Distributed Algorithms for Aligning Massive Networks

Mathematically, the Network Alignment (NA) problem can be defined as follows: Let $A = (V_A, E_A)$ and $B = (V_B, E_B)$ be two graphs with vertex set V and edge set E. Let L be the weighted bipartite graph $L = (V_A \cup V_B, E_L, w)$ derived from A and B with weight function $w : E_L \rightarrow \mathbf{R}^+$. Let α and β be two positive constants. Together A, B, L, α, and β form the input. A matching M_L in L is a subset of edges $M \subset E_L$ such that no two edges in M_L are incident on the same vertex. The weight of a matching is the sum of the weights of edges in M. We say that an edge (i, j) in E_A is *overlapped* with an edge (i', j') in E_B only if both the edges (i, i') and (j, j') are in M_L. The network alignment problem is to find a matching M_L that maximizes:

$$\alpha \times \text{weight of the matching} + \beta \times \text{number of overlapped edges in A and B.} \quad (1)$$

Network alignment is an NP-hard with no known approximation algorithms. We presented heuristic algorithms for alignment in [1]. We refer you to this work for further details. Here, we will only provide the basic intuition of the algorithms, and discuss their limitations which motivated our current work. We focus on the belief propagation algorithm, which is iterative in nature. There are two phases in each iteration: (i) Sparse matrix computations related to belief propagation, and (ii) computation of a maximum weight bipartite matching in L.

Our current work is motivated by the unwieldy memory requirements of the belief propagation algorithm, which is given by $O(5 * |V_A| + 5 * |V_B| + 15 * |E_L| + 5 * |E_S|)$. Where, E_S represents the number of overlapped edges in A and B. To illustrate the requirements consider one of our inputs where $|V_A| = |V_B| \approx 3$ million. This requires at least 110 GB of memory when L is sparse and $|E_L| \approx 60$ million. When L is a complete bipartite graph then the requirement grows over 6 TB of memory.

While distributed-memory platforms provide a means for developing scalable implementations and the ability to solve large problems, the combination of sparse matrix kernels and graph algorithms make efficient parallel implementations extremely challenging. The matrix operations involve representations of S and L (a bipartite graph can be interpreted as a sparse matrix). The structure of these two matrices remain constant, but the values change based on the computation of matching in the second phase. Spatial and temporal locality gets destroyed by the alternating matrix and graph kernels. Memory accesses are irregular making it hard to prefetch or distribute data efficiently. In order to

Problem	Memory	2	4	8	16	32	64
LargeSyn	110GB	7910.68	4359.3	2283.86	1370.26	904.33	790.12
dmela-homo	150GB	NA	13333.0	10824.25	10505.40	9068.85	8639.20
scere-homo	190GB	NA	17412.1	15425.80	13989.85	12437.50	10469.45

Table 1: Strong Scaling

Problem	**OpenMP** (HW locks)	**OpenMP** (`omp_locks_t`)	**GMT** (`gmt_locks`)
dmela-scere	2.10	6.33	76.67
homo-muso	3.29	8.23	69.88

Table 2: Comparing between shared memroy and distributed implementation.

simplify implementation and exploit dynamic optimization techniques we built our application on a runtime system designed for irregular applications.

GMT (Global Memory and Threading library) is a new runtime library developed at the Pacific Northwest National Laboratory. GMT is based on the Partitioned Global Address Space (PGAS) model and targeted for irregular applications with large memory requirements. Techniques such as multilevel aggregation and lightweight multithreading are used to maximize memory and network bandwidth and tolerate latencies arising from memory accesses and communication. GMT has specialized threads for computation (worker threads) and internode communication (helper threads). We implemented the network alignment algorithms using memory allocation and parallel loop constructs in GMT in a spirit similar to shared-memory programming model (OpenMP) that resulted in a dramatic programmer productivity.

The experiments are run on a compute cluster (Olympus) at PNNL consisting of 604 nodes with each node has 64 GB of DDR3 memory and nodes communicate through a QDR Infiniband switch. All global data is distributed equally among the participating nodes. GMT handles global memory allocation, and data movement exploiting aggregation where possible. We solve problems with $> 110GB$ which we could not solve before with shared memory implementation. Strong scaling profiles of these problems are shown in Table 1.

We ran smaller problems to understand the overheads of the distributed implementation in Table 2. We observe that distributed implementation is actually $8 - 15$x slower which is mostly due to the internode communication. We notice that achieving parallel efficiency on distributed-memory platforms remains a hard challenge. However, by exploiting a specially designed runtime system we demonstrate scalable performance and programmer productivity. We plan to perform several optimizations in the near future and hope to solve many network alignment problems of interest that are currently unsolvable.

References

[1] Khan, A. M., Gleich, D. F., Pothen, A., and Halappanavar, M. A multithreaded algorithm for network alignment via approximate matching. In *SC'12* (2012), pp. 64–64.