Parallel Community Detection for Massive Graphs

E. Jason Riedy, Henning Meyerhenke, David Ediger, and David A. Bader

Georgia Institute of Technology

Abstract. Tackling the current volume of graph-structured data requires parallel tools. We extend our work on analyzing such massive graph data with the first massively parallel algorithm for community detection that scales to current data sizes, scaling to graphs of over 122 million vertices and nearly 2 billion edges in under 7300 seconds on a massively multithreaded Cray XMT. Our algorithm achieves moderate parallel scalability without sacrificing sequential operational complexity. Community detection partitions a graph into subgraphs more densely connected within the subgraph than to the rest of the graph. We take an agglomerative approach similar to Clauset, Newman, and Moore's sequential algorithm, merging pairs of connected intermediate subgraphs to optimize different graph properties. Working in parallel opens new approaches to high performance. On smaller data sets, we find the output's modularity compares well with the standard sequential algorithms.

1 Communities in Graphs

Graph-structured data inundates daily electronic life. Its volume outstrips the capabilities of nearly all analysis tools. The Facebook friendship network has over 500 million users each with an average of 130 connections [8]. Twitter boasts over 140 million new messages each day [23], and the NYSE processes over 300 million trades each month [19]. Applications of analysis range from database optimization to marketing to regulatory monitoring. Global graph analysis kernels at this scale tax current hardware and software architectures due to the size *and* structure of typical inputs.

One such useful analysis kernel finds smaller communities, subgraphs that locally optimize some connectivity criterion, within these massive graphs. We extend the boundary of current complex graph analysis by presenting the first algorithm for detecting communities that scales to graphs of practical size, over 120 million vertices and nearly two billion edges in less than 7300 seconds on a shared-memory parallel architecture with 1 TiB of memory.

Community detection is a graph clustering problem. There is no single, universally accepted definition of a community within a social network. One popular definition is that a community is a collection of vertices more strongly connected than would occur from random chance, leading to methods based on modularity [16]. Another definition [21] requires vertices to be more connected to others within the community than those outside, either individually or in aggregate. This aggregate measure leads to minimizing the communities' conductance. We consider disjoint partitioning of a graph into connected communities guided by a local optimization criterion. Beyond obvious visualization applications, a disjoint partitioning applies usefully to classifying related genes by primary use [25] and also

to simplifying large organizational structures [13] and metabolic pathways [22]. We report results for maximizing modularity, although our implementation also supports minimizing conductance.

Contributions. We present the first published parallel agglomerative community detection algorithm. Our algorithm scales to practical graph sizes on available multithreaded hardware but with the same sequential operation complexity as current state-of-the-art algorithms. Our approach is both natively parallel and simpler than most current sequential community detection algorithms. Also, our algorithm is agnostic towards the specific criterion; any criterion expressible as individual edge scores can be maximized (or minimized) locally with respect to edge contractions. Our implementation supports four different criteria, and here we report on two modularity-maximizing criteria. Validation experiments show that our implementation yields solutions comparable in quality to the state-of-the-art sequential agglomerative algorithm.

Capability and performance. On a 128 processor Cray XMT with 1 TiB of memory, our algorithm extracts communities from a graph of 122 million vertices and 1.99 billion edges into communities by maximizing modularity in under 7300 seconds. Currently, our method for *generating* artificial test data rather than our community detection algorithm is limiting our largest input sizes. Our edge-list implementation scales in execution time up to 128 processors on sufficiently large graphs.

2 Agglomerative Community Detection

Agglomerative clustering algorithms start with every vertex in a singleton community. Edges are scored through some metric, and a local optimization heuristic chooses the next edge(s) to contract. To increase available parallelism, we choose multiple contraction edges simultaneously as opposed to Clauset, Newman, and Moore [7]'s sequential algorithm. Chosen edges form a maximal cardinality matching that approximates the maximum weight, maximal cardinality matching. We consider maximizing metrics (without loss of generality) and also target a local maximum rather than a global, possibly non-approximable, maximum. There are a wide variety of metrics in use for optimizing and evaluating communities [10]. We focus on the established measure modularity defined in Section 2.2.

