Algorithms for Balanced Graph Colorings with Applications in Parallel Computing

Hao Lu, Mahantesh Halappanavar, Daniel Chavarría-Miranda, Assefaw H. Gebremedhin, Ajay Panyala, and Ananth Kalyanaraman, *Member, IEEE*

Abstract—Graph coloring—in a generic sense—is used to identify subsets of independent tasks in parallel scientific computing applications. Traditional coloring heuristics aim to reduce the number of colors used as that number also corresponds to the number of parallel steps in the application. However, if the color classes produced have a skew in their sizes, utilization of hardware resources becomes inefficient, especially for the smaller color classes. Equitable coloring is a theoretical formulation of coloring that guarantees a perfect balance among color classes, and its practical relaxation is referred to here as balanced coloring. In this paper, we consider balanced coloring models in the context of parallel computing. The goal is to achieve a balanced coloring of an input graph without increasing the number of colors that an algorithm oblivious to balance would have used. We propose and study multiple heuristics that aim to achieve such a balanced coloring for two variants of coloring problem, distance-1 coloring (the standard coloring problem) and partial distance-2 coloring (defined on a bipartite graph). We present parallelization approaches for multi-core and manycore architectures and cross-evaluate their effectiveness with respect to the quality of balance achieved and performance. Furthermore, we study the impact of the proposed balanced coloring heuristics on a concrete application-viz. parallel community detection, which is an example of an irregular application. In addition, we propose several extensions to our basic balancing schemes and evaluate their balancing efficacy and performance characteristics. The thorough treatment of balanced coloring presented in this paper from algorithms to application is expected to serve as a valuable resource to parallel application developers who seek to improve parallel performance of their applications using coloring.

Index Terms—Balanced coloring, parallel graph coloring, distance-1 coloring, partial distance-2 coloring, Tilera manycore architecture, community detection, graph algorithms

1 Introduction

Decomposing a computational task into constituent parts that can be executed simultaneously or identifying elements of composite data that can be safely updated simultaneously is a pervasive primitive in parallel computing. An associated need is that of scheduling the identified subtasks (or data update operations) onto the processing units of a platform. In such a scenario, one would, for performance reasons, need to both maximize the amount of parallel execution (or data update) attained in a given step *and* minimize the total number of steps needed. In cases where the computational or data dependency between entities can be abstracted using a *graph*, this dual objective can be modeled and solved as a graph coloring problem.

In this paper, we consider two types of graph coloring problems: *distance-1 coloring*, which is defined on a general (unipartite) graph, and, *partial distance-2 coloring*, which is defined on a bipartite graph. A distance-1 coloring of a general graph $G(V, E)$ is an assignment of colors to vertices such that any two adjacent vertices receive different colors. A partial distance-2 coloring of a bipartite graph $G_b = (V_1, V_2, E)$ is an assignment of colors to one of the vertex sets, say V_2, such that any two vertices in V_2 that are two edges away from each other receive different colors. More formal definitions of these problems will be given in Sections 2 and 5, but we remark at this point that distance-1 coloring is the "usual" graph coloring problem.

Standard formulations of the distance-1 and partial distance-2 coloring problems aim at minimizing the number of colors used (that is, the number of independent subsets or color classes) without any requirement on the size of the color classes relative to each other. They therefore permit cases where the color classes can be highly unbalanced. In fact, by their nature, most practical algorithms for the standard formulations of these graph coloring problems produce highly skewed color classes. This will be undesirable as the smaller color classes may *not* provide sufficient workload for parallel efficiency.

In this paper, we deal with the design, implementation and performance evaluation of algorithms for the distance-1 and partial distance-2 coloring problems that also require that color classes be *balanced* in their sizes.

There is a body of work in the graph theory literature on *equitable* distance-1 colorings—a formulation in which color classes are required to be *perfectly* balanced—but little work exists on fast, practical, balanced coloring algorithms and their parallelization on contemporary and emerging

- *H. Lu, A.H. Gebremedhi, and A. Kalyanaraman are with Washington State University, Pullman, WA 99164. E-mail: luhowardmark@wsu.edu, {assefaw, ananth}@eecs.wsu.edu.*
- *M. Halappanavar, D. Chavarría-Miranda, and P. Panyala are with Pacific Northwest National Laboratory, Richland, WA 99354. E-mail: {hala, daniel.chavarria, ajay.panyala}@pnnl.gov.*

Manuscript received 29 Oct. 2015; revised 31 Aug. 2016; accepted 9 Sept. 2016. Date of publication 21 Oct. 2016; date of current version 12 Apr. 2017. Recommended for acceptance by R. Vuduc.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2620142

platforms. Further, to the best of our knowledge, there exists no prior work on balanced partial distance-2 coloring. We seek to address these deficiencies.

The scope and contributions of this paper are as follows:

- *Algorithms*. We investigate a variety of techniques, grouped in two main categories, for achieving balanced coloring in both the distance-1 and partial distance-2 coloring cases. Algorithms in the first category aim to obtain a balanced coloring in a single attempt (*ab initio*). Those in the second category begin with an initial coloring oblivious to balance and use that information to produce a new coloring that is also balanced (*guided*). We propose three different types of guided balanced coloring algorithms (each of which could further be specialized to tune for performance) and examine several variants of *ab initio* approaches.
- *Parallelization*. We parallelize all of the balanced coloring algorithms we explore targeting two different architectures: conventional multicore x86 architectures and a specialized many-core platform, Tilera TileGx36.
- *Application and Evaluation*. We demonstrate the impact of balanced distance-1 coloring on *community detection*—a widely used graph application. Our results show that using balanced coloring for this application could yield significant performance improvements while preserving quality. In addition, we present a thorough evaluation of the different heuristics based on their balancing efficacy and performance.

A preliminary version of this paper that focused only on distance-1 coloring has appeared in [1]. The remainder of the paper is organized as follows. We provide background and motivate our work in Section 2. We describe the sequential versions of the various algorithms we explore for balanced distance-1 coloring in Section 3 and discuss how they are parallelized in Section 4. We discuss how the algorithms developed for distance-1 coloring are modified to handle the partial distance-2 coloring case in Section 5. We review essential features of the platforms for which the implementations are targeted in Section 6. We present and discuss experimental results in Section 7. We conclude in Section 8.

2 Background

2.1 Basics of Balanced Distance-1 Coloring

Given a general graph $G = (V, E)$, a distance-1 coloring of G is an assignment of colors to vertices such that any two adjacent vertices (which are at distance 1 from each other) are assigned different colors. This is the "usual" coloring problem, and henceforth, for brevity, we will drop the qualifier "distance-1" unless we need to distinguish it from distance-2 coloring.

A coloring is said to be *equitable* if the sizes of any two color classes differ by at most one. The concept of equitable coloring was introduced by Meyer in a 1973 paper [2]. Its history, however, goes even further back to a conjecture by Erdös, a conjecture settled in 1970 by Hajnal and Szemerédi [3] forming their celebrated theorem: a graph with maximal degree Δ is equitably k-colorable if $k \geq \Delta + 1$. This bound is sharp. One of the directions of early theoretical research in this field had been to obtain better upper bounds than $\Delta + 1$ for special graph classes [4].

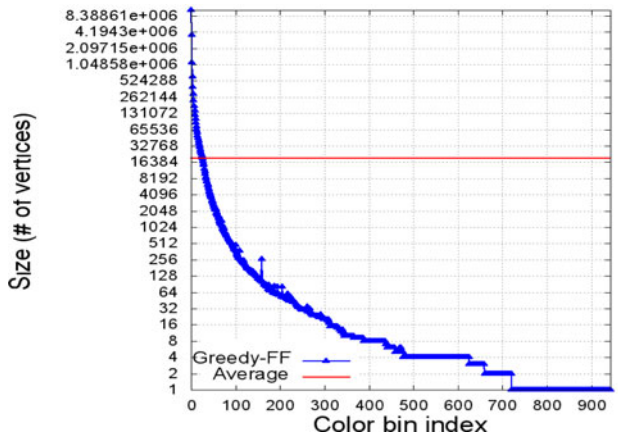
The equitable coloring problem asks for an equitable k-coloring with the smallest possible k. This problem is NP-hard, as the classical coloring problem can be trivially reduced to it. Polynomial time equitable coloring algorithms are known for various special classes of graphs, including trees [5], r-partite graphs [6], line graphs [6], and planar graphs [7]. Furmanczyk [8] provides a survey of work on equitable colorings until the early 2000's.

In equitable coloring, as stated earlier, the difference in size between any pair of color classes is required to be at most one. This ideal can for some practical needs be unnecessarily stringent and too costly to attain. In the closely related *heuristic* variant we refer to here as *balanced coloring*, the restriction is relaxed; the difference in color class size instead is allowed to be at most a "small" number greater than 1. One formal way to state this is to say that each color class is bounded by some parameter l. Bodleander and Fomin [10] study this problem and show that it, as well as the equitable coloring problem itself, can be solved in polynomial time for graphs with bounded treewidth. In this work, we take a less formal route and think of balanced coloring without fixing a parameter l. More specifically, given a graph $G(V, E)$, the problem is to compute a distance-1 coloring such that each color class receives approximately $\frac{|V|}{C}$ vertices, where C is the number of colors used.

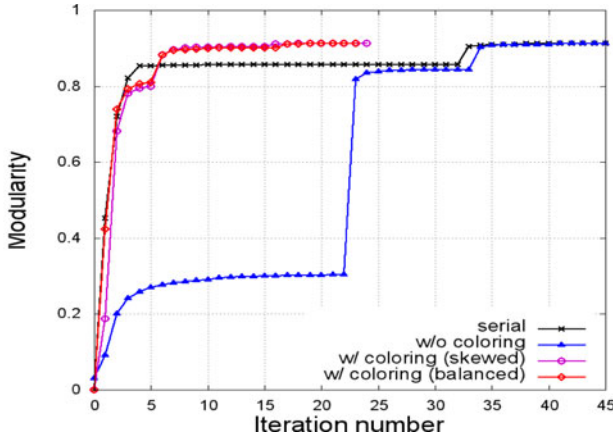
Equitable coloring and balanced coloring (in the sense just mentioned) find important applications in various areas. Examples include load-balanced partitioning for domain decomposition methods [11], parallel sparse matrix computations on irregular grids [12], and various types of scheduling and timetabling problems [13]. Tucker in a 1973 paper [14] discusses how equitable coloring theory has been used in helping out Operations Researchers at the Urban Science Department at Stony Brook, who were faced with a challenging routing problem that sought to optimize scheduling of garbage collecting trucks in the city.

Balanced coloring in the context of parallel scientific computing was studied by Gjertsen, Jones and Plassmann [15], where they developed a balanced, distributed memory parallel coloring heuristic building on their own earlier work on parallel graph coloring that was unconcerned with balancing color classes [16]. Their balancing heuristic draws ideas from approximation algorithms for the *bin packing* problem [17] and a coloring work in [18].

