Parallel Algorithms through Approximation: b-Edge Cover

Arif Khan
Data Analytics
Pacific Northwest National Laboratory
Richland, Washington, USA
ariful.khan@pnnl.gov

Alex Pothen
Computer Science
Purdue University
West Lafayette, Indiana, USA
apothen@purdue.edu

S M Ferdous
Computer Science
Purdue University
West Lafayette, Indiana, USA
sferdou@purdue.edu

***Abstract*—We describe a paradigm for designing parallel algorithms via approximation, and illustrate it on the b-EDGE COVER problem. A b-EDGE COVER of minimum weight in a graph is a subset C of its edges such that at least a specified number $b(v)$ of edges in C is incident on each vertex v, and the sum of the edge weights in C is minimum. The GREEDY algorithm and a variant, the LSE algorithm, provide $3/2$-approximation guarantees in the worst-case for this problem, but these algorithms have limited parallelism. Hence we design two new 2-approximation algorithms with greater concurrency. The MCE algorithm reduces the computation of a b-EDGE COVER to that of finding a b'-MATCHING, by exploiting the relationship between these subgraphs in an approximation context. The S-LSE is derived from the LSE algorithm using static edge weights rather than dynamically computing effective edge weights. This relaxation gives S-LSE a worse approximation guarantee but makes it more amenable to parallelization. We prove that both the MCE and S-LSE algorithms compute the same b-EDGE COVER with at most twice the weight of the minimum weight edge cover. In practice, the 2-approximation and $3/2$-approximation algorithms compute edge covers of weight within 10% the optimal. We implement three of the approximation algorithms, MCE, LSE, and S-LSE, on shared memory multi-core machines, including an Intel Xeon and an IBM Power8 machine with 8 TB memory. The MCE algorithm is the fastest of these by an order of magnitude or more. It computes an edge cover in a graph with billions of edges in 20 seconds using two hundred threads on the IBM Power8. We also show that the parallel depth and work can be bounded for the SUITOR and b-SUITOR algorithms when edge weights are random.**

***Keywords*-b-EDGE COVER; b-MATCHING; Approximation Algorithms; Parallel Algorithms.**

I. INTRODUCTION

We consider a paradigm for designing practical parallel algorithms for certain graph problems through approximation algorithms. Algorithms for solving these problems exactly are impractical for massive graphs, and possess little concurrency. Often approximation algorithms can solve such problems faster in serial, but to take advantage of parallelism, new algorithms that possess high degrees of concurrency need to be designed.

We illustrate this paradigm by considering the minimum weight b-EDGE COVER problem, where the objective is to choose a subset of edges C in the graph such that *at least* a specified number $b(v)$ of edges in C is incident on each vertex v. Subject to this restriction on the subset of edges, we minimize the sum of the weights of the edges in C. The closely related maximum weight b-MATCHING problem chooses a subset of *at most* $b(v)$ edges incident on each vertex v to include in the matching, and then we maximize the sum of the weights of the matched edges. We will describe the complementary relationship between these two problems for approximation algorithms.

The paradigm of designing approximation algorithms for parallelism has been considered in the theoretical computer science community for vertex and set cover problems by Khuller, Vishkin and Young [14], and for facility location, max cut, set cover, and low stretch spanning trees, by Blelloch, Tangwongsan and coauthors, e.g., [3], [22]. The idea underlying many of these parallel algorithms is that a greedy algorithm chooses a most cost-effective element in each iteration, and by allowing a small slack, a factor of $(1+\epsilon)$, more elements can be selected at the cost of a slightly worse approximation ratio. These are algorithms with poly-logarithmic depth, and although some of them have linear work requirements, there are no parallel implementations that we know of.

The approximation paradigm for parallelism has been previously employed for MATCHING and b-MATCHING problems. The GREEDY algorithm for MATCHING does not have much concurrency, and Preis [19] developed the Locally Dominant edge algorithm, which was implemented for shared-memory parallel machines by Manne and Bisseling [16]. Manne and Halappanavar [17] developed the SUITOR algorithm, which has even more concurrency at the expense of annulled proposals, and this algorithm was extended to the b-SUITOR algorithm for b-MATCHINGS on both shared-memory and distributed-memory computers by our group [12], [13]. Azad et al. [1] have applied a $2/3-\epsilon$-approximation algorithm for weighted perfect matchings in bipartite graphs to compute good orderings for sparse Gaussian elimination.

The minimum weight b-EDGE COVER problem is rich in the space of approximation algorithms, and we consider four such algorithms here. A GREEDY algorithm and a variant, the LSE (locally subdominant edge) algorithm that we have designed earlier, have $3/2$-approximation ratios. Since these algorithms do not have much parallelism, we describe new 2-approximation algorithms that are more concurrent. We implement the new approximation algorithms on a multicore shared-memory multiprocessor, and compare their performance with the earlier $3/2$-approximation algorithms for this problem. Thus we trade off increased parallel performance for a slightly higher worst-case approximation ratio. We show that in practice nearly minimum weight edge covers are obtained. In the next few paragraphs, we add more detail to these statements.

The GREEDY algorithm for the b-EDGE COVER problem requires the *effective weight* of each edge, which is the weight of the edge divided the number of its endpoints that do not yet have their $b(.)$ values satisfied by edges included in the cover. Thus initially this is half the weight of an edge (u, v); it could then equal the weight of the edge, or become infinite when the $b(v)$ values of one or both of its endpoints are satisfied. At each iteration, an edge with the minimum value of the effective weight is added to the cover, and the weights of neighboring edges are updated. The order in which the edges are added to the cover and the dynamic updates of the edge weights cause the algorithm to be not amenable to parallelization.

Earlier, we have proposed a $3/2$-approximation algorithm called the LSE algorithm [11], which relaxes the order in which edges are added to the cover, making it more concurrent. An edge (u, v) is *locally sub-dominant* if it has the minimum weight among all edges incident on its endpoints u and v. The LSE algorithm adds a locally sub-dominant edge to the cover, deletes this edge from the graph, updates the effective weights of neighboring edges, and updates the $b(.)$ values of its endpoints. The algorithm iterates until all $b(.)$ values are satisfied. Unfortunately, the dynamic weight update in the LSE algorithm makes the parallel implementation inefficient. If we work with the static edge weights instead of the dynamic effective weights, we obtain a 2-approximation guarantee, while significantly improving the run time performance and scalability. We call this algorithm S-LSE, i.e., LSE with static edge weights and no effective weight update, and this is a new contribution in this paper. The S-LSE algorithm iteratively adds a set of locally sub-dominant edges to the current edge cover.

Our earlier paper [11] discusses the GREEDY and LSE algorithms in detail. Both algorithms have the effective weight update step in common, and this weight update step takes $85\% - 90\%$ of total time for the LSE algorithm. The reason is that in any given iteration, the edges whose weight need to be updated reside in different parts of the graph, making the memory accesses for weight updates irregular, causing loss of performance.

A major contribution of this paper is to describe a new 2-approximation algorithm, the MCE algorithm for b-EDGE COVER that first computes a b'-MATCHING, and then takes the complement of the matched edges. (The value of $b'(v) = \deg(v) - b(v)$, where $\deg(v)$ is the degree of a vertex v.) There is a complementary relationship between these problems in the context of optimal matchings and edge covers, and we extend it to approximate solutions, discuss the condition under which this relationship holds, and use it to design the MCE algorithm.