2.1 Defining the algorithm

We take an input graph $G_0 = (V_0, E_0)$ and re-interpret G_0 as a *community graph* $G = (V, E)$. Each vertex in a community graph is a disjoint subset of the input graph's vertices, and we begin with $V = \{\{i\} | i \in V_0\}$. Each edge i, j in E corresponds to an edge between communities, $E = \{\{i, j\} | \exists i_0 \in i \, \exists j_0 \in j$ such that $\{i_0, j_0\} \in E_0\}$. Assign edge weights $w(\{i, j\})$ to count the number of edges between communities i and j. To make matrix expressions simpler, let self-edge weights $w(\{i, i\})$ count the volume or sum of degrees where both end vertices lie in community i. Our algorithm will contract edges $\{i, j\}$ in G to form a new community graph G' with vertices representing the union of disjoint communities i and j.

We define our agglomerative algorithm with matrix operations. Consider the typical mapping of an undirected graph $G = (V, E)$ with edge weights $w(\{i, j\})$ to

a sparse, symmetric adjacency matrix A. The matrix A has dimension $|V| \times |V|$, where $|V|$ is the number of vertices in G. The matrix has two non-zero entries for each edge $\{i, j\} \in E$ with $i \neq j$, $A(i, j) = A(j, i) = w(\{i, j\})$. Self edges appear along the diagonal, $A(i, i) = w(\{i, i\})$.

All of Section 2.2's metrics use additive edge weights. Represent a set of edge contractions with a matching matrix M of dimension $|V'| \times |V|$, where $|V'|$ is the number of vertices of the *contracted* graph, and entries $M(i, j) = 1$ when vertex $j \in V$ contracts into vertex $i \in V'$. A maximal matching should produce a matrix M with as many degree-two columns as possible, and ideally $|V'| \approx |V|/2$. With M and additive weights, we represent the graph contraction from A to A' with $A' = M \cdot A \cdot M^T$.

Our agglomerative algorithm is agnostic of local maximization criteria. Given a metric of interest, our algorithm works abstractly as follows:

1. Construct the symmetric sparse matrix A representing undirected multi-graph G and its edge multiplicities.
2. Set the initial trivial community mapping $C := I$, the $|V|^2$ identity matrix.
3. While the number of communities and the largest community size have not reached a pre-set limit, repeat:
   (a) Compute a sparse matrix of edge scores according to the optimization metric given A and v.
   (b) Compute a matching matrix M on the score matrix to maximize the metric of interest.
   (c) If the matching does not change the number of vertices, quit.
   (d) Contract $A := M \cdot A \cdot M^T$.
   (e) Update the community mapping $C := M \cdot C$.

When a matching does not change the number of vertices, no edges increase the metric of interest. Section 3's parallel implementation works with an edge list data structure rather than typical compressed formats and implements the sparse operations directly.

Typical sequential agglomerative algorithms like Clauset, *et al.* (CNM) [7]'s method contract a single edge in each iteration. Our algorithm generalizes this sequential approach by identifying many contraction edges simultaneously. If the matching M contracts most edges of the graph at once, most edge scores will need recomputation. This differs from CNM [7]'s incremental edge scoring but does not affect the algorithm's asymptotic complexity.

Assuming all edges are scored in a total of $O(|E|)$ operations and some heavy weight maximal matching is computed in $O(|E|)$ [20] where E is the edge set of the current community graph, each iteration of our algorithm's inner loop requires $O(|E|)$ operations. As with other algorithms, the total operation count depends on the community growth rates. If our algorithm halts after K contraction phases, our algorithm runs in $O(|E| \cdot K)$ operations. If the community graph is halved with each iteration, our algorithm requires $O(|E| \cdot \log |V|)$ operations. If the graph is a star, only two vertices are contracted per step and our algorithm requires $O(|E| \cdot |V|)$ operations. This matches experience with the CNM algorithm [24].