The work in [15] has connections to that in this paper, but it differs both in terms *context* and *approach*. In particular, the context in [15] is a distributed memory setting in which the vertex set of a graph is already partitioned among processors. Further, the authors assume that the partitioning is a good one in the sense that each processor is assigned nearly the same number of vertices. Based on an initial coloring of the partitioned graph, the authors then run a balancing heuristic that respects the vertex partitioning (avoids relocation of vertices to processors). In contrast, in this work, we do not assume any *a priori* partitioning of the vertex set. In fact, the assignment of vertices to processors (or threads) is expected to be done after the balanced coloring is achieved, which is an advantage. In terms of approach, the work in [15] focuses on one class of algorithms: given an initial coloring, how can balancing be achieved without increasing the number of colors used? In contrast, here we consider a wider variety of algorithms, provide implementations on modern day multi-core



(a) Greedy Coloring, Input: uk-2002



(b) Community Detection, Input: cnr

Fig. 1. a) The size distribution of the color classes obtained by the Greedy First Fit heuristic for distance-1 coloring on an input graph (*uk-2002*) obtained through a web crawl of the .uk domain. b) The evolution of modularity gain across the iterations of a parallel implementation of the Louvain method [19]. Four curves are depicted there. Two of the curves correspond to results obtained when coloring (skewed and balanced) is used in the parallel implementation, the third corresponds to results when coloring is *not* used, and the fourth corresponds to results on a serial implementation.

and manycore platforms, and experimentally evaluate their performance as well as the trade-offs they offer.

2.2 A Foundational Scheme

For the standard distance-1 graph coloring problem, despite its NP-hardness, the greedy scheme outlined in Algorithm 1 is often found quite effective in practice, since the scheme gives usable solutions and can be implemented to run in linear-time for graphs that arise in practice.

Algorithm 1. Greedy

```
Greedy (G = (V, E))
  for each v ∈ V in some order do
    for each w adjacent to v do
      Mark the color of w as forbidden to v
    Assign v a color not marked as forbidden to v
```

The scheme Greedy can be specialized in a variety of ways depending on a) the technique used to determine the order in which the vertices are processed and b) the strategy used to pick a color (among a set of permissible colors) for a vertex at a given step.

A common strategy with regards to (a) is to rank the vertices in a non-ascending order of "degree", where degree is suitably defined (e.g., as the number of neighbors, or the number of already colored neighbors, or the number of differently colored neighbors). The intuition is to treat vertices that are likely to be harder to color, earlier in the process.

A common strategy used with regards to (b) is to pick the *smallest* (we assume colors are positive integers) permissible color for a vertex in each step. This strategy is sometimes referred to as First Fit (FF), since, considering the analogy to the bin packing problem mentioned earlier, it strives to place the vertex in the first bin (color) it could be placed in. The rationale behind choosing the smallest color is that one can then *guarantee* that the number of colors used by the scheme is bounded from above by $\Delta + 1$ (where Δ is the maximum degree in the graph) regardless of the order in which the vertices are processed and by $K + 1$ (where K is the *core number* of the graph) if the degeneracy order of the vertices is used. A degeneracy order, also known as Smallest Last ordering, can be obtained in linear-time.

The FF strategy is attractive for the bounds on the number of colors it assures. The color classes it produces, however, could be highly skewed, with a vast majority containing significantly smaller number of vertices—an expected result out of selecting the first available bin for every vertex. The chart in Fig. 1a confirms this expectation on a real world graph. Small-sized color classes can become scalability bottlenecks in an end-application, where typically the color classes are processed in different steps (to honor dependencies) and the smaller classes limit the degree of parallelism during those steps.

2.3 Community Detection: A Motivating Application

Overcoming such scalability bottlenecks is in part what motivated our current work. We sought to investigate algorithms for achieving balanced coloring and their effective use in parallel computing applications. As a case-study, we consider balanced coloring in the context of parallel community detection, based on an implementation called "*Grappolo*" that we developed for multi-core and manycore architectures [19], [20]. The parallel implementation is based on the sequential Louvain heuristic [21]. The Louvain method, which is one of the most widely used community detection algorithms, uses the modularity function [22] as the objective function to be maximized.

Grappolo consists of multiple phases, each in turn containing multiple iterations. Within each phase, the algorithm starts with every vertex placed in a community of its own. A series of iterations is then performed until a convergence criterion is met. Within each iteration, all vertices are scanned in parallel. For each vertex, a greedy decision is made as to whether the vertex should migrate to a different community (selected from one of its neighbors) or should stay in its current community, so as to maximize the net modularity gain. This approach places multiple constraints on concurrent processing of neighboring vertices. In previous work, we had extensively explored the use of graph coloring in effectively addressing the challenges associated with these constraints [19]. Our findings showed that the use of coloring significantly accelerates convergence and, for many input cases, also improves the quality of communities output (as measured by the modularity function).

TABLE 1
A Comprehensive List of Balancing Strategies for Distance-1 Coloring Presented and Studied in This Paper

Strategy	Category	Description
Greedy-LU	*ab initio*	Run Algorithm 1 with LU color choice among permissible colors.
Greedy-Random	*ab initio*	Run Algorithm 1 with Random color choice among permissible colors.
Shuffling-unscheduled	guided	Run Algorithm 1 with FF color choice strategy. Based on the obtained coloring identify over-full and under-full bins. Move select vertices from over-full to under-full bins without changing the number of color classes. Further specializations include Vertex-centric FF (*VFF*) and Color-centric LU (*CLU*).
Shuffling-scheduled	guided	Run Algorithm 1 with FF color choice strategy. Based on the obtained coloring identify over-full and under-full bins. Move select vertices from over-full to under-full bins in a scheduled manner without changing the number of color classes.
Recoloring	guided	Run Algorithm 1 with FF color choice strategy. Let the number of colors used be C. Let $\gamma = \lvert V\rvert/C$. Construct an ordered set of vertices $W = \{V(C), V(C-1), \ldots, V(1)\}$, where $V(i)$ denotes the set of vertices having the color i. Re-color vertices in W in that order using Algorithm 1 such that in each step, a vertex v is assigned the smallest permissible color k such that the size of bin k is less than γ.

The input graph is denoted by $G = (V, E)$. The same set of strategies are also extended to obtain a balanced partial distance-2 coloring on bipartite graph inputs.

However, since the color classes are processed in parallel one at a time, large skews in color class sizes (as shown in Fig. 1a) can reduce overall scalability, particularly while processing the smaller color classes. The purpose of balancing the color classes is thus to improve thread utilization for those smaller color classes, while ensuring that the overall output quality of the solution (modularity) is maintained.

The chart in Fig. 1b demonstrates this purpose. The chart shows that balanced coloring matches skewed coloring in its impact on community detection both in terms of convergence rate (i.e., number of iterations taken to complete) and in terms of output quality (final modularity), while offering the added advantage of improved thread utilization within every iteration, since the color classes are balanced.

3 Algorithms for Balanced Distance-1 Coloring

In this section, we present multiple heuristics to compute a balanced distance-1 coloring of an input graph, as summarized in Table 1.

We explore two categories of approaches. Approaches in the first category aim at obtaining a balanced coloring in a single attempt. We refer to these as "*ab initio*" approaches. Those in the second category follow a two-step procedure, where an initial coloring obtained using a balance-oblivious procedure, is subsequently balanced in the second step. We refer to these approaches as "guided" (to signify that they are guided by an initial coloring).

3.1 *Ab initio* Balancing Strategies

Within the *ab initio* category, we consider two well-known variants of the Greedy scheme outlined in Algorithm 1 that differ in how the choice of color to be assigned to a vertex in each step is done. Both variants seek to achieve balanced coloring by virtue of the color choice strategy:

- *Greedy-LU*: A vertex is assigned the *least used* color among all currently available permissible colors. If no permissible color exists, then a new color is created and assigned to the vertex.
- *Greedy-Random*: A vertex is assigned a color picked at random from the set of permissible colors. The particular Greedy-Random variant we consider assumes the existence of a reasonable bound B on the number of colors needed. One such easy-to-compute bound is $B = \Delta + 1$. Then, a vertex v is assigned a randomly chosen color from the set of permissible colors $P(v) \subseteq \{1, 2, \ldots, B\}$.

Manne and Boman analyze balanced greedy coloring using the strategies LU and Random in the context of sparse random graphs [23].

3.2 Guided Balancing Strategies

In the guided category, we study different approaches for obtaining a balanced coloring given an initial coloring. We note here that *all* of the proposed guided approaches can be applied to an initial coloring produced by an arbitrary coloring method. However, a subset of these approaches is designed to exploit certain properties of an initial coloring produced by the Greedy coloring scheme that uses the FF color choice strategy (henceforth abbreviated as Greedy-FF).

Given an input graph $G = (V, E)$, let the number of colors used by the initial coloring step be C. In all our guided strategies, we make use of the quantity $\gamma = |V|/C$ to guide our methods. Note that in a strictly balanced setting, the size of each color class would be roughly γ. Consequently, we refer to a color class of size greater than γ as an *over-full bin*, and a color class of size less than γ as an *under-full bin*. (We use the terms *bin* and *color class* interchangeably throughout the paper.)

Broadly, we classify our guided strategies into two types. In the first type, a subset of vertices from each over-full bin is moved to under-full bins so that a better balance is attained. Since this is achieved *without* increasing the number of color classes, we refer to this type of methods *Shuffling*-based. In the second type, instead of enforcing that the number of color classes remains unchanged, *all* vertices are colored afresh, this time with a balance constraint imposed. We call this strategy *Recoloring*.

The Shuffling methods in turn comprise two specializations: *unscheduled* and *scheduled* moves. The motivation for this distinction comes from parallel performance needs that will be explained in Section 4.

The Recoloring method takes advantage of an interesting property of the Greedy-FF scheme. Suppose a coloring of a graph $G = (V, E)$ is obtained using Greedy-FF in some vertex order. Let the number of colors used be C. Now suppose the vertices of G are ordered such that vertices in the same color class are listed consecutively. Then re-applying Greedy-FF using this new ordering will produce a new coloring of G using C or fewer colors. Culberson [24] applied this idea iteratively in his method called Iterated Greedy (IG) to successively reduce the number of colors and draw the number as close to the optimal as possible. There is a degree of freedom in how the color classes themselves could be ordered for IG to be successful. One of the better strategies is to list the color classes in reverse order—i.e., beginning from the vertices of the highest color index.

We build on this property to devise our Recoloring method for balancing. A key extension in our case is that we maintain the sizes of bins during the new coloring and use those to impose balance. In particular, in each step of the recoloring, a vertex is assigned the smallest permissible color k such that the size of the bin k is less than γ.

4 Parallel Algorithms

We parallelized all of the guided balanced coloring algorithms presented in Section 3 for the shared memory model. For each heuristic we developed two OpenMP-based implementations—one for conventional multicores and another for the Tilera manycore platform. To obtain the initial coloring we used a parallel implementation available for Greedy-FF from a previous effort [25]. In this section, we describe the parallel algorithms underlying the implementations of the balancing schemes.

To parallelize our shuffling-based approaches, we considered two ways of moving a vertex from an over-full bin to an under-full bin. The first type of move is "*unscheduled*". Here, the choice of the target bin for a given vertex is decided dynamically (using either the FF or LU strategy) based on the state of the color bins—encompassing both size and composition. This approach strives to achieve a good balance, if possible; as a trade-off, however, it entails the cost needed to keep each dynamic state up-to-date. More specifically, concurrent updates to the sizes of the same bin need to be synchronized.