We design parallel versions of the LSE, S-LSE and MCE algorithms, and compare the run time performances of these algorithms on Intel Xeon and IBM Power8 multiprocessors. We show that the MCE algorithm is the fastest among these algorithms both on serial and shared memory multi-threaded processors, outperforming others by at least an order of magnitude. The MCE algorithm employs the b-SUITOR algorithm for computing a b'-MATCHING; the latter algorithm scales to 16K cores of a distributed memory machine [13]. We show here that b-EDGE COVERS in a graph with billions of edges can be computed in seconds with a Terabyte-scale shared memory machine using hundreds of threads.

The rest of this paper is organized as follows. We provide background on the b-EDGE COVER problem and its relation to matchings in Section II. In Section III, we discuss our proposed 2-approximation algorithms S-LSE and MCE. Parallel implementations of these algorithms are described in Section IV. The worst-case approximation guarantee of 2 for the MCE algorithm, and that the MCE and the LSE algorithms compute the same edge cover, are proved in Section V. The parallel depth and work of the SUITOR and b-SUITOR algorithms on which the MCE algorithm depends, are included in Section VI. Our experiments and results are described in Section VII, and we conclude in Section VIII.

II. BACKGROUND

The well-known k-nearest neighbor graph construction to represent noisy and dense data is related to the b-EDGE COVER problem. The formulation as a b-EDGE COVER problem is more general, since instead of using a uniform value of b, we can choose $b(v)$ to depend on each vertex v. Furthermore, as the work in this paper shows, this construction creates redundant edges which may be removed to obtain a sparser graph while satisfying the $b(v)$ constraints. Finally, our work shows that this construction creates a subgraph whose weight can be proved to be at most twice the minimum weight obtainable. We explore this relationship in more detail in [20], and focus here on the MCE and S-LSE algorithms.

The b-EDGE COVER problem arises in communication or distribution problems where reliability is important, i.e., each communication node has to be *"covered"* several times to increase reliability in the event of a communication link failing [15]. We have also used a b-EDGE COVER to solve the adaptive anonymity problem [6], where we wish to publish a database, with individuals corresponding to rows, features corresponding to columns, and we mask a few elements before publication in order to satisfy the privacy requirements of individuals. (Although [6] uses b-MATCHINGS for k-anonymity, we have shown that it is the b-EDGE COVER problem that should be used for adaptive anonymity.)

An exact algorithm for the minimum weight EDGE COVER problem can be obtained by reducing it to the minimum weight perfect MATCHING problem, as described in Schrijver [21]. This reduction makes a second copy of the original graph G, and then connects corresponding vertices in the two copies by an edge with weight equal to twice the minimum weight of an edge incident on the vertex in the original graph. (We call these edges linking edges). A minimum weight perfect matching in the latter can be transformed to a minimum weight edge cover in the original graph by including the matched edges in the original graph, and replacing every matched linking edge by a lowest weight edge incident on that vertex.

Let $b(V) = \sum_{v \in V} b(v)$, $\beta = \max_{v \in V} b(v)$, n denote the number of vertices, and m denote the number of edges in a graph. An exact algorithm for b-MATCHING has $O(b(V)\, m \log n)$ time complexity, but this is impractically slow for large graphs, and it does not have much concurrency. There have been no practical parallel algorithms and implementations for b-EDGE COVER in earlier work.

A minimum weight b-EDGE COVER can be computed as the complement of a b'-MATCHING, as described in the Introduction. In earlier work, we have developed a $1/2$-approximation algorithm for b-MATCHING called b-SUITOR, which is related to proposal based algorithms for the Stable Fixtures problem, a variant of stable matchings. The serial b-SUITOR algorithm has time complexity $O(m \log \beta)$, and it is currently the fastest practical algorithm on serial, shared memory, and distributed memory machines, scaling to $16K$ cores or more [12], [13]. On serial machines, the b-SUITOR algorithm is several orders of magnitude faster than earlier exact algorithms for b-MATCHING; it is about 900 times faster than an integer linear programming algorithm and 300 times faster than a belief propagation algorithm. We employ the b-SUITOR algorithm to compute b-EDGE COVERS in this paper, and hence will discuss it in more detail later.

The b-EDGE COVER problem is a special case of the *Set Multicover* problem: Here we are given a collection of subsets of a set, each with a cost, and we are required to find a sub-collection of subsets of minimum total cost to cover each element e in the set a specified number $b(e)$ times. If each subset has exactly two elements, then we have the b-EDGE COVER problem. Chvatal [7] obtained an H_n approximation algorithm for the minimum cost *Set Cover* problem, where H_n is the n-th harmonic number. Dobson [9] proposed an H_a-approximation algorithm using integer programming for the minimum cost Set Multicover problem, where a is the maximum number of elements in any subset.

III. NEW 2-APPROXIMATION ALGORITHMS

In this section, we introduce two 2-approximation algorithms: i) S-LSE is the LSE algorithm without the effective weight update step, and ii) MCE, the matching complement edge cover algorithm, uses a b'-MATCHING to compute a b-EDGE COVER.

A. S-LSE: LSE *with no weight update*

The S-LSE algorithm iteratively computes a set of locally sub-dominant edges to add to the edge cover. Ties are broken by prioritizing an edge with lower numbered endpoints. In each iteration locally sub-dominant edges are uniquely defined, and are independent of each other, i.e., they do not share an endpoint. The algorithm iteratively finds a set of locally sub-dominant edges, adds them to the edge cover and updates $b(v)$ values. These edges are marked as deleted from the graph, and new locally dominant edges are identified. If both endpoints of an edge have their $b(v)$ values satisfied, then it is marked as deleted from the graph. The algorithm is described in Algorithm 1.

At each iteration, we calculate the set of locally sub-dominant edges S as follows. Each vertex u sets a pointer to the edge of least weight incident on it. If the endpoints of an edge point to each other, then the edge is locally sub-dominant. We pick each such edge, add it to the cover, remove it from further consideration, and decrement the $b(v)$ values of the end points. When the $b(.)$ values are satisfied for all vertices, we break the loop and then do a post-processing step called the *Redundant Edge Removal* step, which is described in the following subsection. After the post-processing, the algorithm terminates with a b-EDGE COVER, EC. The time complexity of the (serial) algorithm is $O(m \log \Delta)$, where Δ is the maximum degree of a vertex.

Algorithm 1 S-LSE($G(V, E, w), b$)

$EC = \emptyset$
while $b(.)$ constraints are not satisfied **do**
    Compute locally sub-dominant edges S of G
    for each $(u, v) \in S$ **do**
        $EC = EC \cup (u, v)$
        $E = E \setminus (u, v)$
        for $x \in \{u, v\}$ **do**
            if $b(x) > 0$ **then**
                $b(x) = b(x) - 1$
EC =Remove_Redundant_Edge(EC)
return b-EDGE COVER EC

1) Redundant Edges: We define a vertex u to be *saturated* if u is covered by exactly $b(u)$ edges, and *super-saturated* if u is covered by more than $b(u)$ edges in a b-EDGE COVER C. An edge $u, v \in C$ is *redundant* if both u and v are super-saturated. The GREEDY, LSE and S-LSE algorithms may have redundant edges. We can remove a redundant edge (u, v) without violating the constraints on $b(.)$ and reduce the weight of the edge cover.

We illustrate redundant edges by an example shown in Figure 1(a). We show a b-EDGE COVER computed using S-LSE algorithm before the post-processing step on a graph G, with $b(u) = b(v) = b(w) = b(x) = b(y) = 1$, and all other vertices have $b(.) = 2$. It shows that all the edges will be selected to be part of the edge cover. Figure 1(b) shows the subgraph induced by the super-saturated vertices with the redundant edges. If we remove (a, b) first, we can either remove (c, d) or (d, e) without violating the constraints and the resulting two possible solutions with their respective cover weights are shown in Figure 1(c). This illustrates that the order in which the algorithm removes redundant edges could determine the edge cover and its weight. This is not desirable in a parallel context because each vertex will be processed by different threads, and the scheduling of the threads depends on the underlying operating system. Therefore, the solution may be different from one run to another. We also want to remove heavier edges to obtain a solution with the lowest possible weight.