2.2 Local optimization metrics

Here we score edges for contraction by modularity, an estimate of a community's deviation from random chance [16,3]. We maximize modularity by choosing the

largest independent changes from the current graph to the new graph by one of two heuristics. Minimization measures like conductance involve maximizing changes' negations.

Modularity. Newman [15]'s modularity metric compares the connectivity within a collection of vertices to the expected connectivity of a random graph with the same degree distribution. Let m be the number of edges in an undirected graph $G = G(V, E)$ with vertex set V and edge set E. Let $S \subset V$ induce a graph $G_S = G(S, E_S)$ with $E_S \subset E$ containing only edges where both endpoints are in S. Let m_S be the number of edges $|E_S|$, and let $\overline{m_S}$ be an expected number of edges in S given some statistical background model. Define the modularity of the community induced by S as $Q_S = \frac{1}{m}(m_S - \overline{m_S})$. Modularity represents the deviation of connectivity in the community induced by S from an expected background model. Given a partition $V = S_1 \cup S_2 \cup \cdots \cup S_k$, the modularity of that partitioning is $Q = \sum_{i=1}^{k} Q_{S_i}$.

Newman [15] considers the specific background model of a random graph with the same degree distribution as G where edges are independently and identically distributed. If x_S is the total number of edges in G where either endpoint is in S, then we have $Q_S = (m_S - x_S^2/4m)/m$ as in [3]. A subset S is considered a module when there are more internal edges than expected, $Q_S > 0$. The m_S term encourages forming large modules, while the x_S term penalizes modules with excess external edges. Maximizing Q_S finds communities with more internal connections than external ones. Expressed in matrix terms, optimizing modularity is a quadratic integer program and is an NP-complete optimization problem [5]. We compute a local maximum and not a global maximum. Different operation orders produce different locally optimal points.

Section 3's implementation scores edges by the change in modularity contracting that one edge would produce, analogous to the sequential CNM algorithm. Merging the vertex U into a disjoint set of vertices $W \in C$, requires that the change $\Delta Q(W, U) = Q_{W \cup U} - (Q_W + Q_U) > 0$. Expanding the expression for modularity,

$$\begin{aligned} m \cdot \Delta Q(W, U) &= m\,(Q_{W \cup U} - (Q_W + Q_U)) \\ &= (m_{W \cup U} - (m_W + m_U) - \\ &\qquad (\overline{m_{W \cup U}} - (\overline{m_W} + \overline{m_U})) \\ &= m_{W \leftrightarrow U} - \overline{(m_{W \cup U} - (m_W + m_U))}, \end{aligned}$$

where $m_{W \leftrightarrow U}$ is the number of edges between vertices in sets W and U. Assuming the edges are independent and identically distributed across vertices respecting their degrees [7],

$$\begin{aligned} \overline{(m_{W \cup U} - (m_W + m_U))} &= m \cdot \frac{x_W}{2m} \cdot \frac{x_U}{2m}, \text{ and} \\ \Delta Q(W, U) &= \frac{m_{W \leftrightarrow U}}{m} - \frac{x_W}{2m} \cdot \frac{x_U}{2m}. \end{aligned} \tag{1}$$

We track $m_{W \leftrightarrow U}$ and x_W in the contracted graph's edge and vertex weights, respectively. The quantity x_W equals the sum of W's degrees or the volume of W. In Section 2's matrix notation, ΔQ is the rank-one update $A/m - (v/2m) \cdot (v/2m)^T$

restricted to non-zero, off-diagonal entries of A. The data necessary for computing the score of edge $\{i, j\}$ are $A(i, j)$, $v(i)$, and $v(j)$, similar in spirit to a rank-one sparse matrix-vector update.

Modularity has known limitations. Fortunato and Barthélemy [9] demonstrate that global modularity optimization cannot distinguish between a single community and a group of smaller communities. Berry *et al.* [4] provide a weighting mechanism that overcomes this resolution limit. Instead of this weighting, we compare CNM with the modularity-normalizing method of McCloskey and Bader [3].