To mitigate the cost of updates, we explored an alternative that we call "*scheduled*" moves, where the target bin for a vertex in an over-full bin is statically determined using a heuristic, and the check to verify if such a move is permissible is deferred until the move is actually attempted. If a move attempt creates a "conflict", which is possible if a neighboring vertex is already in the same target bin, no further attempt is made and the vertex remains in its original bin. The advantage of this approach is the expected improvement in parallel performance, as no atomic operation or lock is needed to update bin sizes. However, this approach could potentially leave bins unbalanced.

4.1 Parallelization using Unscheduled Moves

For obtaining guided balanced coloring using unscheduled moves, we considered two parallelization schemes. In the *vertex-centric* schemes, the loop-parallelization is around a set of vertices, and vertices from different color classes are allowed to be processed concurrently. In the *color-centric* schemes, vertices processed concurrently must belong to the same color class. In both schemes, only vertices in over-full bins are considered for color reassignment. Furthermore, once an over-full bin i reaches balance (i.e., size reaches γ) at any point in the execution, then vertices from that bin are no longer considered for color reassignment. Hence, these schemes represent partial recoloring methods that proceed until either a balance is achieved or a balance is no longer possible (i.e., there exist no more permissible moves from any of the remaining over-full bins).

Vertex-Centric Parallelization Scheme. Processing vertices from possibly different color classes exposes maximum concurrency. However, it could also cause conflicts. To handle such conflicts in parallel we adopt the *Speculation-and-Iteration* framework described in [25]. The basic idea in this framework is to maximize concurrency by temporarily tolerating inconsistencies. Consider a simple loop-parallelization over the set of vertices in the Greedy scheme (using FF or LU) outlined in Algorithm 1. Such a parallelization will not preclude the possibility of a pair of adjacent vertices from receiving the same color. In our adoption of the speculation-and-iteration framework, once vertices are moved to their target color classes, the idea is to detect conflicts (in parallel) in a separate phase in the same round and resolve them in a subsequent round. The algorithm proceeds iteratively in this fashion until all conflicts are resolved.

A template for the vertex-centric parallelization scheme is presented in Algorithm 2. This algorithm corresponds to the *Vertex-centric First Fit* (*VFF*) balancing method. It should be easy to see that the same algorithm can be easily adapted to the *Vertex-centric Least Used* (*VLU*) balancing method with a change to the target bin (k) selection criterion.

Algorithm 2. Vertex-Centric Parallel Scheme for Balanced Coloring (using FF)

```
VertexParallelGuidedBalancing(G = (V, E))
  Obtain an initial coloring of G
  Let U be the set of vertices from over-full bins
  while U ≠ 0 do
    for v ∈ U do in parallel
      Let k be the smallest index of an under-full bin that is
          permissible                                   ▷FF
      if k exists then
        Let j ← color[v]
        color[v] ← k
        Update size of bins k and j                   ▷synch. step
    R ← ∅
    for v ∈ U do in parallel                          ▷check for conflicts
      for w ∈ adj(v) do
        if (color[w] = color[v] and v > w) then
          R ← R ∪ {v}
          Update size of bin color[v]                 ▷synch. step
    U ← R
```

Note that the maximum number of conflicts per a vertex v in the above algorithm can be upper-bounded by $\min\{d(v), b\}$, where $d(v)$ is the number of vertices adjacent to v and b is the number of under-full bins. This upper-bound is rather weak. In practice, we observed that the closely related quantity—the actual number of iterations needed to clear all conflicts—is typically a small constant.

Color-Centric Parallelization Scheme. In the color-centric scheme for parallelization, we allow only vertices from the same color class to be processed concurrently. This is achieved by processing one over-full bin at a time and performing the moves departing from that over-full bin in parallel until a balance is achieved or no more move is possible. This scheme, therefore, avoids conflicts, and the balancing procedure requires no more than a single pass of the over-full bins. However, the trade-off is in parallel performance of the balancing procedure, which requires as many parallel steps as there are number of over-full bins in the initial coloring. Depending on the strategy used to pick an under-full bin (FF or LU), we refer to this Color-centric parallelization scheme as either CFF or CLU. A template for the color-centric scheme is shown in Algorithm 3.

Algorithm 3. Color-Centric Parallel Balanced Coloring

```
ColorParallelGuidedBalancing(G = (V, E))
  Obtain an initial coloring of G
  Let Q be the set of over-full bins
  for each j ∈ Q do
      Let V(j) denote the set of vertices with color j
      for v ∈ V(j) do in parallel
          Let k be the smallest index of an under-full bin that
              is permissible to v                          ▷FF
          if k exists then
            color[v] ← k
            Update size of bin k, j                  ▷synch. step
```

Initial Coloring. We note here a special property emerging from the use of Greedy-FF for generating the initial coloring. Any initial coloring produced by Greedy-FF satisfies the following property: Assume a linear ordering of colors from $1, \ldots, C$. If a vertex v is assigned color j, where $j > 1$, then it implies that v contains at least one neighbor in each of the previous colors $1, \ldots, j-1$ (otherwise, v would have been assigned a smaller color). Therefore, if we follow the Greedy-FF initial coloring by another FF-based strategy during the subsequent *balancing* step (e.g., VFF or CFF), then the closest permissible bin, say k, we identify through that procedure would also correspond to a color that has a high incidence of edges on the source over-full color bin. Given that k represents a permissible bin despite its high incidence makes it intuitively an attractive target for this vertex. On the other hand, an LU-based strategy (VLU or CLU) operates oblivious to the ordering of the initial colors, and is therefore better suited for scenarios where the initial coloring was generated by schemes *other than* Greedy-FF.

It is for these reasons that we use the Greedy-FF strategy for computing an initial coloring in VFF and CFF, while for VLU and CLU the use of any initial coloring scheme is reasonable.

4.2 Parallelization using Scheduled Moves

To parallelize guided balancing using scheduled moves we take advantage of both the incidence property (observed above) and another size-related property of the Greedy-FF initial coloring: owing to its First Fit strategy, Greedy-FF is expected to assign more vertices to smaller-indexed color classes. In other words, color classes are expected to be in non-increasing order of their sizes as one proceeds from color 1 through color C. This expectation agrees with the size distributions depicted in Fig. 1.

Our parallel algorithm with scheduled moves is outlined in Algorithm 4. Intuitively, we identify an arbitrary subset of surplus vertices from the sequence of over-full bins and mark each of them for assignment to a corresponding under-full color.[1] At this point, no explicit checks are made to identify conflicts. In the next step, all vertices from the over-full bins that were scheduled for recoloring are processed in parallel to check if any of them conflicts with the assigned target bin. A move is completed only if it generates no conflicts.

Algorithm 4. Parallel Shuffling using Scheduled Moves

```
ScheduledBalancing(G = (V, E))
  Obtain an initial coloring of G using Greedy-FF
  Let C be the number of colors used, and let γ = |V|/C
  Let Q_O be an ordered set of over-full bins in increasing order
      of color index
  Let Q_U be an ordered set of under-full bins in decreasing
      order of color index
  Let L (initially ∅) maintain a list of moves from over-full to
      under-full bins
  for each j ∈ Q_O do
      Let V(j) denote {u ∈ V | color[u] = j}
      Select V'(j) ⊆ V(j) such that |V'(j)| = |V(j)| − γ
      for each k ∈ Q_U AND V'(j) ≠ ∅ do
        Let V'_k(j) ⊆ V'(j) denote vertices that can be moved
            to k such that |V'_k(j)| + |V(k)| ≤ γ
        L ← L ∪ V'_k(j)
        V'(j) ← V'(j) \ V'_k(j)
for V'_k(j) ∈ L do
  for v ∈ V'_k(j) do in parallel
      if (k is a permissible color for v) then
        color[v] ← k
```

This simple approach requires no synchronization on the bin sizes. However it could leave the bins imbalanced. To improve the chance of obtaining a better balance, we fill the under-full bins (set Q_U in Algorithm 4) in the *decreasing* order of color index (we refer to this approach as *Scheduled Reverse*, or more simply, *Sched-Rev*). Attempting to fill the under-full bins in decreasing order increases the likelihood of color co-assignment of vertices—i.e., two vertices being moved from the same source over-full bin are likely to co-locate in the same target under-full bin, thus minimizing the chance of conflicts. This is a consequence of the aforementioned size-related property of the Greedy-FF initial coloring.

4.3 Parallel Recoloring

The parallelization we use for the balancing based on Recoloring is outlined in Algorithm 5. This is similar to the vertex-centric parallel scheme given in Algorithm 2 with the main difference being that we recolor *all* the vertices from scratch and that balance is *imposed* as the recoloring proceeds. In particular, the recoloring approach colors vertices roughly in the order of vertices as they appear from the

1. This step is performed serially in our current implementation since it was very quick for most inputs; however, if required, this step can also be parallelized using parallel prefix (details omitted).

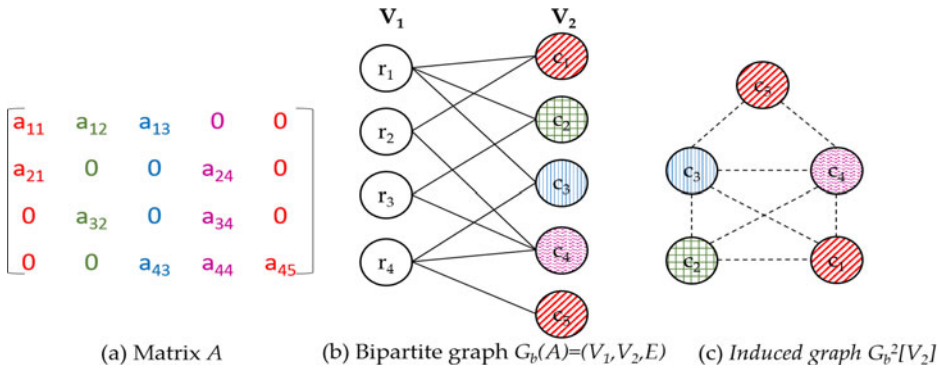


Fig. 2. Illustration of equivalence among structurally orthogonal partition of a matrix A (a), partial distance-2 coloring of the vertices in V_2 in bipartite graph $G_b = (V_1, V_2, E)$ of A (b), and distance-1 coloring of the subgraph of the square graph induced by V_2, that is ${G_b}^2[V_2]$ (c).

largest color to the smallest color of the initial coloring. The rationale for this ordering, as mentioned in Section 3.2, is that the re-coloring procedure would have an opportunity to use fewer colors, since now the vertices that were "difficult" to color initially are processed earlier. In the process of recoloring, we also strive for the size of each color class to be as close as possible to the average size of a color class γ obtained from the initial coloring.