We achieve both of these goals by removing locally dominant edges in the subgraph induced by the redundant edges. An edge u, v is a *locally dominant* edge if its weight is maximum relative to the weights of all neighboring edges. Similar to locally sub-dominant edges, with a consistent tie-breaking scheme, the set of locally dominant edges is also uniquely defined, i.e., it does not depend on the order in which one processes a vertex. We consider the subgraph induced by the redundant edges [Figure 1(b)], and iteratively remove locally dominant edges. In the example described in Figure 1(b), (b, c) and (d, e) are locally dominant edges. The removal of these edges results in the b-EDGE COVER shown in Figure 1(d); it has lower weight than the other two edge covers shown in Figure 1(c), and is independent of the order in which vertices are processed.

B. Relationship between b'-MATCHING *and* b-EDGE COVER

We refer to b'-MATCHING instead of b-MATCHING to avoid ambiguity in this subsection. Given a graph $G = (V, E, b)$, a minimum weight b-EDGE COVER can be obtained from a maximum weight b'-MATCHING [21] as follows:

1) For each vertex v, compute $b'(v) = \deg(v) - b(v)$.
2) Compute M_{opt}, a maximum weight b'-MATCHING.
3) Compute a b-EDGE COVER as the complement of the matching: $C_{opt} = E \setminus M_{opt}$.

In this construction, steps 1 and 3 ensure that the computed b-EDGE COVER is a valid cover, and the optimality of the cover depends on step 2. If we compute an approximate b'-MATCHING, keeping steps 1 and 3 fixed, then the solution to the b-EDGE COVER *may not necessarily*

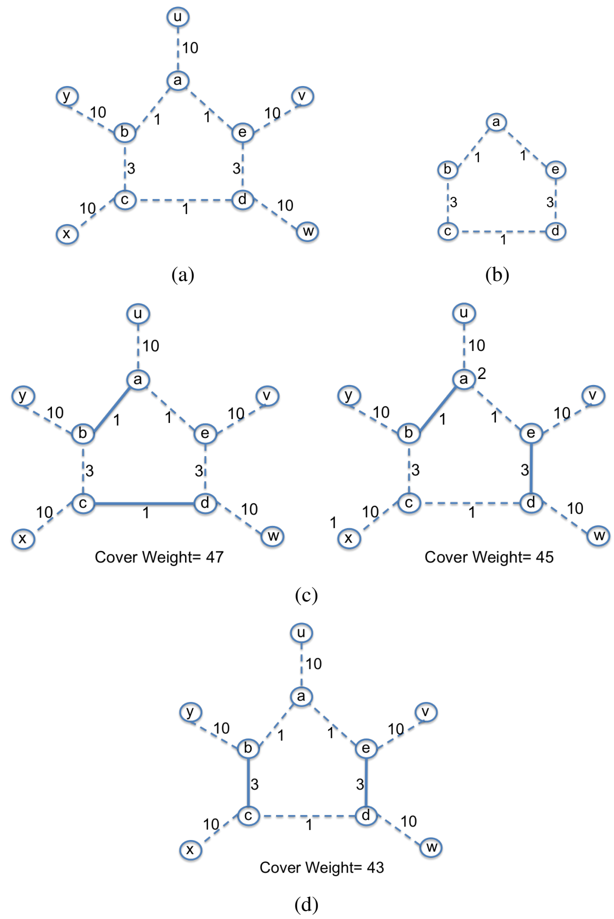

Figure 1. Removing redundant edges in a b-EDGE COVER. Subfigure (b) shows the subgraph induced by the potential redundant edges, and Subfigures (c) and (d) show different choices for edges to remove.

be an approximate solution for b-EDGE COVER. However, we show that if the b'-MATCHING is computed using the GREEDY algorithm (or an algorithm that matches locally dominant edges), then the corresponding b-EDGE COVER will satisfy 2-approximation bounds. We use b-SUITOR in step 2 and propose a new 2-approximation algorithm for b-EDGE COVER, and we call it the MCE algorithm.

Since b-SUITOR is an essential part of the MCE algorithm, we briefly describe a serial version of it in Algorithm 2. For more details, we refer the reader to our papers [12], [13]. This algorithm extends the SUITOR algorithm of Manne and Halappanavar [17] to b'-MATCHING. We describe a recursive version of the algorithm since it is easier to explain, although the versions we have implemented use iteration rather than recursion. Here $N(u)$ is the adjacency list of u, $S(u)$ is a priority queue of suitors of a vertex u, and $T(u)$ is an array of vertices that u has extended proposals to. The algorithm processes all of the vertices, and for each vertex u, it seeks to match up to $b'(u)$ neighbors. In each iteration a vertex u proposes to a heaviest neighbor v it has not proposed to yet, if the weight $W(u, v)$

is heavier than the weight offered by the last ($b'(v)$-th) suitor of v. If it fails to find a partner, then we break out of the loop. If it succeeds in finding a partner x, then the algorithm calls the function MakeSuitor to make u the Suitor of x. This function updates the priority queue $S(u)$ and the array $T(u)$. If when u proposes and becomes the Suitor of x, it annuls the proposal of the previous Suitor of x, the vertex y, then the algorithm looks for an eligible partner z for y, and calls MakeSuitor recursively to make y a Suitor of z. The vertex $S(v).last$ has the lowest weight of the $b'(v)$ suitors of v; it is zero is there are fewer than $b'(u)$ suitors. The time complexity of the algorithm is $O(m \log \beta')$.

Algorithm 2 b-SUITOR(G, b)

Create a min-priority heap $S(v)$ of size $b(v)$ for each v
for $u \in V$ **do**
  for $i = 1$ to $b(u)$ **do**
    $x = \underset{v \in N(u) \setminus T(u)}{\arg\max} \{W(u, v) : W(u, v) > W(v, S(v).last)\}$
    if $x = NULL$ **then**
      break
    else
      MakeSuitor(u, x)
return S

Algorithm 3 MakeSuitor(u, x)

$y = S(x).last$
$S(x).insert(u)$
$T(u).insert(x)$
if $y \neq NULL$ **then**
  $T(y).remove(x)$
  $z = \underset{v \in N(y) \setminus T(y)}{\arg\max} \{W(y, v) : W(y, v) > W(v, S(v).last)\}$
  if $z \neq NULL$ **then**
    MakeSuitor(y, z)

IV. PARALLEL b-EDGE COVER ALGORITHMS

In this section, we describe the parallel multi-threaded implementation of the MCE algorithm, using *OpenMP* for parallelization. Both the MCE and S-LSE algorithms compute *identical edge covers*, whether in serial or in parallel (irrespective of the number of threads). The LSE algorithm computes a different edge cover, but it also computes the same cover on both serial and parallel machines. This is a robust property of the parallel LSE, S-LSE, and MCE algorithms considered here that the edge covers computed are *the same on serial and parallel machines*. Tie-breaking in edge weights might change the edge cover computed, but it will not change the weight of the edge cover. Hence repeating the experiment does not change cover weights.

All the algorithms use locks for synchronizing multiple threads to ensure sequential consistency. We do not describe multi-threaded shared memory versions of the S-LSE and LSE algorithms here due to space limitations. The parallel MCE algorithm is described in Algorithm 4. First, we compute the b' values for each vertex in parallel; next we call the Parallel_b-SUITOR algorithm with input b'; and finally, we complement the matching by choosing the unmatched edges incident on each vertex.