McCloskey and Bader's algorithm (MB) only merges vertices into the community when the change is deemed statistically significant against a simple statistical model assuming independence between edges. The sequential MB algorithm computes the mean $\overline{\Delta Q(W,:)}$ and standard deviation $\sigma(\Delta Q(W,:))$ of all changes adjacent to community W. Rather than requiring only $\Delta Q(W, U) > 0$, MB requires a tunable level of statistical significance with $\Delta Q(W, U) > \overline{\Delta Q(W,:)} + k \cdot \sigma(\Delta Q(W,:))$. Section 4 sets $k = -1.5$. Sequentially, MB considers only edges adjacent to the vertex under consideration and tracks a history for wider perspective. Because we evaluate merges adjacent to all communities at once by matching, we instead filter against the threshold computed across all current potential merges.

3 Mapping Our Algorithm to the Cray XMT

Our algorithm matches the sequential CNM algorithm's operation complexity while avoiding potential bottlenecks in priority queues. Here we outline the mapping from our algorithm to a massively multithreaded shared-memory platform, the Cray XMT. The Cray XMT provides a flat, shared-memory execution environment; Section 5 discusses other possibilities. The parallel mapping is straight-forward for this environment and still scales to massive graphs.

The Cray XMT is a supercomputing platform designed to accelerate massive graph analysis codes. The architecture tolerates high memory latencies using massive multithreading. There is no cache in the processors; all latency is handled by threading. Each Threadstorm processor within a Cray XMT contains user-available 100 hardware streams each maintaining a thread context. Context switches between threads occur every cycle, selecting a new thread from the pool of streams ready to execute.

A large, globally shared memory enables the analysis of graphs using a simple shared-memory programming model. Physical address hashing breaks up locality and ensures every node contributes to the aggregate memory bandwidth. Synchronization occurs at the level of 64-bit words through full/empty bits and primitives like an atomic fetch-and-add. The cost of synchronization is amortized over the cost of memory access. Combined, these features assist developing scalable parallel implementations for massive graph analysis.

Within our implementation, the edge scoring heuristics (CNM and MB) parallelize evenly across the edges. Evaluating the scores for all $|E|$ edges requires access to $O(|E|)$ scattered memory locations. Our implementation stores the graph as a vector of self-edge weights and an array of edges $\{i, j\}$ with $i > j$, equivalent to an unpacked lower-triangular sparse matrix representation. Given a matching M, we implement the sparse projection $M \cdot A \cdot M^T$ in-place. Vertices

are relabeled and duplicate edges eliminated using $|V| + |E|$ workspace. The implementation forms linked lists of potential duplicates and walks that list; we will see that this list-walking limits our concurrency and ultimate scalability.

To compute the matching M, we begin with a greedy, non-maximal algorithm. We iterate in parallel across the edge array. Each edge $\{i, j\}$ checks its end points i and j. If the current edge is the best possible match seen so far for *both* i and j, the edge registers itself with both endpoints. The Cray XMT's full/empty word synchronization ensures that edge registration occurs without race conditions. Because there is no ordering enforced between edges, this provides neither a maximal nor an approximately maximum weight matching but uses only $2|V|$ working space and $O|E|$ operations.

To compute a maximal matching, we run the non-maximal passes until they converge. On convergence, we have a maximal matching where every edge dominates its neighbors, ensuring a $1/2$ approximation to the maximum weight [12,14]. We have not analyzed the convergence rate, but our test cases converge in fewer than ten iterations.

Our implementation currently does not track the dendogram, or history of vertex contractions. The dendogram is a tree and can be represented by a $|V|$-long vector of parent pointers updated in $O(|V|)$ time per contraction step with no additional memory use beyond the tree itself.

4 Evaluating Parallel Community Detection

4.1 Parallel performance

We evaluate parallel scalability using artificial R-MAT [6,1] input graphs derived by sampling from a perturbed Kronecker product. R-MAT graphs are scale-free and reflect many properties of real social networks. We generate an each R-MAT graph with parameters $a = 0.55$, $b = c = 0.1$, and $d = 0.25$ and extract the largest component. An R-MAT generator takes a scale s and edge factor f as input and generates a sequence of $2^s \cdot f$ edges over 2^s vertices, including self-loops and repeated edges. We accumulate multiple edges within edge weights.