Algorithm 5. Parallel Recoloring for Balance

```
ParallelRecoloring(G = (V, E))
  Obtain an initial coloring of G using Greedy-FF
  Let C be the number of colors used, and let γ = |V|/C
  Let V(j) denote {u ∈ V | color[u] = j}
  Construct an ordered set W = {V(C), V(C − 1), ..., V(1)}
  Initialize bin[i] = 0, for i = 1, ..., C
  U ← W
  while U ≠ ∅ do                          ▷perform a fresh coloring
    for v ∈ U do in parallel
      color[v] ← smallest permissible color k such that
          bin[k] ≤ γ
      Increment bin[k] by 1                          ▷synch. step
    R ← ∅
    for v ∈ U do in parallel
      for w ∈ adj(v) do
        if (color[w] = color[v] and v > w) then
          R ← R ∪ {v}
    U ← R
```

4.4 Complexity

With careful choice of data structures, the sequential Greedy scheme (Algorithm 1) that underlies all of our parallel algorithms, can be implemented such that its runtime is upper-bounded by $O(|V| \cdot \Delta)$, where Δ is the maximum degree in the graph. In each of the templates outlined in Algorithms 2 through 5, the total "additional" work incurred due to parallelization is no more than the work involved in Algorithm 1. Furthermore, the number of rounds required by the iterative variants (Algorithms 2 and 5), as argued earlier, is typically a small constant in practice. Therefore, the net total work in any of our schemes can be upper-bounded by $O(|V| \cdot \Delta)$.

The above complexity represents a worst-case, where for instance, multiple vertices with relatively high degrees tend to occupy the overfull bins in the initial coloring. But such cases also require a relatively high number of such vertices in the input, which is less likely to be observed in real world networks with power-law like degree distributions.

5 Partial Distance-2 Coloring

5.1 Preliminaries

As mentioned in Section 1, besides distance-1 coloring, we considered in the current work a balanced version of another coloring problem, *partial distance-2 coloring*, that is defined on a bipartite graph $G_b = (V_1, V_2, E)$. The coloring "rule" here is to assign colors to vertices in one of the vertex sets, say V_2, such that any pair of vertices v_i, v_j from V_2 that share a common neighbor v_c in V_1 (that is, $(v_c, v_i) \in E$ and $(v_c, v_j) \in E$) get different colors.[2] The objective of the standard version of the problem, as in distance-1 coloring, is to use as few colors as possible. The balanced variant seeks to, in addition, balance the color classes.

The partial distance-2 coloring problem is an important model in computations involving nonsymmetric matrices. For example, a partitioning of the columns of a nonsymmetric matrix A into groups of *structurally orthogonal* columns—a group in which no two columns share nonzero entries in the same row position—can be modeled by a partial distance-2 coloring of the column vertices V_2 of the bipartite graph $G_b(A) = (V_1, V_2, E)$ representing the sparsity structure of the matrix A [26]. Such a model is useful, for example, in an efficient computation of a Jacobian matrix A using automatic differentiation. Fig. 2 shows a small example that illustrates how a structurally orthogonal column partition of a matrix is modeled as as a partial distance-2 coloring of the bipartite graph.

Fig. 2c illustrates yet another equivalence. In general, a partial distance-2 coloring on the vertex set V_2 of a bipartite graph $G_b = (V_1, V_2, E)$ is equivalent to a distance-1 coloring of a certain derived graph—the subgraph of the *square graph* of G_b induced by the vertex set V_2, which we denote by ${G_b}^2[V_2]$. For a general graph $G = (V, E)$, the square graph $G^2 = (V, F)$ is a graph defined on the same vertex set V and where the edge set F consists of pairs of vertices that are distance less than or equal to two edges from each other. In

2. This variant is called "partial" since only one of the two vertex sets in the bipartite graph is colored. It is called "distance-2" since, in the vertex set to be colored, any two vertices that are two edges away from each other are required to get different colors. The naming was first used in [26].

other words, $F = E \cup E'$, where E' corresponds to pairs of vertices that are at distance exactly two edges from each other in G. If the graph under consideration is a bipartite graph $G_b = (V_1, V_2, E)$, then the square graph ${G_b}^2 = (V_1, V_2, F)$, which no longer is bipartite, is such that the edge set can be viewed as having three parts: $F = E \cup E_1 \cup E_2$, where E_1 corresponds to the "new" edges that run among vertices in V_1 (those that are at distance 2 in G_b) and E_2 corresponds to the edges that run among vertices in V_2 (those that are at distance 2 in G_b). The subgraph of ${G_b}^2$ induced by the vertex set V_2 could then be written as $G_c = (V_2, E_2)$. In network science literature, the graph G_c is also sometimes referred to as a "projection" on to the vertex set V_2, and in the numerical optimization community G_c is also known as the "column intersection graph" of the associated non symmetric matrix.

The discussion in the paragraph above focused on the case where the vertex set in the bipartite graph to be colored is V_2. Entirely analogous discussion and definitions apply if the vertex set to be colored were V_1 instead.

Now if partial distance-2 coloring on the vertex set V_2 of the bipartite graph $G_b = (V_1, V_2, E)$ is equivalent to distance-1 coloring of the intersection graph $G_c \equiv {G_b}^2[V_2]$, why don't we then simply construct G_c and solve the distance-1 coloring problem on it using algorithms developed for distance-1 coloring, instead of developing algorithms tailored for partial distance-2 coloring? As has been argued in [26], there are several reasons for this. First, the partial distance-2 coloring formulation offers greater flexibility. In particular, the intersection graph necessarily looses structural information contained in the original bipartite graph. For example, considering the context of an underlying nonsymmetric matrix, given two column vertices in V_2 joined by an edge in G_c, one cannot determine the row at which the two columns share nonzero entries. Second, for some structures, the intersection graph ${G_b}^2[V_2]$ could turn out to be substantially denser (have more edges) than the original graph G_b and hence require more memory. Note that by construction, each vertex $u \in V_1$ of $G(V_1, V_2, E)$ would correspond to a k-clique in $G_b^2[V_2]$, where k is the degree of u. Third, the partial distance-2 coloring formulation avoids the need for building a different data structure than the one used to represent the input graph.

5.2 Algorithms

The greedy scheme for distance-1 coloring outlined in Algorithm 1 can be easily modified to obtain a solution for the partial distance-2 coloring problem. The needed modifications are: (i) the input is a bipartite graph $G_b = (V_1, V_2, E)$, (ii) the for-loop iterates over $v \in V_2$, and (iii) the *neighbors* of a vertex v are those vertices x that are connected to v by a path v, w, x of length two edges. The algorithm modified in this manner is outlined in Algorithm 6. Let $\delta(V_2)$ denote the average degree of a vertex in V_2, and $\Delta(V_1)$ denote the maximum degree of a vertex in V_1. Consequently, the runtime complexity of the algorithm is as follows [26]: $\mathcal{O}(|V_2|\delta(V_2)\Delta(V_1))$, which is same as $\mathcal{O}(|E|\Delta(V_1))$.

We note here that it is possible to develop a faster greedy algorithm by exploiting the bipartite structure of the graph and by relaxing the constraint on the maximum number of colors used. One such approach is to linearly scan the vertices in V_1 and assign different colors to all its neighbors in V_2, while using colors in a greedy first-fit fashion. Such an approach, while improving the runtime ($\mathcal{O}(|E|)$), could potentially use more colors relative to Algorithm 6. Furthermore, parallelization of such an approach may necessitate multiple iterations to resolve potential conflicts introduced during the parallel coloring procedure. For these reasons, we use Algorithm 6 as the basis for the balanced variants developed and results presented in this paper.

Algorithm 6. Greedy Partial Distance-2 Coloring

```
GreedyPD2C(G_b = (V_1, V_2, E))
  for each v ∈ V_2 in some order do
    for each w adjacent to v do
      for each x adjacent w do
        Mark the color of x as forbidden to v
    Assign v a color not marked as forbidden
```

The parallelization strategies discussed in Section 4 focused on distance-1 coloring. The corresponding algorithms for partial distance-2 coloring follow the same methodology, with the modifications outlined earlier in this paragraph. We therefore omit detailed presentation of algorithms for partial distance-2 coloring.

5.3 Another Example of an Application

Parallel implementation of the *coordinate descent algorithm* is another example of an application in which partial distance-2 coloring is useful. Let us review the coordinate descent algorithm briefly to show how partial distance-2 coloring is relevant there. In the coordinate descent algorithm, the rows of matrix A correspond to *samples* and the columns correspond to *features*. The corresponding dimensions are often denoted by n (samples) and k (features). Formulated in a generic fashion, the coordinate descent (CD) algorithm consists of four steps that are performed iteratively until convergence is reached:

Step (1) *Select* a set of coordinates J
Step (2) *Propose* increment δ_j, $j \in J$
Step (3) *Accept* some subset $J' \subseteq J$ of the proposals
Step (4) *Update* weight w_j for all $j \in J'$.

The approach taken in the Select step determines the type of the CD Algorithm. For example in *cyclic* or *stochastic* CD the selection targets a singleton, whereas in *greedy* CD one selects a set—and in the extreme case (fully greedy), the entire set of features is selected. Clearly, greedy is better suited for parallel CD. In a parallel formulation of CD, the Propose and Update steps need to be performed concurrently. In order to perform the Update step with maximal exploitation of parallelism, one needs to identify groups of structurally orthogonal columns (features), since then matrix entries in all the columns in a group can be updated safely concurrently. This identification in turn is what is modeled using partial distance-2 coloring (on the column-vertices) of the bipartite graph representing the sparsity structure of the matrix A.

TABLE 2
Input Statistics for the Graphs used in Our Distance-1 Coloring Study

Input graph	Num. vertices (n)	Num. edges (m)	Degree stats max.	Degree stats avg.
random2	100,000	9,994,356	263	199.89
CNR	325,557	2,738,970	18,236	16.28
coPapersDBLP	540,486	15,245,729	3,299	56.41
rgg11-22	4,194,304	27,306,228	23	13.02
Channel	4,802,000	42,681,372	18	17.77
MG2	11,005,829	674,142,381	5,466	122.50
NLPKKT200	16,240,000	215,992,816	27	26.60
uk-2002	18,520,486	261,787,258	194,955	28.27
Europe-osm	50,912,018	54,054,660	13	2.12

6 Implementation on the Tilera Platform

We have ported our parallel balanced coloring algorithms to the Tilera manycore platform. The Tilera TileGX36 system implements a manycore processor based on a two-dimensional mesh topology. Each core (called a "tile" in Tilera's terminology), consists of a three-way VLIW processing unit, a private 32 KB, two-way set associative L1 data cache, a private 32 KB, direct-mapped instruction cache and a 256 KB, eight-way set associative unified L2 cache. The cache line granularity is 64 bytes across all three caches. Each tile is connected via multiple links to several networks-on-chip (NOCs) in a 2D mesh configuration. (These NOCs include one for coherence traffic, a user-programmable message passing NOC, and a dedicated I/O NOC.)

Tilera's caching policies are the salient features that we exploit to optimize this application. For each individual memory page, the system can set the *home* tile of its data in the cache subsystem. There are two principal modes for setting the home tile of a memory page: *homed* (a particular tile is the home for the whole page) and *hashed* (individual cache lines on the page are distributed in a round-robin manner to the L2 caches of all tile).

For the balanced coloring algorithms and the community detection application we use a heap manager with a backing store of homed huge pages (16 MB/page) for all thread private data. The global shared data structures (Compressed Sparse Row Representation of the graph, arrays of colors and bin sizes) are allocated on default-sized pages (64 KB/page) using the hashed policy. Previous experience with basic coloring and community detection on Tilera [20] has shown this configuration to be the most performant one for all input data sets. The OpenMP threads created by the application are pinned to contiguous sets of cores on the manycore mesh architecture in order to avoid costly thread migration and subsequent cache misses.