Algorithm 4 MCE$(G(V, E, w), b)$

$EC = \emptyset$
for $v \in V$ **in parallel do**
  $b'(v) = max\{0, \delta(v) - b(v)\}$
M=Parallel_b-SUITOR(G, b')
for $v \in V$ **in parallel do**
  $EC = EC \cup \{N(v) \setminus M(v)\}$
return b-EDGE COVER EC

The parallel b-SUITOR algorithm is described in Algorithm 5. It is the "*Delayed Partial*" variant of the b-SUITOR algorithm described in [13]. The algorithm maintains a queue of unsaturated vertices Q for which it tries to find partners during the current iteration of the **while** loop, and also a queue of vertices Q' whose proposals are annulled in this iteration, and will be processed again in the next iteration. (This is what "delayed" means; annulled vertices are not processed in the same iteration. "Partial" means that the adjacency lists are partially sorted to find a subset of heaviest neighbors.) The algorithm then seeks a partner for each vertex u in Q in parallel. It tries to find $b(u)$ proposals for u to make while the adjacency list $N(u)$ has not been exhaustively searched thus far in the course of the algorithm.

Consider the situation when a vertex u has $i - 1 < b(u)$ outstanding proposals. The vertex u can propose to a vertex p in $N(u)$ if it is a heaviest neighbor in the set $N(u) \setminus T_{i-1}(u)$ (the array $T(u)$ from the previous step), and if the weight of the edge (u, p) is greater than the lowest offer that p has. In this case, p would accept the proposal of u rather than its current lowest offer.

If the algorithm finds a partner p for u, then the thread processing the vertex u attempts to acquire the lock for the priority queue $S(p)$ so that other vertices do not concurrently become Suitors of p. This attempt might take some time to succeed since another thread might have the lock for $S(p)$. Once the thread processing u succeeds in acquiring the lock, then it needs to check again if p continues to be an eligible partner, since by this time another thread might have found another Suitor for p, and its lowest offer might have changed. If p is still an eligible partner for u, then we increment the count of the number of proposals made by u, and make u a Suitor of p. If in this process, we dislodge the last Suitor x of p, then we add x to the queue of vertices Q' to be

Algorithm 5 Parallel_b-SUITOR(G, b)

$Q = V$; $Q' = \emptyset$;
$S(v) = \emptyset$, min-priority heap $\forall v$
while $Q \neq \emptyset$ **do**
  for vertices $u \in Q$ **in parallel do**
    $i = 1$;
    while $i <= b(u)$ and $N(u) \neq exhausted$ **do**
      Let $p \in N(u)$ be an eligible partner of u;
      if $p \neq NULL$ **then**
        Lock $S(p)$;
        if p is still eligible **then**
          $i = i + 1$;
          Add u to $S(p)$;
          if u annuls the proposal of v **then**
            Add v to Q'; Update $db(v)$;
            Remove v from $S(p)$;
        Unlock $S(p)$;
      else
        $N(u) = exhausted$;
  Update Q using Q'; Update b using db;
return S

processed in the next iteration. Finally the thread unlocks the queue $S(p)$.

We fail to find an eligible partner p for a vertex u when we have exhaustively searched all neighbors of u in $N(u)$, and none offers a weight greater than the lowest offer u has, $S(u).last$. In this case u has fewer than $b(u)$ matched neighbors. After we have considered every vertex $u \in Q$ to be processed, we can update data structures for the next iteration. We update Q to be the set of vertices in Q'; and the vector b to reflect the number of additional partners we need to find for each vertex u using $db(u)$, the number of times u's proposal was annulled by a neighbor.

V. APPROXIMATION BOUNDS

In this section, we show that MCE is a 2-approximation algorithm for b-EDGE COVER, and that both MCE and S-LSE algorithms compute the same b-EDGE COVER. We will need a Lemma from [13]. The GREEDY algorithm for b-MATCHING matches edges in increasing order of (static) edge weights.

Lemma 5.1: When the GREEDY algorithm for b-MATCHING matches an edge, it is a locally dominant edge in the residual graph (the graph induced by the currently unmatched edges).

Theorem 5.2: MCE is a 2-approximation algorithm for b-EDGE COVER.

Proof: Let the optimal minimum weight b-EDGE COVER be denoted by C_{opt}, the complement of an optimal maximum weight b'-MATCHING, M_{opt}. Also, let the b-EDGE COVER computed by MCE be denoted by C, which takes the complement of the $1/2$-approximate matching M, obtained by b-SUITOR.

Consider an edge $e(u, v) \in C_{opt} \setminus C$, which belongs to the optimal edge cover but not the approximate edge cover. This implies that $e(u, v) \in M \setminus M_{opt}$ since the covers are obtained by complementing the matched edges. The worst case scenario for b'-MATCHING is when b-SUITOR matches the edge $e(u, v)$, and thus cannot match two other edges that belong to M_{opt}, say $e(x, u) \in M_{opt}$ and $e(v, y) \in M_{opt}$. Hence $e(x, u) \notin M$ and $e(v, y) \notin M$. Since the b-SUITOR algorithm computes the same matching as the GREEDY algorithm, $e(u, v)$ must be a locally dominating edge when it is matched, by Lemma 5.1. Thus

$$\begin{aligned} &w(u,v) \geq w(x,u); \quad w(u,v) \geq w(v,y); \quad \text{hence} \\ &2w(u,v) \geq w(x,u) + w(v,y). \end{aligned} \tag{1}$$

Since $e(x, u) \notin M$ and $e(v, y) \notin M$, both of these edges belong to the approximate cover C. Therefore, the weight of C can be bounded as follows.

$$\begin{aligned} w(C) &= w(C_{opt}) - w(u,v) + w(x,u) + w(v,y) \\ &\leq w(C_{opt}) - w(u,v) + 2w(u,v) \quad \text{(from Eqn 1)} \\ &= w(C_{opt}) + w(u,v). \end{aligned} \tag{2}$$

By summing over all edges in the optimal cover that are not included in the approximate cover, $C_{opt} \setminus C$, we obtain

$$\begin{aligned} w(C) &\leq w(C_{opt}) + \sum_{(u,v) \in C_{opt}} w(u,v) \\ &= \ w(C_{opt}) + w(C_{opt}) \quad = 2\ w(C_{opt}). \end{aligned} \tag{3}$$

Thus MCE is a 2-approximation algorithm for b-EDGE COVER. ■

Lemma 5.3: A b-EDGE COVER computed by the MCE algorithm does not have redundant edges.

Proof: An approximate maximum weight b'-MATCHING M of a graph computed by the b-SUITOR algorithm cannot have two neighboring vertices u and v, with u having fewer than $b'(u)$ and v having fewer than $b'(v)$ incident edges belonging to M. For, then we can add the edge $e(u, v)$ to the b'-MATCHING without violating the matching constraints and increase the weight of the approximate matching. But this contradicts the fact that the b-SUITOR algorithm computes a maximal matching. By considering the complement, a b-EDGE COVER obtained by the MCE algorithm cannot have two super-saturated neighboring vertices in C. Hence a cover computed by the MCE algorithm does not have redundant edges. ■

Let us denote the edge cover obtained from the MCE algorithm by C_m, and the edge cover obtained from the S-LSE algorithm by C_l. We proceed to prove that these edge covers are identical. Consider the graph $G' = C_m \oplus C_l$, obtained by taking the symmetric difference of the two edge covers.

Lemma 5.4: If a vertex v in the symmetric difference graph G' has an equal number of edges from the covers C_m and C_l incident on it, then the vertex v is either super-saturated or saturated with respect to both edge covers C_m and C_l.

