mode) and the frontier size increases drastically. This leads to costly remote memory accesses, as there is no message reduction in 0-Box. While 2-Box aggregates the messages and sends only one message per vertex (not per boundary edge, unlike 0-Box; note: average degree in R-MAT graphs is 16); (it, however, has the overhead of explicit communication phases.) For traversal based algorithms, like SSSP, 0-Box activates the neighbors in remote partition even in the first superstep, leading to better load balancing, having many active vertices in the same iteration.

Therefore, to leverage the benefits of both 2-Box and 0-Box, we introduce a novel Hybrid design, which starts execution in 0-Box mode (to have active vertices across partitions); switches the execution to 2-Box mode during burst mode; and finally executes the remaining supersteps in 0-Box mode to hide the communication cost. This leads to reducing the time spent in the burst mode, and offers major gain in performance.

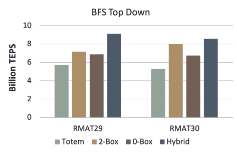
3 EXPERIMENTS AND DISCUSSION

We evaluate our NUMA-aware communication approaches on *Breadth-first Search* (both the classic level-synchronous Top Down and Direction-optimized by Beamer et al. [3]) and *Single-source Shortest Path* (similar to the Bellman Ford algorithm) - both are the kernels used in Graph500 to benchmark supercomputers worldwide.

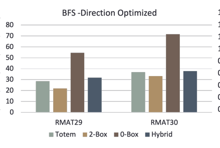
We use R-MAT [4] Scale 29 and 30 graphs, generated following Graph500 standards, with an average vertex degree of 16. For SSSP, which requires weighted graph, we generate weights randomly between (0,100]. We run our experiments on a *four socket Intel Xeon machine* (E7-4870 v2; #cores: 60; Memory: 1536 GB; LLC: 120 MB).

Fig. 1 presents the performance of all the designs for the three algorithms. Our Hybrid design performs up to 94% faster than Totem. In our previous study [2], we observed that 2-Box performs best for algorithms with very high message density, like PageRank. While, our 0-Box performs best for algorithms with extremely low message density, like Direction-optimized BFS. *Our Hybrid designs improves the performance of the algorithms, like BFS Top Down and SSSP, which has very high message density during initial few supersteps (burst mode) and for rest it has low message density* - often the case with many traversal based algorithms. Further, compared to our previous study [2], with our new design we are up to 8.6× faster than state-of-the-art NUMA-aware framework Polymer [6] for the mentioned workload and graph algorithms.

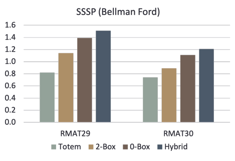
For BFS-Top Down and SSSP, we execute initial supersteps in 0-Box mode, to have many active vertices across partitions (first superstep in Fig. 2), and once the frontier size increases drastically (i.e., in the burst mode) we switch to the 2-Box mode. This helps in doing all the accesses/updates locally, as remote accesses/updates are expensive. During the communication phase, it sends the outbox buffer which has one message per vertex, because of message aggregation in the outbox. Note that during the burst mode, 0-Box ends up doing remote accesses/updates = number of active boundary edges. As shown in Fig. 2, during second superstep, even though



(a) BFS-TD



(b) BFS-DO



(c) SSSP

Figure 1: Billion Traversed Edges Per Second (TEPS) achieved by Totem vs. the NUMA-aware communication designs. Note, the Y-axis is for TEPS (higher the better). In [1], the authors have shown Totem out performs other well known frameworks such as Galois, Gunrock and Graphmat

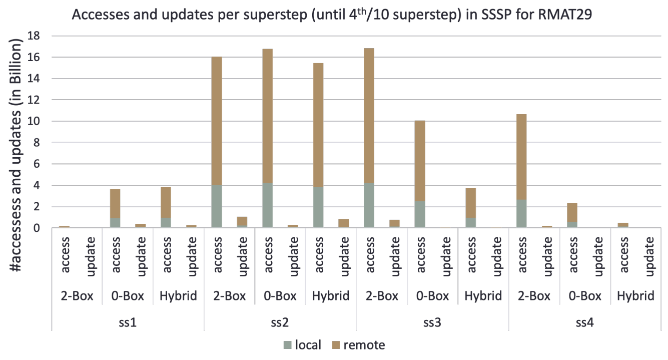


Figure 2: All (local and remote) accesses (reads) and updates (writes) in SSSP in computation phase for RMAT-29 graph. The X-axis is for supersteps. We show only for first four supersteps out of total ten supersteps, as most of the accesses happen in these supersteps. NOTE: Remote access/update in 2-Box points to outbox which is allocated locally - so essentially all the accesses are local in 2-Box.

0-Box does very few remote updates, it accesses more number of remote/boundary edges, hence is slower.

Since Direction-optimized BFS (BFS-DO) has extremely low message density, 2-Box design ends up sending almost empty buffer and checks for updates, if any. This leads to the communication phase being the most expensive. As Bottom-up phase is the most expensive in BFS-DO, we execute 2-Box mode during Bottom-up phase. But since 2-Box suffers from extremely high communication cost, Hybrid design doesn't have any benefit over 0-Box design.

4 GRAPH500 CHALLENGE

In the Graph500 challenge, we rank First among single node submissions in "small dataset" category (graphs with size at least Scale 32/1 TB/128 Billion directed edges) for SSSP kernel. The performance improvement will help us stay among the top single node submissions for BFS and SSSP kernel in the Graph500 challenge (list will release at SC'19). So far, we have been able to submit for graphs with 512 billion directed edges (Scale 34, edge-list size: 4 TB) on our NUMA machine (at RIKEN).

REFERENCES

[1] T. K. Aasawat, T. Reza, and M. Ripeanu. 2018. How Well do CPU, GPU and Hybrid Graph Processing Frameworks Perform?. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 458–466.

[2] T. K. Aasawat, T. Reza, and M. Ripeanu. 2018. Scale-Free Graph Processing on a NUMA Machine. In *2018 IEEE/ACM 8th Workshop on Irregular Applications:*

Architectures and Algorithms (IA3). 28–36.

[3] Scott Beamer, Krste Asanović, and David Patterson. 2012. Direction-optimizing Breadth-first Search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*. Article 12, 10 pages.

[4] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the Fourth SIAM Int. Conf. on Data Mining*. Society for Industrial Mathematics, p. 442.

[5] Abdullah Gharaibeh, Lauro Beltrão Costa, Elizeu Santos-Neto, and Matei Ripeanu. 2012. A Yoke of Oxen and a Thousand Chickens for Heavy Lifting Graph Processing. In *Proc. of the 21st Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT '12)*. 345–354.

[6] Kaiyuan Zhang, Rong Chen, and Haibo Chen. 2015. NUMA-aware Graph-structured Analytics. In *Proc. of the 20th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP 2015)*. 183–193.