7 Experimental Results

7.1 Experimental Setup

Test Platforms. We used two platforms for testing: the manycore Tilera TileGX36 presented in Section 6, and an x86 AMD Interlagos platform.

The TileGX36 platform is equipped with 32 GB of DDR3 memory separated into two 16 GB banks, with the cores running at 1.2 GHz. The TileGX36 runs a custom version of Linux adapted for Tilera's hardware. The compiler and runtime environment are adapted from GCC 4.8.2 and retargeted for the TileGX's 64-bit VLIW cores. The community detection code has been parallelized using OpenMP and Tilera-specific extensions for memory management, synchronization and atomic operations.

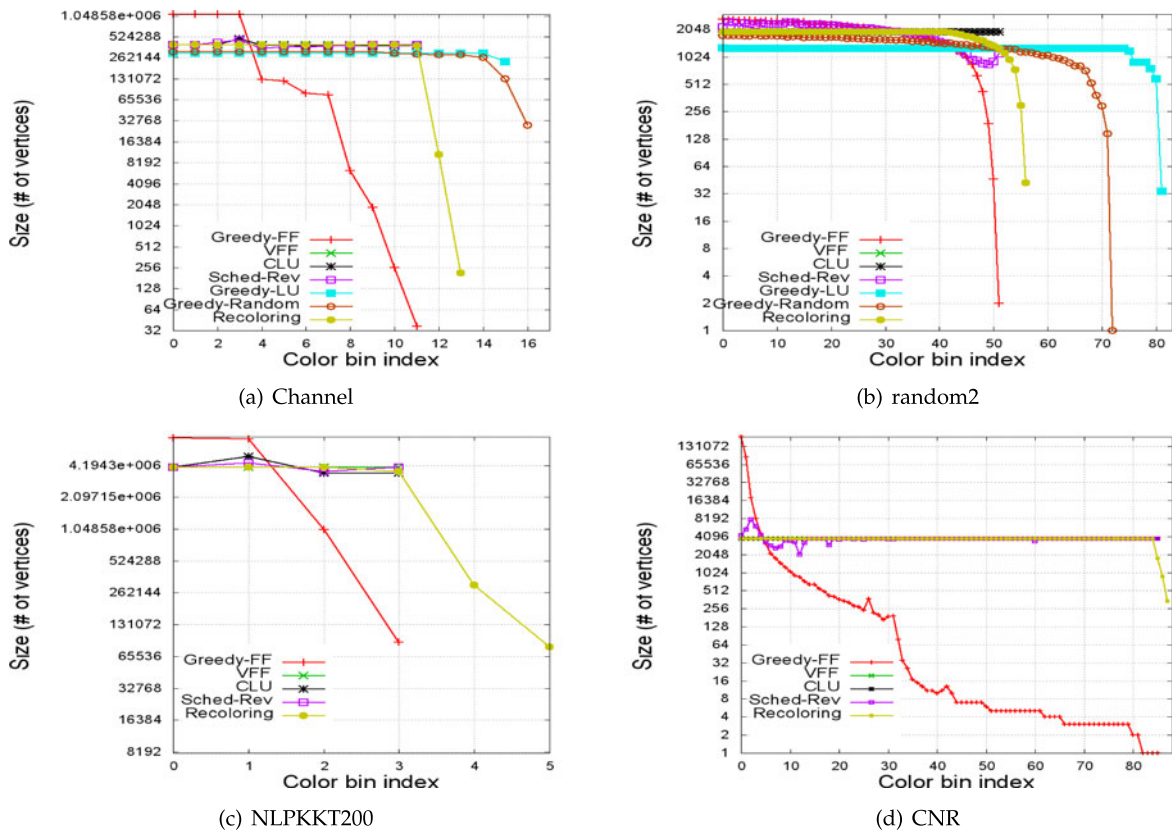
The AMD multi-core platform consists of a dual-socket Interlagos processor with 64 GB of memory. Each socket has 16 cores running at 2.1 GHz. Each pair of cores is grouped into a module sharing a single floating-point functional units and separate integer functional units. Each core contains 16 KB of L1 cache, while each module shares 2 MB of L2 cache. Four modules share 8 MB of L3 cache. Each socket contains eight modules on our system.

Test Inputs. The test inputs used in the different experiments on distance-1 coloring are summarized in Table 2. These inputs were all downloaded from the University of Florida sparse matrix collection [27], with the following exceptions: $MG2$ is a custom-built biological network obtained from protein sequences of a metagenomics data set [28]; $rgg11-22$ represents a random geometric graph [29] that was generated using the generator described in [30]; and $random2$ was generated using a simple random function that applies a probability of edge between any pair of vertices. These inputs were chosen to encompass a variety in graph sizes and color class properties such as the number of colors and color size distribution (Table 3).

TABLE 3
Quality of Balance Obtained by the Different Heuristics on Different Inputs

Input graph	Init. coloring Greedy-FF		Guided schemes VFF	CLU	Sched-Rev	Recoloring		*Ab initio* schemes Greedy-LU		Greedy-Random	
random2	37.17%	(52)	**0.08%**	0.09%	26.21%	22.74%	(57)	9.10%	(82)	21.04%	(73)
CNR	587.73%	(85)	**0.03%**	0.04%	16.44%	13.81%	(88)	12.03%	(211)	24.29%	(209)
coPapersDBLP	342.41%	(336)	0.69%	**0.15%**	12.01%	10.17%	(340)	**0.11%**	(337)	23.15%	(338)
rgg11-22	129.03%	(15)	**0.00%**	**0.00%**	14.69%	26.66%	(15)	4.29%	(17)	38.93%	(16)
Channel	128.99%	(12)	**0.00%**	7.16%	7.55%	35.75%	(14)	4.84%	(16)	20.05%	(17)
MG2	1,272.31%	(2,143)	0.38%	**0.21%**	9.57%	25.34%	(2335)	18.09%	(2169)	94.37%	(2172)
NLPKKT200	99.88%	(4)	**0.00%**	17.65%	7.95%	72.13%	(6)	0.00%	(23)	1.64%	(23)
uk-2002	1,885.15%	(943)	0.08%	**0.01%**	4.88%	3.53%	(945)	2.94%	(1010)	28.34%	(1018)
Europe-osm	126.87%	(5)	**0.00%**	**0.00%**	6.70%	39.90%	(6)	0.00%	(7)	12.07%	(6)

Entries in each cell show the Relative Standard Deviation (in percent) of color class sizes obtained by a given heuristic (the lower the values, the better the balance). The guided schemes VFF and CLU produce the same number of colors as the initial coloring scheme (Greedy-FF). The number of colors produced by Greedy-FF, Recoloring and the two ab initio *schemes is provided in parenthesis (next to their respective RSD values).*



(a) Channel (b) random2

(c) NLPKKT200 (d) CNR

Fig. 3. *Distance-1 coloring:* Distribution of color class sizes produced by the different balanced coloring schemes (horizontal axis corresponds to colors (bins) and vertical axis to sizes of color classes). Recall that smaller color class sizes correspond to reduced parallelism in the end-application, while higher number of colors corresponds to increased number of parallel steps within the application. For Channel and random2, color class sizes from all balancing schemes are shown. For NLPKKT200 and CNR, color class sizes only from the balancing schemes that produce same or comparable number of colors to the Greedy-FF scheme are shown.

All results pertaining to distance-1 coloring are presented in Section 7.2 through Section 7.4 The test inputs and results on partial distance-2 coloring are presented in Section 7.5. The extensions to balanced coloring are presented in Section 7.6.

7.2 Balance Quality Assessment

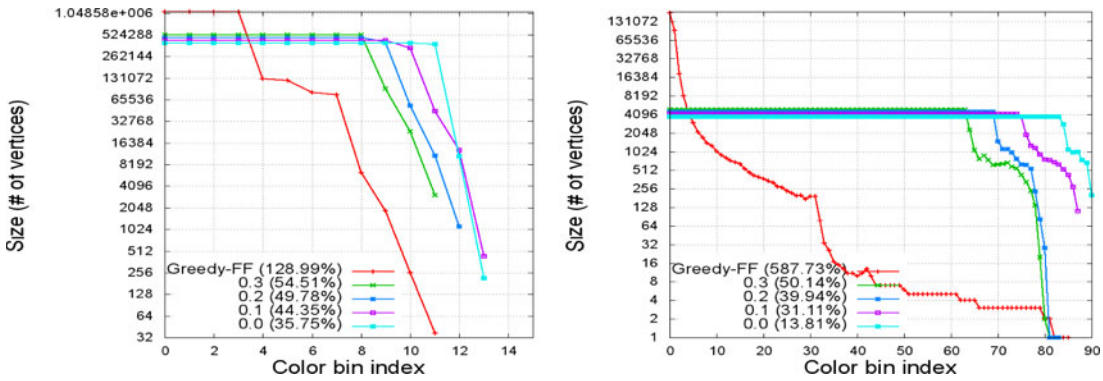
In this section, we compare the quality of balance in the color class sizes produced by the different balancing schemes proposed in the paper. (Please refer to Table 1 for an overview of all the schemes.) To measure balance, we use the Relative Standard Deviation of the color class sizes (expressed in percent), which is the ratio of the standard deviation to the mean color size. The closer this value is to zero the better is the balance. For the schemes {Recoloring, Greedy-LU and Greedy-Random} we also compared the number of colors they produce to the number of colors produced by the Greedy-FF scheme (initial coloring).

Table 3 shows the results of our quality assessment. First, we observe the very large skews in the color sizes produced by the Greedy-FF scheme (which was the primary motivation behind this work). With respect to balancing, we observe that schemes VFF and CLU generally outperform all other schemes in either category (guided or *ab initio*). We note here that if the initial coloring was generated by a scheme other than Greedy-FF, then CLU is expected to outperform VFF. The Sched-Rev scheme was also effective in reducing the skew although the degree of balance achieved was lower than VFF and CLU—as can be expected due to its scheduled strategy. One way to improve the performance of the scheduled strategy is to iterate the procedure a constant number of times; however the tradeoff is that it would increase run-time. In fact, we evaluate this tradeoff in the context of partial distance-2 coloring (see Section 7.5).

Among the schemes that do not guarantee the same number of colors as Greedy-FF (viz. Recoloring, Greedy-LU and Greedy-Random), we observed consistently that all those three schemes produced more colors than the Greedy-FF scheme. However, the number of colors produced by Recoloring was generally close to the number of colors produced by Greedy-FF and other guided schemes (VFF, CLU), and the balancing obtained was comparable to the Sched-Rev scheme. On the other hand, Greedy-LU and Greedy-Random produced significantly higher number of colors making them less desirable from the end-application perspective.

As described in Section 4.3, the main advantage of the Recoloring scheme is that it processes the vertices with larger color indices earlier. Since these vertices have higher degree and consequently harder to color, there is potential benefit in processing them earlier in the Recoloring scheme. However, the balancing constraint imposed during the recoloring process coupled with parallel execution which disturbs the intended order of vertex processing explains the less-than-optimal performance displayed by this scheme.