Our implementation scales to massive graphs, but evaluating strong scalability against a single Cray XMT processor requires using a smaller data set. We generate R-MAT graphs of scale 18 and 19 and with edge factors 8, 16, and 32. Table 1 shows the size of the largest component in each case. We use the largest component to investigate performance of the core algorithm and not heuristics for filtering the many singleton vertices and tiny components not connected to the largest component. The Cray XMT used for these experiments is located at Pacific Northwest National Lab and contains 128 Threadstorm processors running at 500 MHz. These 128 processors support over 12 000 hardware thread contexts. The globally addressable shared memory totals 1 TiB.

Figure 1 shows the speed-up against a single Cray XMT processor from three runs on each of Table 1's graphs. The speed-up plot shows some performance variation from the parallel, non-deterministic matching procedure. Unlike sequential experience, performance for the CNM and MB scoring methods is roughly similar. Figure 2 shows that performance plateaus when the matching phase takes as long as contraction. We are investigating why the phases' fractions of

Scale	Fact.	$\|V\|$	$\|E\|$	Avg. degree	Edge group
18	8	236 605	2 009 752	8.5	2M
	16	252 427	3 936 239	15.6	4M
	32	259 372	7 605 572	29.3	8M
19	8	467 993	3 480 977	7.4	4M
	16	502 152	7 369 885	14.7	8M
	32	517 452	14 853 837	28.7	16M

Table 1. We evaluate performance against multiple graphs generated by R-MAT with the given scale and edge factor. The graphs are lumped into rough categories by the number of R-MAT generated edges.

time change with more processors. To test scalability to large data sets, applying our algorithm to a 122 million vertex and 1.99 billion edge graph generated using scale 27 and edge factor 16 requires 7258 seconds using CNM and 7286 seconds using MB.

4.2 Community quality

Computing communities quickly is only good if the communities themselves are useful. We compare the modularity results from Table 1's scale 18 graphs between our parallel implementation and the state-of-the-art implementation in SNAP[2]. Because our parallel matching algorithm is non-deterministic, we use three runs for each P value. All evaluations are run sequentially through SNAP using the output community maps.

Figure 3 shows the modularity values from our parallel community detection implementation against those returned by SNAP. Forcing a more balanced merge though a maximal matching produces communities not as modular as sequential CNM optimization, but more modular than sequential MB. The number of communities also finds a compromise between the different sequential methods.

5 Related Work

Graph partitioning, graph clustering, and community detection are tightly related topics. A recent survey by Fortunato [10] covers many aspects of community detection with an emphasis on modularity maximization. Nearly all existing work of which we know is sequential and targets specific contraction edge scoring mechanisms.

Zhang *et al.* [26] recently proposed a parallel algorithm that identifies communities based on a custom metric rather than modularity. Gehweiler and Meyerhenke [11] proposed a distributed diffusive heuristic for implicit modularity-based graph clustering. Classic work on parallel modular decompositions [18] finds a different kind of module, one where any two vertices in a module have identical neighbors and somewhat are indistinguishable. This could provide a scalable pre-processing step that collapses vertices that will end up in the same community, although removing the degree-1 fringe may have the same effect.

Work on sequential multilevel agglomerative algorithms like [17] focuses on edge scoring and local refinement. Our algorithm is agnostic towards edge scoring

methods and can benefit from any problem-specific methods. The Cray XMT's word-level synchronization may help parallelize refinement methods, but we leave that to future work. Outside of the edge scoring, our algorithm relies on well-known primitives that exist for many execution models. The matching matrix M is equivalent to an algebraic multigrid restriction operator; implementations for applying restriction operators are widely available.

