

# MATCHINGS IN MASSIVE GRAPHS ON TERASCALE COMPUTERS VIA APPROXIMATION

Mahantesh Halappanavar, Florin Dobrian, and Alex Pothen

*Introduction:* A matching  $M$  in a graph is a set of edges such that each vertex is incident on at most one edge in  $M$ . A maximum weight matching is a matching that maximizes the sum of the weights of the matched edges. We describe the implementation of a parallel approximation algorithm for computing a maximum weighted matching on massive graphs (with several hundred million edges) and present results on thousands of processors on massively parallel terascale computers. To the best of our knowledge, this is the first massive scalability study for an approximate matching algorithm.

Matching is a fundamental combinatorial problem that has applications in many contexts: high-performance computing, bioinformatics, network switch design, web technologies, etc. Examples in the first context include sparse linear systems, where matchings are used to place large matrix elements on or close to the diagonal, block triangular decomposition, computing a sparse basis for the null space or column space of under-determined matrices, and multi-level graph partitioning algorithms where matchings are used in the coarsening phase.

*Approach:* Algorithms for computing optimal matchings are inherently sequential and have poor locality properties that make the design of parallel algorithms difficult. Therefore, we focus our attention on approximation algorithms. A half-approximation algorithm can be computed by repeatedly matching the currently heaviest edge and removing the edges that are incident on it. However, this approach processes edges in a global order and is therefore inherently sequential. In a pointer based approach, each vertex maintains a pointer to its neighbor along a heaviest edge. If two vertices point to each other then the corresponding edge is a *locally dominant edge*, which means that it is heavier than the edges incident on its endpoints. The algorithm proceeds by repeatedly matching locally dominant edges, removing edges incident on them, and by adjusting the pointers of unmatched vertices to point to a current heaviest neighbor. It is important to break ties in a consistent fashion to avoid deadlock. The pointer based approach is more amenable to parallelism and is the approach we adopt.

The input graph is distributed and stored in a vertex-oriented fashion where each processor owns a subset of vertices, and stores the adjacency lists and weights of the vertices and edges in the subgraph induced by the vertices it owns. Each processor also stores the identities of the endpoints of the cut edges it does not own (ghost vertices) and the identities of the processors that own them. Each processor runs the pointer based algorithm. For an interior vertex  $v$ , all of whose neighbors are owned by one processor, a processor can compute the pointer to a heaviest neighbor of  $v$  as in the sequential algorithm. For a boundary vertex  $v$  a heaviest neighbor could be a ghost vertex  $w$ , the other endpoint of a cut edge  $(v, w)$ . To determine if  $w$  points to  $v$  as its heaviest neighbor, the first processor sends a request message to the second processor. If the second processor also sends a request message to the first asking to match  $w$  to  $v$ , then the cut edge  $(v, w)$  is added to the matching; if  $w$  does not point to  $v$  but is matched to another neighbor, then the second processor sends an unavailable message to the first. In this case,  $v$  resets its pointer to its next heaviest unmatched neighbor. Once a vertex is matched, it is deleted from the adjacency lists of its

neighbors, and the latter reset the pointers to their heaviest unmatched neighbors if needed. The algorithm proceeds in this manner until all the vertices are either matched or fail to find unmatched neighbors. Messages sent by one processor to another can be bundled, and is critical for better performance of an implementation of this algorithm.

*Results:* The experimental work was conducted on three systems: (i) a Cray XT-4 system at NERSC with 38,640 cores, (ii) an IBM BlueGene-P system at Argonne National Laboratory with 163,840 cores, and (iii) a SiCortex SC5832 system at Purdue University with 4536 cores. We experiment with different classes of graphs - model problems based on five-point grids, graphs from different applications in science and engineering, and different types of synthetic graphs. Experiments from serial implementations show that the pointer based algorithm computes matchings of high quality in terms of the weight as well as the cardinality relative to optimal algorithms and other half-approximation algorithms. Also, the run times are very small, usually a fraction of a second for large instances of graphs.

A strong scaling study on circuit modeling problems from the University of Florida Sparse Matrix Collection reveals good scaling on all the three systems. For 4096 processors of the SiCortex, we observed a speedup of 732 for one of these problems (Freescale). This can be attributed to the existence of good edge separators. For small numbers of processors the Cray XT-4 is faster but, as the number of processors increases, the BlueGene-P provides a faster implementation. This can be attributed to the relatively small size of the problems (tens of millions of edges). The SiCortex is targeted for performance per unit of power consumed rather than raw performance, and has slower processors and an innovative interconnect. However, at two thousand processors it is comparable to the XT-4 in execution time, and is about four times slower than the BlueGene-P for one of the problems (G3).

In a strong scaling study on the BlueGene-P using a 16,000 x 16,000 grid having 256 million vertices and about 500 million edges, we observe efficiency equal to 50% when 2048 processors are used with about 16% of edges cut. For larger numbers of processors, efficiency is lower due to the cut sizes that grow to about 60% for 8192 processors.

In a weak scaling study on the XT-4 and BlueGene-P using grid graphs, the graph sizes are chosen in proportion with the number of cores, and quadruples as the number of cores roughly doubles. The largest input solved on 10,000 cores is a graph with 400 million vertices and 800 million edges, and had a runtime of 0.36 seconds on the BlueGene-P and 0.55 seconds on the XT-4. We expect the run time to double as the number of processors double due to the way we have scaled the problem, and the execution times agree with these predictions. For the communication-dependent phase of the algorithm the execution on the XT-4 is faster than the BlueGene-P for runs with less than 5000 cores, but it is slower for runs with greater than 5000 cores.

*Conclusions:* We demonstrate scalable results on tens of thousands of processors for classes of graphs that can be partitioned into subgraphs such that few edges are cut by the partition. In practice, the matchings computed have more than 90% of the weight of an optimal matching. By carefully analyzing the runtime we show how different parameters affect the runtime and when algorithmic techniques such as speculation will be helpful.

We expect to provide runs on larger numbers of cores and larger-sized problems by the conference.

The current work marks only the beginning of the development of parallel matching algorithms on terascale and petascale computers. Improved results could be achieved by bundling messages more aggressively, and by other algorithmic approaches. Developing parallel matching algorithms for graphs without good separators is an important direction to consider in the future.