Fig. 3 illustrates the effect of the different balancing schemes—it shows the sizes of all the color classes produced by the different balancing schemes.



(a) Channel (b) CNR

Fig. 4. *Recoloring:* The figure illustrates the impact of different bounds on bin sizes for the recoloring scheme (Algorithm 5). The default recoloring scheme which uses $\tau = 0.0$ (identifed in the chart by the "0.0" label) fixes the average size for each bin based on the numbers from the initial (unbalanced) coloring scheme. The average size is then varied from 10, 20, and 30 percent (identified by labels "0.1", "0.2" and "0.3" respectively), which results in a smaller number of colors used and possibly a higher imbalance in bin sizes than the default recoloring scheme. The percentages inside paranthesis against each τ setting indicates the imbalance, measured by the Relative Standard Deviation of the resulting color class sizes.

Looking Further into the Recoloring Scheme. For the Recoloring scheme, by allowing some of the color classes to slightly exceed the average color class size (γ), it is possible to achieve a reduction in the number of colors used compared to the baseline version of Recoloring (Algorithm 5). However, the balancing quality may not necessarily be maintained. We empirically investigated this tradeoff between the number of colors and balancing quality. In particular, we modified Algorithm 5 such that the line $bin[k] \leq \gamma$ is replaced by $bin[k] \leq \gamma \cdot (1 + \tau)$, where τ is a small fraction indicating how much offset from γ is "tolerated". Fig. 4 shows results for four values of τ (0, 0.1, 0.2 and 0.3) on two test inputs. The results suggest that the nonzero values for τ help reduce the number of colors used. However, as expected, the balancing quality degrades with increase in τ.

7.3 Performance Evaluation

The balancing schemes were also compared against one another for their parallel performance. We tested both our Tilera and x86 implementations on a range of inputs and thread counts. Tables 4 and 5 show the run-times taken by the VFF balancing scheme. (We select VFF because it was one of the schemes that produced the best balancing results (as was discussed in Section 7.2).) The corresponding speedup charts are shown in Fig. 5.

The results show that the scaling in Tilera manycore is significantly superior to the scaling results in x86. For instance, a top speedup of $13\times$ was observed on 16 Tilera cores. The improved scalability delivered by the Tilera manycore platform can be largely attributed to a scalable on-chip network interconnect, which reduces the costs of synchronization and latency for irregular memory accesses. On the other hand, we found synchronization overhead to be a significant factor impacting the parallel performance on the x86 architecture. We confirmed this by comparing the run-times between the VFF (that uses atomic operations to update bin sizes) and Sched-Rev (that does *not*). On the x86 architecture, we consistently observed Sched-Rev to be $8\times$ or more faster than VFF on all inputs tested (data not shown). The corresponding performance gain on the Tilera platform was a more modest $2\times$ (Table 6).

The speedup trends observed on both architectures also show the impact of the number of initial colors on parallel performance. As shown in Figs. 5a and 5b, the speedups obtained on MG2 (2 K colors) and uk-2002 (943 colors) are superior than on other inputs. Intuitively, fewer colors imply a higher probability for concurrent bin size updates.

From the parallel runtimes shown in Tables 4 and 5, it can be observed that, on a *per-core* basis, the Tilera platform is generally slower than the x86 system. The main reason is the relatively modest frequency and instruction level parallelism (ILP) of the Tilera cores (3 packed operations per VLIW instruction, statically scheduled by the compiler), in comparison to double the frequency on the x86 system and wider superscalar instruction scheduling. However, at full system scale, the runtimes on the Tilera platform begin to become comparable to the runtimes on the x86 platform. This

TABLE 4
Parallel Run-Time (in Seconds) of the VFF Scheme on Different Number of Cores of the Tilera Platform

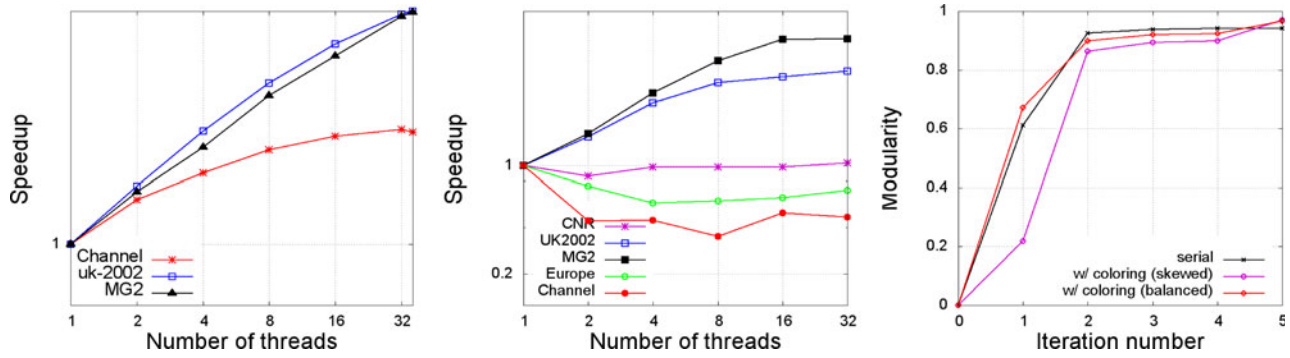
Input graph	Number of threads						
	1	2	4	8	16	32	36
Channel	7.55	4.57	3.37	2.59	2.23	**2.06**	2.13
uk-2002	163.22	84.68	45.59	26.32	16.87	12.11	**11.66**
MG2	460.22	254.66	154.16	85.97	54.56	34.95	**33.29**

Times shown are only for the balancing procedure (i.e., initial coloring time is not *included).*

TABLE 5
Parallel Run-Times (in Seconds) of the VFF Scheme on Different Number of Cores of the AMD x86 Platform

Input graph	Number of threads					
	1	2	4	8	16	32
CNR	0.15	0.17	0.15	0.15	0.15	**0.14**
Channel	**0.79**	1.79	1.77	2.27	1.59	1.70
uk-2002	35.60	23.30	14.03	10.31	9.49	**8.74**
Europe	**11.78**	15.98	20.55	19.90	18.97	16.96
MG2	70.67	44.14	24.00	14.84	10.76	**10.69**

Times shown are only for the balancing procedure (i.e., initial coloring time is not *included).*



(a) Speedup on Tilera (b) Speedup on x86 (c) Community detection

Fig. 5. (a, b) Speedup obtained by our Tilera manycore and x86 multi-core implementations of the VFF balancing scheme. Speedups are relative to one thread executions on both systems. (c) Application study: Evolution of modularity values within the first phase of a parallel community detection implementation (Grappolo) on uk-2002, performed with the use of VFF balanced coloring. The chart also shows the corresponding modularity curves for the runs made *without balanced* coloring and the best performing serial implementation [21].

observation, coupled with the observation that the Tilera platform exhibited a better scalability than x86, (compare Figs. 5a versus 5b), indicates that the Tilera platform, on larger system sizes, has the potential to make up for the difference with respect to x86 and even surpass its absolute performance.

In Table 6, we compare the run-times of three of the most competitive balancing schemes {VFF, Sched-Rev and Recoloring} on the Tilera manycore platform. As expected the Sched-Rev scheme outperforms the other two schemes. More specifically, we observed Sched-Rev to be $\sim 2\times$ faster than VFF.[3] Considering the fact that Sched-Rev also performed appreciably well in terms of balance quality (Section 7.2), we conclude that it provides a reasonable trade-off between quality and performance among the different balancing schemes presented in this paper.

Table 7 compares the runtime performance between a guided scheme (VFF) and an *ab initio* scheme (Greedy-LU). As was shown earlier in Table 3, the VFF and Greedy-LU schemes represent one of the top balancing schemes in their respective categories (guided and *ab initio*). Since the guided schemes require an initial coloring, we include that runtime as well in the Table 7. The results confirm the runtime advantage expected for *ab initio* schemes as they compute a balanced coloring directly, without requiring an initial coloring. However, in most cases, the total runtime for guided schemes (initial coloring + VFF balancing) is comparable to the *ab initio* runtimes, while in general delivering a better balancing quality (Table 3).

7.4 Impact on the Community Detection Application

To evaluate the effectiveness of the proposed balanced coloring schemes in a real world application, we studied the parallel community detection code, Grappolo, described in Section 2.3. Since VFF was one of the leading schemes for balance quality, we used VFF as our default balancing scheme on the Tilera platform. We ran Grappolo in two modes: i) using the original skewed coloring, and ii) using the balanced coloring produced by VFF.

Table 8 shows the results of our evaluation in the context of community detection using Grappolo. In this table, we compare both end-to-end performance (run-time) and output quality (modularity). We note here that the our current implementation of Grappolo is configured to use coloring only during the first phase of its algorithm. However, the algorithm itself is multi-phase and configuring to use coloring in subsequent phases is one of our planned future extensions. However, for this paper, we used coloring only for the first phase, and therefore, the benefits of balanced coloring observed in Table 8 are understated.

From the table, we can observe the following: The overhead introduced in balancing is compensated by the run-time gains achieved in the community detection. This is true for three of the five inputs tested—for instance, in the case of MG2, balancing yields a total end-to-end run-time savings of **44.11** percent. Note that this is for a single execution of the community detection code. In practice, a user may run multiple instances of community detection under different parametric settings (while the coloring is a one-time preprocessing task). The CNR input is the smallest in the number of vertices and edges that we processed and the gains from parallelism (with or without balancing) is insignificant. As for Europe-osm, the first phase only consumed 6 percent of the total run-time and therefore the benefits of balanced coloring are not directly evident from Table 8.

The results in Table 8 also demonstrate the ability of the VFF balanced scheme to preserve quality of output (in terms of modularity). In fact, we observed that introducing balancing has a positive impact on the progression of modularlity in the first phase, as illustrated in Fig. 5c—which is a consequence of the revised ordering of vertices due to balanced coloring.

7.5 Results on Partial Distance-2 Coloring

Our work on partial distance-2 coloring is partly motivated by its application in the parallelization of the stochastic

TABLE 6
Parallel Run-Times (in Seconds) of the Three Balancing Schemes {VFF, Sched-Rev and Recoloring} on 16 Tilera Cores

Input graph	VFF	Sched-Rev	Recoloring
Channel	2.23	**2.19**	3.28
uk-2002	16.87	**8.71**	36.97
MG2	54.56	**27.70**	185.19

3. This performance improvement was even more pronounced in x86 architecture as noted earlier.

TABLE 7
Runtime (in Seconds) Comparison for the Guided VFF Scheme versus *Ab initio* (Greedy-LU) Balancing Scheme

Input graph	Guided		*Ab initio*
	Init. coloring	VFF	(Greedy-LU)
CNR	0.15	0.14	0.19
Channel	1.29	1.70	3.21
uk-2002	5.74	8.74	10.1
Europe	8.69	16.96	21.85
MG2	27.75	10.69	36.88

All runs were performed on 32 threads of the AMD x86 platform.