6 Observations

Our algorithm and implementation, the first parallel algorithm for agglomerative community detection, scales to 128 processors on a Cray XMT and can process massive graphs in a reasonable length of time. Finding communities in graph with 122 million vertices and nearly two billion edges requires slightly more than two hours. Our implementation can optimize with respect to different local optimization criteria, and its modularity results are comparable to a state-of-the-art sequential implementation. As a twist to established sequential algorithms for agglomerative community detection, our parallel algorithm takes a novel and naturally parallel approach to agglomeration with maximum weighted matchings. That difference appears to reduce differences between the CNM and MB edge scoring methods. The algorithm is simpler than existing sequential algorithms and opens new directions for improvement. Separating scoring, choosing, and merging edges may lead to improved metrics and solutions.

7 Acknowledgments

This work was supported in part by the Pacific Northwest National Laboratory (PNNL) CASS-MT Center and NSF Grants CNS-0708307 and IIP-0934114. We also thank PNNL and Cray, Inc. for providing access to Cray XMT hardware.

References

1. Bader, D., Gilbert, J., Kepner, J., Koester, D., Loh, E., Madduri, K., Mann, W., Meuse, T.: HPCS SSCA#2 Graph Analysis Benchmark Specifications v1.1 (Jul 2005)
2. Bader, D., Madduri, K.: SNAP, Small-world Network Analysis and Partitioning: an open-source parallel graph framework for the exploration of large-scale networks. In: Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2008). Miami, FL (Apr 2008)
3. Bader, D., McCloskey, J.: Modularity and graph algorithms (Sep 2009), presented at UMBC
4. Berry., J., Hendrickson, B., LaViolette, R., Phillips, C.: Tolerating the community detection resolution limit with edge weighting. CoRR abs/0903.1072 (2009)
5. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Trans. Knowledge and Data Engineering 20(2), 172–188 (2008)
6. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: Proc. 4th SIAM Intl. Conf. on Data Mining (SDM). SIAM, Orlando, FL (Apr 2004)

7. Clauset, A., Newman, M., Moore, C.: Finding community structure in very large networks. Physical Review E 70(6), 66111 (2004)
8. Facebook, Inc.: User statistics (March 2011), http://www.facebook.com/press/info.php?statistics
9. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. Proc. of the National Academy of Sciences 104(1), 36–41 (2007)
10. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75 – 174 (2010)
11. Gehweiler, J., Meyerhenke, H.: A distributed diffusive heuristic for clustering a virtual P2P supercomputer. In: Proc. 7th High-Performance Grid Computing Workshop (HGCW'10) in conjunction with 24th Intl. Parallel and Distributed Processing Symposium (IPDPS'10). IEEE Computer Society (2010)
12. Hoepman, J.H.: Simple distributed weighted matchings. CoRR cs.DC/0410047 (2004)
13. Lozano, S., Duch, J., Arenas, A.: Analysis of large social datasets by community detection. The European Physical Journal - Special Topics 143, 257–259 (2007)
14. Manne, F., Bisseling, R.: A parallel approximation algorithm for the weighted maximum matching problem. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 4967, pp. 708–717. Springer Berlin / Heidelberg (2008)
15. Newman, M.: Modularity and community structure in networks. Proc. of the National Academy of Sciences 103(23), 8577–8582 (2006)
16. Newman, M., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 026113 (Feb 2004)
17. Noack, A., Rotta, R.: Multi-level algorithms for modularity clustering. In: Vahrenhold, J. (ed.) Experimental Algorithms, Lecture Notes in Computer Science, vol. 5526, pp. 257–268. Springer Berlin / Heidelberg (2009)
18. Novick, M.B.: Fast parallel algorithms for the modular decomposition. Tech. rep., Cornell University, Ithaca, NY, USA (1989)
19. NYSE Euronext: Consolidated volume in NYSE listed issues, 2010 - current (March 2011), http://www.nyxdata.com/nysedata/asp/factbook/viewer_edition.asp?mode=table&key=3139&category=3
20. Preis, R.: Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) STACS 99, Lecture Notes in Computer Science, vol. 1563, pp. 259–269. Springer Berlin / Heidelberg (1999)
21. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proc. of the National Academy of Sciences 101(9), 2658 (2004)
22. Ravasz, E., Somera, A.L., Mongru, D.A., Oltvai, Z.N., Barabási, A.L.: Hierarchical organization of modularity in metabolic networks. Science 297(5586), 1551–1555 (2002)
23. Twitter, Inc.: Happy birthday Twitter! (March 2011), http://blog.twitter.com/2011/03/happy-birthday-twitter.html
24. Wakita, K., Tsurumi, T.: Finding community structure in mega-scale social networks. CoRR abs/cs/0702048 (2007)
25. Wilkinson, D.M., Huberman, B.A.: A method for finding communities of related genes. Proceedings of the National Academy of Sciences of the United States of America 101(Suppl 1), 5241–5248 (2004)
26. Zhang, Y., Wang, J., Wang, Y., Zhou, L.: Parallel community detection on large networks with propinquity dynamics. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 997–1006. KDD '09, ACM, New York, NY, USA (2009)