Proof: Suppose that v has $t \geq 1$ edges from C_m and $t \geq 1$ edges from C_l incident on it in the graph G'. Also suppose that a set of $k \geq 0$ edges incident on v in the original graph G are included in both edge covers C_m and C_l. These latter edges do not belong to the symmetric difference graph G'. Then the vertex v has $k + t$ edges incident on it in both C_m and C_l. If $b(v) = k + t$ then v is saturated in both C_m and C_l, and otherwise it is super-saturated in both edge covers. ■

Lemma 5.5: If a vertex $v \in G'$ has more edges from the edge cover C_m incident on it than from the edge cover C_l, then v is a super-saturated vertex in C_m. Similarly if the vertex v has more edges from the cover C_l incident on it than from the edge cover C_m, then v is a super-saturated vertex in C_l.

Proof: Consider the first of the two statements. Since C_l is a b-EDGE COVER, there are at least $b(v)$ edges in the graph G belonging to C_l incident on v. By the condition of the lemma, there are more than $b(v)$ edges in the graph G belonging to C_m incident on v, and hence it is super-saturated with respect to the edge cover C_m. The proof of the second statement is similar. ■

We proceed to show that the symmetric difference graph G' consists of isolated vertices, i.e., it does not have any edges, implying that the two edge covers C_m and C_l are identical.

Lemma 5.6: The symmetric difference graph G' does not have a vertex u with more C_m edges incident on it than C_l edges.

Proof: Let $C_m(u)$ denote the edges in C_m that are incident on u, and consider an edge $(u, v) \in C_m(u)$. If vertex u has more C_m edges incident on it than C_l, it must be super-saturated in C_m. Now v must be saturated in C_m, by Lemma 5.5. (The vertex v could be saturated or super-saturated in C_l.) The edge (u, v) incident on v belongs to C_m, and since v is at least saturated in C_l, there is an edge (v, x) that belongs to $C_l \setminus C_m$. Now since the S-LSE algorithm includes locally sub-dominant edges in the cover C_l, we have the inequality $w(v, x) < w(u, v)$. Now consider the approximate matching M from which the MCE algorithm computed the edge cover C_m. Since u is supersaturated in C_m, it has fewer than $b'(u)$ matched edges in M incident on it. Hence v could have proposed to its neighbor u, but did not, since $(u, v) \in C_m$, and not to its complement M. But the edge $(v, x) \in M$, since it does not belong to C_m. This implies that $w(v, x) > w(u, v)$. The two inequalities contradict each other, completing the proof. ■

Lemma 5.7: The symmetric difference graph G' does not have a vertex u that has an equal number of C_m and C_l edges incident on it.

Proof: Consider a vertex u in the graph G', and an edge (u, v) that belongs to $C_m \setminus C_l$. There are four subcases to consider with respect to the edge cover C_m.

The first case is when u and v are both super-saturated with respect to C_M, but this will make the edge (u, v) redundant, and such edges are deleted from C_m.

The second case is when u is super-saturated and v is saturated with respect to C_m. Since v is at least saturated in C_l, there is an edge $(v, x) \in C_l \setminus C_m$ in G'. This edge also belongs to the matching M from the MCE algorithm. Since the edge $(v, x) \in C_l$ and $(u, v) \notin C_l$, it must be a locally sub-dominant edge, and hence $w(v, x) < w(u, v)$. However, since u is super-saturated in C_m, it has fewer than $b'(u)$ matched edges from M incident on it. Thus v could have proposed to u, but instead it proposed to x, implying that $w(v, x) > w(u, v)$. Again, the two inequalities contradict each other.

The third case is when u is saturated in C_m and v is super-saturated in C_m. But this case reduces to the second case with u and v interchanged.

Finally, we have the case when u and v are both saturated in C_m. Choose an edge $(u, v) \in C_m \setminus C_l$ in G'. Since u and v are at least saturated in C_l, we have the edges $(t, u)\ inC_l \setminus C_m$, and $(v, x)\ inC_l \setminus C_m$. Now from the S-LSE algorithm, we have $w(t, u) < w(u, v)$ and $w(v, x) < w(u, v)$, which implies that the edge (u, v) is a locally dominant edge. Thus this edge should be chosen by the approximation algorithm for matching to include in M, which contradicts the assumption that it belongs to the edge cover C_m. This completes the proof. ■

Lemma 5.8: The symmetric difference graph G' does not have a vertex u with fewer C_m edges incident on it than C_l.

Proof: Let (u, v) be an edge that belongs to C_l, and let u have fewer C_m edges incident on it than C_l. Thus u is super-saturated with respect to C_l, and v must be saturated in C_l, by Lemma 5.5. We consider two cases.

The first case is when v is super-saturated in C_m. Now v is saturated in C_l implies that there are more C_m edges incident on v than C_l edges, and this reduces to Lemma 5.6.

The second case is when v is saturated in C_l. Since v is also saturated in C_m, an equal number of C_m and C_l edges are incident on v, and this reduces to Lemma 5.7.

This completes the proof of the Lemma. ■

Theorem 5.9: The S-LSE algorithm computes the same b-EDGE COVER as the MCE algorithm, and hence it is a 2-approximation algorithm for b-EDGE COVER.

Proof: From Lemmas 5.6, 5.7, and 5.8, the symmetric difference graph G' has only vertices of zero degree. Therefore, the two edge covers are the same, i.e., $C_m = C_l$. ■

VI. PARALLEL DEPTH AND WORK

In this section we show that the SUITOR [17] and the b-SUITOR algorithms have provably low parallel depth and work. The depth is the number of time steps needed by the parallel algorithm, and the work is the total number of operations performed by the algorithm. These are the first results on the depth of the SUITOR and b-SUITOR algorithms that we know of.

Theorem 6.1: The expected parallel depth of the SUITOR algorithm that computes a $1/2$-approximate 1-matching in a graph is $O(\log(\Delta) \log m)$, when the weights of the edges are chosen uniformly at random.

Proof: We begin by analyzing an algorithm related to the SUITOR algorithm, the LOCALLY DOMINANT EDGE algorithm. This algorithm adds an edge to the approximate matching when there are no neighboring edges of higher weight (it becomes locally dominant), and then deletes all of the neighboring edges. An algorithm of Blelloch, Fineman and Shun [2] for computing an unweighted maximal matching in parallel uses random priorities on the edges to compute the matching. Hence it is equivalent to the LOCALLY DOMINANT EDGE algorithm for weighted matching with random edge weights, and an analysis of the maximal matching algorithm shows that the LOCALLY DOMINANT EDGE algorithm has the stated parallel depth.

Now we turn to the SUITOR algorithm and consider its relationship to the LOCALLY DOMINANT EDGE algorithm. Specifically we consider the "delayed" version of the algorithm in which a vertex with a proposal annulled is queued for further processing in the next iteration. In the LOCALLY DOMINANT EDGE algorithm, an edge is matched when it becomes locally dominant, detected by its two endpoints pointing to each other. In the SUITOR algorithm, each vertex u keeps track of the highest weight of the proposal it has received so far. A neighbor of u could use this information, if it is already available, to propose to its next heaviest eligible neighbor without first proposing to u. Hence if we view the computations of these algorithms in rounds, in the SUITOR algorithm, a vertex gets matched in the same or an earlier round relative to the LOCALLY DOMINANT EDGE algorithm. Hence the SUITOR algorithm also has $O(\log(\Delta) \log m)$ depth. ■

Theorem 6.2: The expected work in the SUITOR algorithm is $O(m)$ when the edge weights are chosen uniformly at random.