TABLE 9
Statistics on Structure of the Bipartite Real-World Graphs $G = (V_1, V_2, E)$ Used in Our Study

Input	$\lvert V_1\rvert$	$\lvert V_2\rvert$	$\lvert E\rvert$	$\Delta(V_1)$	$\Delta(V_2)$
KDD-A	8.4×10^6	2.0×10^7	3.0×10^8	85	6.7×10^4
KDD-B	1.9×10^7	2.9×10^7	5.6×10^8	75	3.0×10^6
LPCRE-B	9.6×10^3	7.7×10^4	2.6×10^5	844	14
LPCRE-D	8.0×10^3	7.3×10^4	2.4×10^5	808	13
Gene-Drug	3.0×10^3	1.4×10^4	2.9×10^4	283	144
Host-Pathogen	8.9×10^3	6.3×10^3	2.2×10^4	166	1,631

Recall that Δ corresponds to maximum degree.

coordinate descent (SCD) algorithms [31]. We therefore selected for our experiments two test instances (KDD-A and KDD-B) related to SCD and representing data from the KDD Cup 2010 Challenge on educational data mining [32]. Further, we selected two additional instances (LPCRE-B and LPCRE-D) from the Florida Sparse Matrix Collection [27] that arise from linear programming problems. We consider the bipartite graphs representation of the sparsity structures of these matrices (instances), $G = (V_1, V_2, E)$, where the set V_1 corresponds to the rows and the set V_2 corresponds to the columns, and the edge set E corresponds to the nonzero entries. In partial distance-2 coloring, we color only the vertex set V_2 (please refer Section 5 for definition and other details of partial distance-2 coloring).

As additional test cases from another application domain, we experimented on two bipartite graph inputs obtained from the biology domain—viz. gene-drug interactions [33] and host-pathogen network [34]. Balanced coloring can be used as a way to determine efficient partial orderings of vertices for parallel processing of such networks in co-clustering applications [35]. The testsets used in our experiments are summarized in Table 9.

Table 10 shows the parallel runtimes for two balancing schemes—VFF and Sched-Rev—on the AMD platform. We note that these are bipartite graph implementations of those schemes. For Sched-Rev we provide results from running the algorithm for one round and for three rounds (simply iterated three times). We observed relatively better performance from three rounds than two rounds. The distributions of color class sizes while using the different schemes are provided in Fig. 6. We observe that the quality of output from Sched-Rev with three rounds is comparable to VFF, but the execution time is relatively high. Sched-Rev (with one round) provides faster execution times with comparable quality. The corresponding distributions of color class sizes from the unbalanced coloring scheme (Greedy-FF) are also provided in Fig. 6 (in red). Overall, the results demonstrate the general effectiveness of our balancing schemes for partial distance-2 balanced coloring as well.

7.6 Extensions to Balanced Coloring

In this section, we present two extensions to balanced coloring, motivated by two different practical considerations.

7.6.1 Weighted Balancing

Color class sizes (the number of vertices having the same color) form the basis for the balancing schemes presented thus far. This assumes that the work associated with each vertex is uniform in the end application. However, in some applications, the workload may vary from vertex to vertex. For such scenarios, a weighted treatment of vertices may be more appropriate. Following a similar approach as in [15], we implemented and evaluated a weighted extension for balancing, by setting the weight of a vertex to its degree. We implemented by modifying the VFF balancing scheme as follows. Let $\omega(u)$ denote the weight of vertex u. Then, the target size for each color C is given by

$$\gamma_\omega = \frac{\sum_{u \in C} \omega(u)}{|C|}.$$

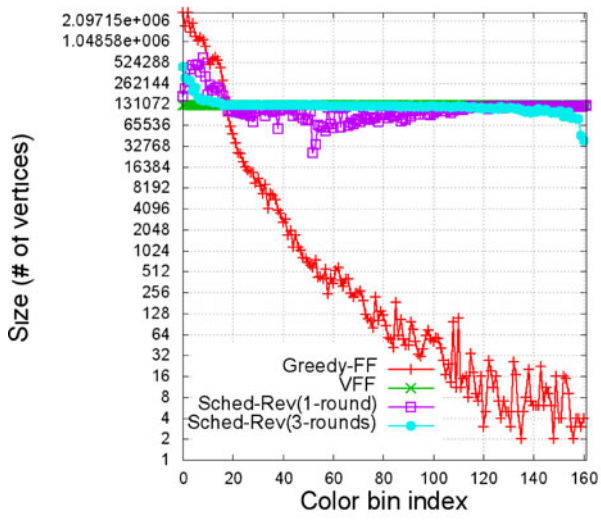
7.6.2 Lower Bound-Based Coloring

One purpose of balancing is to ensure a uniform distribution of parallel workload across color classes in the end application. An alternative, more pragmatic strategy would be to *not* aim for a balancing across all color classes, but rather focus on ensuring that the smaller color classes are filled to a threshold size, sufficient to utilize the threads efficiently. Such a scheme

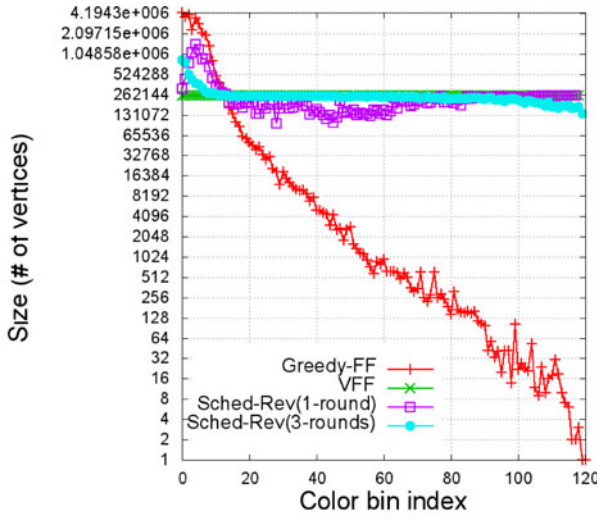
TABLE 8
Evaluation of the Balancing Heuristics on a Parallel Community Detection application, *Grappolo*

Input graph	w/o balanced coloring			w/ balanced coloring			
	Run-time		Modularity	Run-time			Modularity
	Init. coloring	Community detection		Init. coloring	*VFF balancing*	Community detection	
CNR	0.15	3.98	0.9124	0.15	*0.15*	4.16	0.9119
Channel	1.87	38.85	0.9348	1.87	*2.13*	**20.94**	0.9328
MG2	37.31	954.81	0.9984	37.31	*33.29*	**483.80**	0.9984
uk-2002	7.83	406.81	0.9895	7.83	*11.66*	**254.27**	0.9894
Europe-osm	17.95	358.26	0.9988	17.95	*20.98*	369.19	0.9988

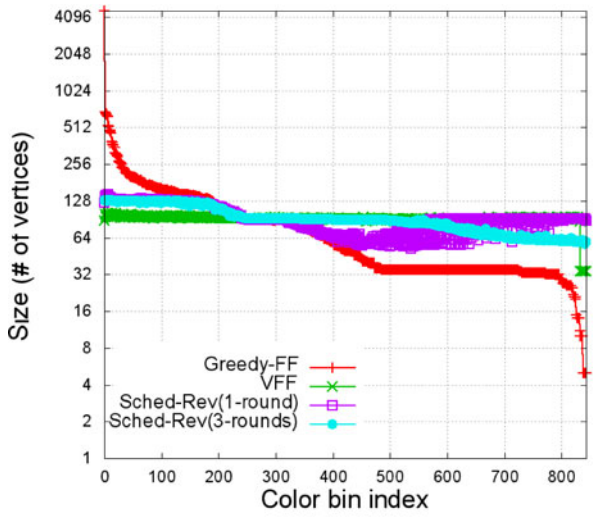
All timing results are in seconds and were obtained on 36 threads of the Tilera manycore platform.



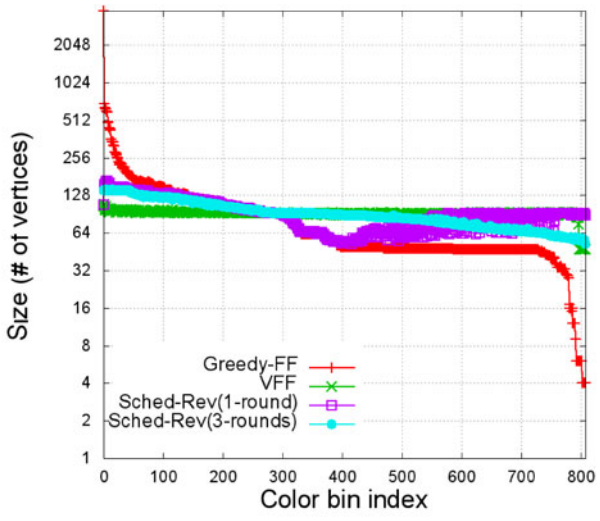
(a) KDD-A



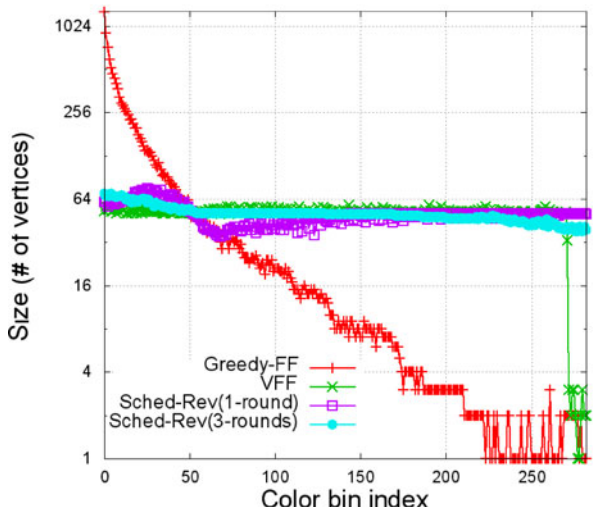
(b) KDD-B



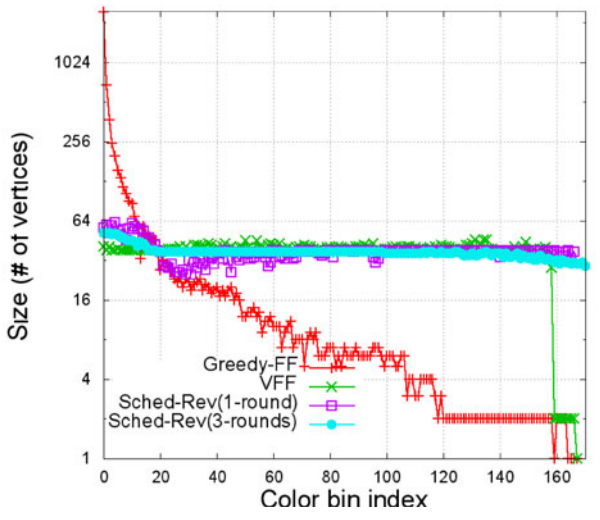
(c) LPCRE-B



(d) LPCRE-D



(e) Gene-Drug



(f) Host-Pathogen

Fig. 6. *Partial distance-2 coloring:* The distributions of color class sizes produced by the different balanced coloring schemes (horizontal axis corresponds to colors (bins) and vertical axis (in $\log_2$ scale) to sizes of color classes).