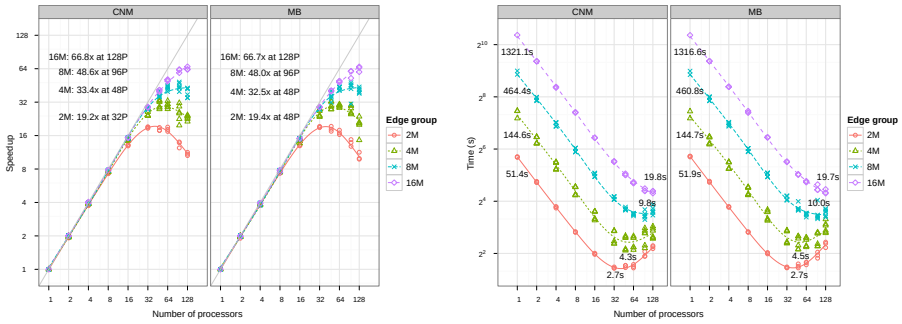

Fig. 1. Execution time on the largest of Table 1's graphs scales up to 128 processors. The left plot shows the best speed-up achieved for each edge group. The right plot shows both the best overall execution time and the best single-processor execution time.

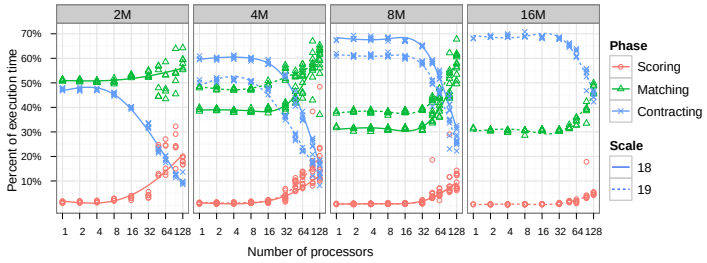

Fig. 2. Execution time fractions show that performance flattens where the matching phase takes as much execution time as the graph contraction.

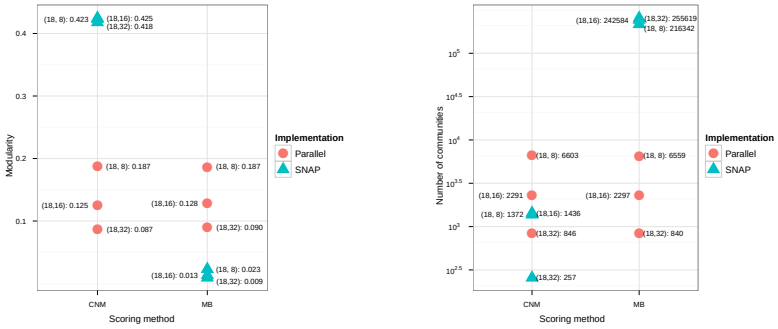

Fig. 3. Comparing modularity values and the number of communities between our parallel agglomerative community detection and a separate, sequential implementation in SNAP shows that ours finds an interesting trade-off between community sizes and modularity. Graph points are labeled by (scale, edge factor) and show either the modularity (left) or number of communities (right).