Proof: The adjacency lists can be sorted in expected linear time using bucket sort when the weights are chosen randomly [8]. The SUITOR algorithm needs to go through the sorted adjacency list of each vertex at most once. ■

Obtaining linear work for the maximal matching algorithm of Blelloch et al. [2] is more complicated, and is accomplished by working on a prefix of the graph whose size is carefully chosen, which increases the depth to $O(\log^4 m/ \log\log m)$.

We now show that these results can be extended to the b-SUITOR algorithm by reducing the b-MATCHING problem to the MATCHING problem in a modified graph. We only sketch the reduction here due to space considerations. We replace each vertex u with $b(u)$ vertices in the modified graph; each edge (u, v) is replaced by a complete bipartite graph of $b(u)\, b(v)$ edges, with weights equal to the original weight of the edge (u, v). We restrict only one of the edges in the bipartite subgraph to be matched, but other vertices in this subgraph could be matched to edges in other subgraphs. We show an example of the reduction in Figure 2. The value of b is 2. We see each edge is replaced by a complete bipartite graph with the same weight. In the example graph, if we choose (A_1, B_1) as a matched edge then we can not match the edge (A_2, B_2). With this restriction, a $1/2$-approximate matching in the transformed graph would correspond to a $1/2$- approximate b-MATCHING in the original graph.

Thus the parallel depth of b-SUITOR algorithm when the edge weights are uniformly random becomes $O(\log(\Delta) \ \log b(V))$. Similarly the work becomes $O(\beta \ b(V))$. (Recall that $\beta = \max_v b(v)$, and $b(V) = \sum_v b(v)$.) For the MCE algorithm for the b-EDGE COVER, the depth is $O(log(\Delta) \ \log b'(V))$; and the work is $O(\beta' \ b'(V))$.

VII. EXPERIMENTS AND RESULTS

We used an Intel Xeon E5-2697 processor based system called *Endeavor*, and an IBM Power8 E880 system to perform our experiments. The Intel machine configuration consists of two processors, each with 18 cores running at 2.4 GHz, thus 36 cores in total, with 45 MB unified L3 cache and 128 GB of memory. The operating system is Red Hat Enterprise Linux 6, and our code was written in C++ and compiled using the Intel C++ Composer XE 2013 compiler (version: 1.1.163) using the -O3 flag. The IBM E880 (9119-MHE) computer is a large memory machine with 8 TB memory, divided into 4 Central Processor Complexes (CPCs, also called CECs). Each CPC has 4 sockets, each socket has 12 cores, and each core can run up to 8 threads using simultaneous multi-threading (SMT). The CPU clock rate is 4.262 GHz, and the cache sizes are 64K for L1, 512K for L2 and 8MB for L3.

Our testbed consists of both real-world and synthetic graphs. For the experiments on the Intel system, we generated two classes of RMAT graphs: (a) G500 representing graphs with skewed degree distribution from the Graph 500 benchmark [18], and (b) SSCA from HPCS Scalable Synthetic Compact Applications graph analysis (SSCA#2) benchmark. We used the following parameter settings: (a) $a = 0.57$, $b = c = 0.19$, and $d = 0.05$ for G500, and (b) $a = 0.6$, and $b = c = d = 0.4/3$ for SSCA. Additionally we consider seven datasets taken from the University of Florida Matrix collection covering application areas such as medical science, structural engineering, and sensor data. We also have a large web-crawl graph [4] and a movie-interaction network [5]. For the IBM system, we solved a

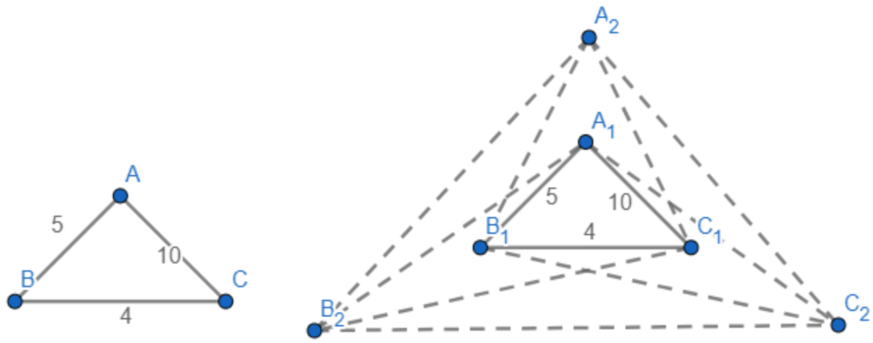

Figure 2. Reduction from a b-MATCHING to a MATCHING. (Left) Original graph, (Right) Reduced graph for $b = 2$.

larger synthetic problem (SSCA) with 268 million vertices and over 2 billion edges.

Table I shows the sizes of our testbed. There are three groups of problems in terms of sizes: six smaller problems with fewer than 90 million edges, five problems with 90 million edges or more, and one problem with over two billion edges. The real-world problems have edge weights; for the synthetic test problems, we generated three sets of weights uniformly at random in the range 0 to **intmax** for 4-byte integers. We run the MCE algorithm with these sets of weights and report results for the weight that gives the median edge cover weight. We repeat each experiment three times and report the average of the runtimes. The coefficient of variation is less than 4% for all the problems. For each triple (graph, weights, 3/2- or 2-approximation algorithm), the edge cover computed is the same, so there is no variation in the weight.

We run experiments with different $b(v) = min\{\delta(v), b\}$ values, where $\delta(v)$ is the degree of a vertex v, and $b = \{1, 2, 3, \ldots, .., 10\}$ in order to observe the impact of $b(.)$ values on the algorithms. For ease of notation, we write $b(v) = b$ rather than the minimum value mentioned earlier. We report results for $b = 5$ here.

A. *Results on the Intel Xeon System*

1) Quality Comparison: We compare the performance of the following algorithms: GREEDY, LSE, S-LSE and MCE. First, we evaluate the impact of the redundant edge removal step on the algorithms. We remind the reader that the GREEDY and LSE algorithms compute identical edge covers satisfying a 3/2-approximation guarantee; the S-LSE and MCE algorithms also compute identical edge covers that satisfy a 2-approximation guarantee. Hence we show the percent reduction in the weight of the LSE (GREEDY) and S-LSE algorithms after redundant edge removal relative to their initial weight in Table II. For the smaller problems the reduction in weight is not significant, i.e., 1.21% and 1.90% on average for the LSE and S-LSE algorithms, respectively. But for larger problems, the weight reduction is significant, 11% and 16% for the LSE and S-LSE algorithms, respectively. Note that the S-LSE algorithm benefits more from this post-processing.

Problems	$\lvert V \rvert$	$\lvert E \rvert$	**Avg. Deg.**
Fault_639	638,802	13,987,881	44
mouse_gene	45,101	14,461,095	641
Serena	1,391,349	31,570,176	45
bone010	986,703	35,339,811	72
dielFilterV3real	1,102,824	44,101,598	80
Flan_1565	1,564,794	57,920,625	74
kron_g500-logn21	2,097,152	91,040,932	87
hollywood-2011	2,180,759	114,492,816	105
G500_21	2,097,150	118,595,868	113
SSA21	2,097,152	123,579,331	118
eu-2015	11,264,052	264,535,097	47
SSCA28	268,435,456	2, 136,323,325	16

Table I
THE STRUCTURAL PROPERTIES OF OUR TESTBED FOR b-EDGE COVER, SORTED IN ASCENDING ORDER OF EDGES.

Problems	**LSE**	**S-LSE**
Fault_639	0.68%	1.33%
mouse_gene	0.95%	1.26%
Serena	0.97%	1.31%
bone010	1.97%	0.96%
dielFilterV3real	1.88%	4.11%
Flan_1565	1.33%	5.43%
Geo. Mean:	**1.21%**	**1.90%**
kron_g500-logn21	8.41%	17.02%
hollywood-2011	***15.52%***	19.74%
G500_21	11.65%	10.16%
SSA21	12.30%	14.90%
eu-2015	9.47%	***19.31%***
Geo. Mean	**11.21%**	**15.79%**

Table II
REDUCTION IN WEIGHT DUE TO REDUNDANT EDGE REMOVAL.

We obtain lower bounds on the weights of b-EDGE COVERS for a subset of the problems, using the relaxation of an integer linear program, solved with a Lagrangian optimization method [10]. This computation

Problems	Lagrange bound	Cover wt LSE	%Gap 3/2 (LSE)	%Increase 2 (MCE)
Fault_639	9.53E+15	9.77E+15	2.55%	1.13%
mouse_gene	2672.19	2898.04	8.45%	6.55%
Serena	6.93E+15	7.09E+15	2.36%	1.51%
bone010	8.21E+08	8.34E+08	1.63%	0.96%
dielFilterV3real	252.055	259.049	2.77%	0.11%
Flan_1565	5.38E+09	5.49E+09	2.02%	4.41%
SSA21	1.67E+12	1.69E+12	1.20%	4.89%
hollywood-2011	891355	922225	3.46%	1.74%
kron_g500-logn21	1.33E+06	1.35E+06	1.22%	***13.53%***
G500_21	1.35E+06	1.33E+06	1.54%	3.26%
eu-2015	9.67E+06	1.11E+07	14.62%	2.33%
Geo. Mean				***2.14%***

Table III
LOWER BOUND ON THE WEIGHT OF EDGE COVERS, AND THE INCREASE IN WEIGHT COMPUTED BY THE 2-APPROXIMATION ALGORITHMS RELATIVE TO THE 3/2-APPROXIMATION ALGORITHMS ($b = 5$).

also uses a shared memory multi-threaded algorithm on 20 cores of an Intel Xeon. All the reported bounds are computed within an hour. The maximum number of iterations is set to $10,000$ and the maximum run time is set to 2 hours. If there is no improvement in the solution in $1,000$ consecutive iterations, the program is terminated.

In Table III, the second column shows the lower bound, the third column shows weight of the cover from the LSE algorithm, the fourth column shows the gap between the third and second columns, and the fifth column shows the weight difference between the LSE and MCE algorithms. The results show that the weights computed are close to the minimum values, and that the two approximations are close to each other in practice. The gap is relatively large for the `mouse_gene` and `eu-2015` problems; unfortunately we cannot tell if the Lagrange bound is lower than the optimal edge cover weight, or if the LSE algorithm computes an edge cover with weight greater than the optimal. One of these is a relatively dense graph, and the other is one of the largest graphs in the test set, and the Lagrange bound computation might obtain higher values if run longer.

Generally we can conclude that if an application does not require the optimal b-EDGE COVER, we may use any of these approximation algorithms, and the faster and scalable algorithms are to be preferred. We identify these in the next set of experiments.

2) Serial Performance: We compare the serial run time performance of the four algorithms in Figure 3. Note that the times are plotted on a logarithmic scale; we cut off the run times after one hour. For large instances, we observe that usually the LSE algorithm is $2-5\times$ faster than the GREEDY algorithm, the S-LSE algorithm is $2-4\times$ faster than the LSE algorithm, and the MCE algorithm is roughly one order of magnitude faster than the S-LSE algorithm. The difference increases with increasing values of $b(v)$. It is clear from the results in Figure 3 that the MCE algorithm is the fastest serial approximation algorithm for the b-EDGE COVER problem, and so we use its performance to evaluate the parallel shared memory performance next.

	Number of Threads							
1 CPC	**1**	**12**	**24**	**36**	**48**	**96**	**192**	**384**
Runtime	3107	303	205	91	71	35	24	22
	Number of Threads							
4 CPC	**1**	**12**	**24**	**48**	**96**	**191**	**382**	**764**
Runtime	3107	303	205	71	38	20	19	23

Table IV
PARALLEL RUN-TIMES FOR **SSA28** ON TWO CONFIGURATIONS OF THE IBM POWER8 USING THE MCE ALGORITHM. THE FIRST SET USED 1 CPC AND ONE THREAD PER CORE UP TO 48 THREADS; AFTER THAT, SMT WAS EMPLOYED TO INCREASE THE NUMBER OF THREADS. ON THE SECOND SET, WE USED 4 CPCS, 191 CORES, ONE THREAD PER CORE INITIALLY; AFTER 191 THREADS, SMT WAS EMPLOYED.

3) Parallel Performance: We have evaluated the performance of MCE, S-LSE, and LSE algorithms using 36 cores of the Xeon ES-2697 multiprocessor. Each core is hyper-threading enabled with two threads per core, i.e., we have a total of 72 threads. Unfortunately, for these problems, hyper-threading does not help, and we use 36 threads to compute the run time performance of the LSE and S-LSE algorithms relative to the MCE algorithm, and report this in Figure 4. A runtime value greater than 1 implies that the MCE algorithm is faster. We observed only one case, *SSA21* with $b(.) = 1$, where the S-LSE algorithm beats the MCE algorithm. But with higher $b(.)$ values the MCE algorithm becomes the fastest for all problems, by a factor of 10 relative to the LSE and S-LSE algorithms.

We present strong scaling results for the MCE algorithm in Figure 5. We observe that for smaller problems, the MCE algorithm does not scale beyond 18 threads, but for most of the larger problems, the algorithm shows a speedup of $35\times$ with 36 threads.

B. Results on the IBM System

Now we experiment with a 2 billion edge graph on a TB-scale shared memory machine using the MCE algorithm. The IBM Power8 E880 system is organized into 4 Central Processor Complexes (CPCs); each CPC has four sockets, each socket has 12 cores, and each core can run a maximum of 8 threads using hyperthreading. We computed a 5-edge cover in the SSCA28 graph, with weights chosen randomly as for the Intel test, and obtained an edge weight of $4.79e15$ for all experiments since they compute the same edge cover. We conducted two experiments: The first runs one thread on each core of a single CPC, and then uses SMT on every core to obtain more threads. For SSCA28, a speedup of 131 is obtained on 192 threads, showing that four-way SMT on a CPC is quite effective for this problem. In the second experiment, we ran one thread each on the 191 cores of the 4 CPCs (one core is reserved for system use), and after that used SMT. In this case, the best speedup of 153 was obtained for one thread on 191 cores for this problem. The incremental increase in speedup for larger numbers of threads was small, and performance seems to be limited by memory latency since the code does not exceed the memory bandwidth available.

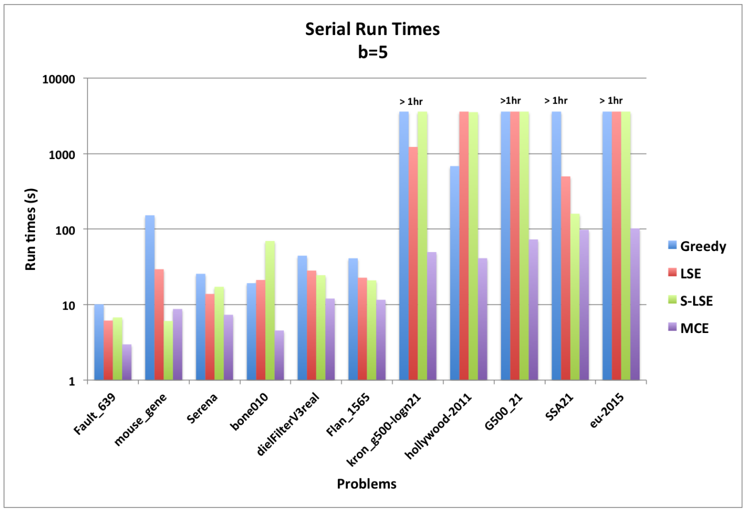

Figure 3. Serial run times of four approximation algorithms for b-EDGE COVER on the Intel Xeon.

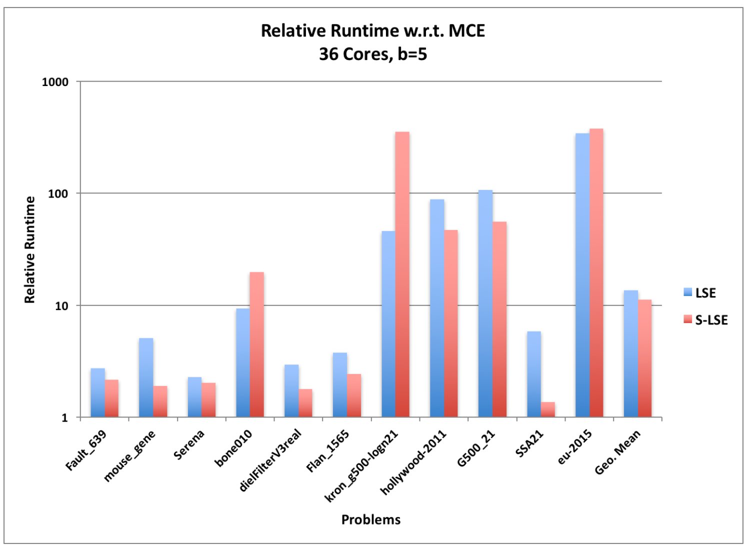

Figure 4. Relative runtimes of LSE and S-LSE algorithms w.r.t. MCE algorithm on 36 cores of an Intel Xeon.

VIII. CONCLUSIONS

We have shown how parallel algorithms for b-EDGE COVERS can be designed using the approximation paradigm. The MCE algorithm is faster than other approximation algorithms for this problem by an order of magnitude or more; it also scales to compute edge covers in a graph with billions of edges using hundreds of threads on a Terabyte-scale shared-memory multiprocessor. By computing lower bounds, the edge covers are seen to have weights within a few percent of the minimum values.

Acknowledgements. This research was supported by NSF grant CCF-1637534; DOE grants ASCR DE-SC001020; and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. DOE Office of Science and the NNSA; and an Intel Parallel Computing Center. We are grateful to the IBM Systems Poughkeepsie Client Center for access to a Power8 Cluster.

REFERENCES

[1] A. AZAD, A. BULUC, X. S. LI, X. WANG, AND J. LANGGUTH, *A distributed-memory approximation algorithm for maximum weight perfect bipartite matching*. ArXiv:1801.09809, 2018.

[2] G. E. BLELLOCH, J. T. FINEMAN, AND J. SHUN, *Greedy sequential maximal independent set and matching are parallel on average*, in SPAA, 2012, pp. 308–317.

[3] G. E. BLELLOCH, R. PENG, AND K. TANGWONGSAN, *Linear work parallel greeedy approximate set cover and variants*, in SPAA, 2011, pp. 23–32.

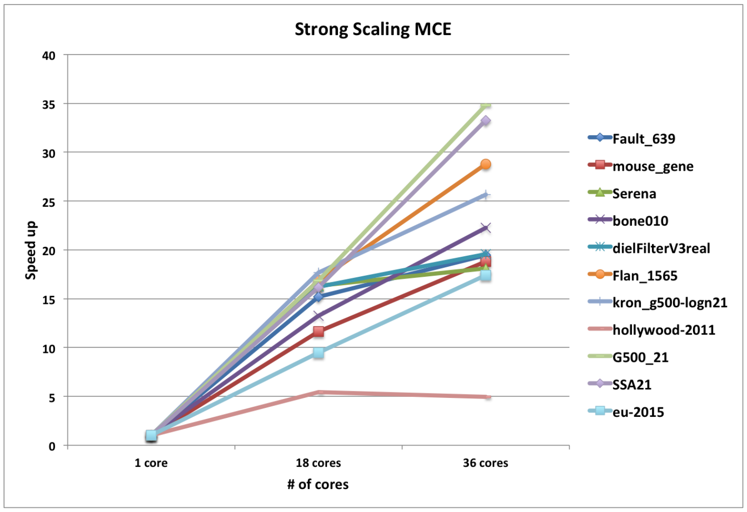

Figure 5. Strong scaling of the parallel MCE algorithm on the Intel Xeon.

[4] P. Boldi, A. Marino, M. Santini, and S. Vigna, *BUbiNG: Massive crawling for the masses*, in Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web, 2014, pp. 227–228.

[5] P. Boldi and S. Vigna, *The WebGraph framework I: Compression techniques*, in WWW 2004, ACM Press, 2004, pp. 595–601.

[6] K. M. Choromanski, T. Jebara, and K. Tang, *Adaptive anonymity via b-matching*, in NIPS, 2013, pp. 3192–3200.

[7] V. Chvatal, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C.Stein, *Introduction to Algorithms*, MIT Press, 2009.

[9] G. Dobson, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Mathematics of Operations Research, 7 (1982), pp. 515–531.

[10] M. L. Fisher, *The Lagrangian relaxation method for solving integer programming problems*, Management Science, 50 (2004), pp. 1861–1871.

[11] A. Khan and A. Pothen, *A new 3/2-approximation algorithm for the b-edge cover problem*, in Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, 2016, pp. 52–61.

[12] A. Khan, A. Pothen, M. M. Patwary, M. Halappanavar, N. Satish, N. Sundaram, and P. Dubey, *Designing scalable b-matching algorithms on distributed memory multiprocessors by approximation*, in Proceedings of ACM/IEEE Supercomputing, 2016, pp. 773–783.

[13] A. Khan, A. Pothen, M. M. Patwary, N. Satish, N. Sundaram, F. Manne, M. Halappanavar, and P. Dubey, *Efficient approximation algorithms for weighted b-matching*, SIAM Journal on Scientific Computing, (2016), pp. S593–S619.

[14] S. Khuller, U. Vishkin, and N. Young, *A primal-dual parallel approximation technique applied to weighted set and vertex covers*, Journal of Algorithms, (1994), pp. 280–289.

[15] G. Kortsarz, V. Mirrokni, Z. Nutov, and E. Tsanko, *Approximating minimum-power degree and connectivity problems*, in Proceedings of 8th Latin American Theoretical Informatics Conference (Lecture Notes in Computer Science), vol. 4957, 2008, pp. 423–435.

[16] F. Manne and R. Bisseling, *A parallel approximation algorithm for the weighted matching problem*, in Proceedings of International Conference on Parallel Processing and Applied Mathematics, 2007, pp. 708–717.

[17] F. Manne and M. Halappanavar, *New effective multithreaded matching algorithms*, in IPDPS, 2014, pp. 519–528.

[18] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, *Introducing the Graph 500*, Cray User's Group, (2010).

[19] R. Preis, *Linear time $1/2$-approximation algorithm for maximum weighted matching in general graphs*, in Proceedings of STACS, (Lecture Notes in Computer Science), vol. 1563, 1999, pp. 259–269.

[20] S M Ferdous, A. Khan, and A. Pothen, *New approximation algorithms for minimum weighted edge cover*, in Proceedings of SIAM Conference on Combinatorial Scientific Computing, 2018. To appear.

[21] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency. Volume A: Paths, Flows, Matchings*, Springer, 2003.

[22] K. Tangwongsan, *Efficient Parallel Approximation Algorithms*, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2011.