TABLE 10
Parallel Run-Times (in Seconds) of Unbalanced and Balanced {VFF, Sched-Rev} Partial Distance-2 Schemes on the AMD Platform

Input graph	Init. coloring	VFF balancing	Sched-Rev (1 round)	Sched-Rev (3 rounds)
KDD-A	213	**130**	117	305
KDD-B	606	**354**	154	424
LPCRE-B	0.08	**0.1**	0.12	0.29
LPCRE-D	0.08	**0.09**	0.11	0.24
Gene-Drug	0.008	**0.02**	0.04	0.07
Host-Pathogen	0.005	**0.01**	0.03	0.04

Thirty-two cores were used for the KDD inputs, and 16 cores were used for the remaining inputs (due to smaller size). All runs were performed to color vertices in V_2.

could have an advantage in reducing the balancing cost—because fewer vertex migrations are needed. This motivated our second extension. We modified the VFF scheme to take as input a *lower bound* (threshold) size (denoted by γ_ℓ) for each color class. During the process of balancing, we consider only bins (colors) that have less than the threshold as potential candidates for receiving vertices (from larger bins). Therefore, the resulting color size distribution is expected to be less balanced compared to the other balancing schemes, whereas the overhead for balancing is reduced. We denote this modified scheme as "LB-based" coloring scheme.

7.6.3 Experimental Results

We compared the two extensions outlined above (weighted and LB-based) against the "baseline" schemes of VFF balancing and the Greedy-FF (initial coloring). Fig. 7 and Table 11 show the results of our comparative evaluation.

Fig. 7 shows the balancing quality produced by the extension schemes and the baseline schemes. As can be expected, the weighted scheme does *not* exhibit a good balance based on the number of vertices per color bin. However, if one were to take into account the sum of the weights of the vertices per bin, then the weighted scheme does indeed achieve a balance. The LB-based scheme also shows an expected behavior, where the color classes that contain less than γ_ℓ vertices in the initial coloring are the only classes to receive new vertices into their bins (until the lower bound is met).

TABLE 11
Run-Time Evaluation of Our Extensions (Weighted and LB-based) in Comparison to the VFF Balancing Scheme and the Greedy-FF (Initial Coloring) Scheme

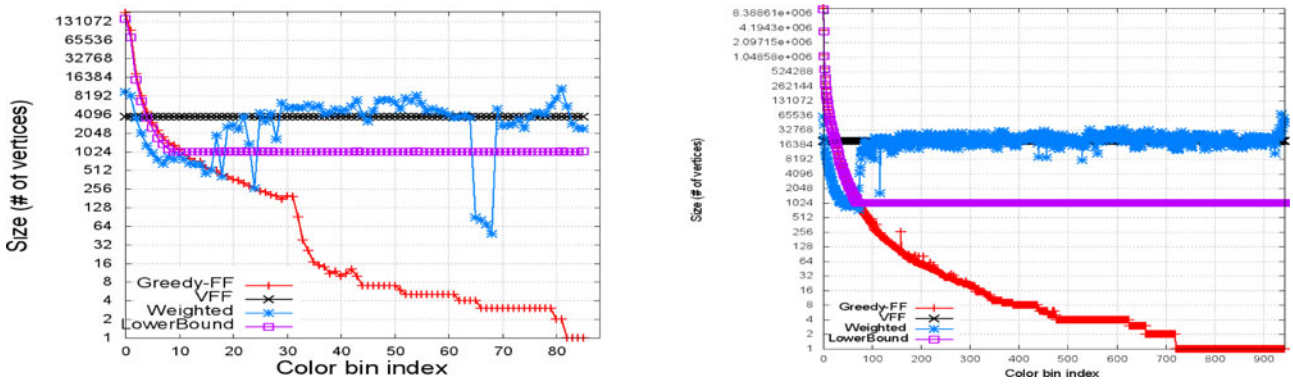
Input	Steps	Greedy-FF	VFF	Weighted	LB-based (γ_ℓ =1024)
CNR	Balancing	0	0.15	0.15	**0.07**
	Comm. Det.	1.30	1.18	1.42	1.25
MG2	Balancing	0	10.33	10.94	**7.86**
	Comm. Det.	204.72	122.67	123.31	149.45
uk-2002	Balancing	0	9.42	9.75	**3.59**
	Comm. Det.	50.91	41.25	44.05	51.08

All executions were performed on 16 threads of our x86 platform.

This implies that the balancing quality achieved by the LB-scheme is sub-optimal but the benefits are expected to lie in its reduced balancing cost (which we evaluate next).

Table 11 shows the run-times for the balancing step and the community detection step (application impact) corresponding to the extension schemes and the baseline schemes. In the case of weighted scheme, the balancing cost is comparable to that of VFF balancing. As for the community detection step, the benefits due to using vertex weights during balancing are *not* readily realized. This is because our parallel implementation of the community detection step (Grappolo) currently uses a vertex-based parallelization instead of an edge-based parallelization. The latter is more conducive to the weighted scheme. As part of our future study, we plan to implement and evaluate such an edge-based scheme in Grappolo, to better exploit the benefits of the weighted scheme.

In the case of LB-based scheme, we observe that the balancing cost is significantly smaller than the VFF balancing cost. This confirms our expectation of reduced work during the balancing procedure through the use of a lower bound size. Note that this may cause sub-optimal balancing (as shown in Fig. 7), which in turn may affect the end application parallel performance. Table 11 shows a marginal increase in the run-times for community detection as per this expectation. In fact, as the lower bound value (γ_ℓ) is further reduced, the size distribution of the color classes would tend closer to the skewed distribution of the initial coloring.



(a) CNR (b) uk2002

Fig. 7. *Balanced coloring extensions:* Distribution of color class sizes produced by the two distance-1 balanced coloring extensions (Weighted, LowerBound) compared to VFF balancing (without weights or lower-bounds) and and initial coloring (Greedy-FF). Horizontal axis corresponds to colors (bins) and vertical axis to sizes of color classes (measured in the *number* of vertices).

8 Conclusions

In this paper, we provided a thorough treatment of the problem of generating balanced graph colorings, with our contributions spanning algorithm development, parallelization, and application. We considered two different coloring variants—distance-1 coloring (the standard coloring problem) and partial distance-2 coloring (defined on a bipartite graph). We presented multiple balancing schemes, developed parallel implementations on conventional multi-cores and an emerging manycore platform (Tilera), and evaluated their effectiveness in achieving a balanced coloring and how such results translate to gains in an application's performance. Coloring is used in a number of parallel computing applications to identify independent tasks, and we expect the detailed study presented in this paper involving a family of balancing heuristics and their implementations on modern day multi-core and manycore architectures, to serve as a valuable reference to application developers who seek to improve parallel performance of their applications using coloring.

Acknowledgments

We thank the anonymous reviewers for the insightful comments, especially on partial distance-2 coloring. This research was support in parts by U.S. Department of Energy grants DE-SC-0006516, DE-AC05-76RL01830, and DE-SC-0010205, the U.S. Department of Defense under the Autotuning for Power, Energy & Resilience (ATPER) project, and by National Science Foundation award IIS-1553528.

References

[1] H. Lu, M. Halappanavar, D. Chavarria-Miranda, A. Gebremedhin, and A. Kalyanaraman, "Balanced coloring for parallel computing applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 7–16.

[2] W. Meyer, "Equitable coloring," *Amer. Math. Monthly*, vol. 80, pp. 920–922, 1973.

[3] A. Hajnal and E. Szemerédi, *Proof of a Conjecture of P. Erdös*. Amsterdam, The Netherlands: North-Holland, 1970, pp. 601–623.

[4] P. Erdös, A. Rényi, and V. Sós, Eds., *Combinatorial Theory and Its Application*. Amsterdam, The Netherlands: North-Holland, 1970.

[5] B. Chen and K. Lih, "Equitable coloring of trees," *J. Combinatorial Theory Series B*, vol. 61, pp. 83–87, 1994.

[6] W. Wang and K. Zhang, "Equitable colorings of line graphs and complete r-partite graphs," *Syst. Sci. Math. Sci.*, vol. 13, no. 2, pp. 190–194, 2000.

[7] H. Yap and Y. Zhang, "Equitable colorings of planar graphs," *J. Combinatorial Math. Combinatorial Comput.*, vol. 27, pp. 97–105, 1998.

[8] H. Furmańczyk, *Equitable Coloring of Graphs*. Providence, RI, USA: Amer. Math. Soc., 2004, pp. 35–53.

[9] M. Kubale, Ed., *Graph Colorings*. Providence, RI, USA: Amer. Math. Soc., 2004.

[10] H. Bodleander and F. Fomin, "Equitable colorings of bounded treewidth graphs," *Theoretical Comput. Sci.*, vol. 349, no. 1, pp. 22–30, 2005.

[11] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition; Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambiridge, U.K.: Cambridge Univ. Press, 1996.

[12] R. Melhem and V. Ramarao, "Multicolor reorderings of sparse matrices resulting from irregular grids," *ACM Trans. Math. Softw.*, vol. 14, pp. 117–138, 1988.

[13] J. Blazewick, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, 2nd ed. Berlin, Germany: Springer, 2001.

[14] A. Tucker, "Perfect graphs and an application to optimizing municipal services," *SIAM Rev.*, vol. 15, pp. 585–590, 1973.

[15] J. Robert, K. Gjertsen, M. T. Jones, and P. Plassmann, "Parallel heuristics for improved, balanced graph colorings," *J. Parallel Distrib. Comput.*, vol. 37, pp. 171–186, 1996.

[16] M. T. Jones and P. Plassmann, "A parallel graph coloring heuristic," *SIAM J. Sci. Comput.*, vol. 14, pp. 654–669, 1993.

[17] J. E. G. Coffman, M. Garey, and D. Johnson, *Approximation Algorithms for Bin Packing: A Survey*. Boston, MA, USA: PWS Publishing Company, 1997, pp. 46–86.

[18] C. Pommerell, M. Annaratone, and W. Fichtner, "A set of new mapping and coloring heuristics for distributed-memory parallel processors," *SIAM J. Sci. Statist. Comput.*, vol. 13, pp. 194–226, 1992.

[19] H. Lu, M. Halappanavar, and A. Kalyanaraman, "Parallel heuristics for scalable community detection," *Parallel Comput.*, vol. 47, pp. 19–37, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819115000472

[20] D. Chavarría-Miranda, M. Halappanavar, and A. Kalyanaraman, "Scaling graph community detection on the Tilera Many-core architecture," in *Proc. IEEE Int. Conf. High Performance Comput.*, Dec. 2014, Art. no. 11.

[21] V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Statist. Mech.: Theory Experiment*, vol. 2008, 2008, Art. no. P10008.

[22] M. E. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, no. 6, pp. 66–133, 2004.

[23] F. Manne and E. Boman, "Balanced greedy colorings of sparse random graphs," in *Proc. Norwegian Informat. Conf.*, 2005, pp. 113–124.

[24] J. Culberson and F. Luo, "Exploring the k-colorable landscape with iterated greedy," in *Cliques Coloring and Satisfiability: Second DIMACS Implementation Challenge*, D. Johnson and M. Trick, Eds. Providence, RI, USA: Amer. Math. Soc., 1996, pp. 245–284.

[25] U. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Comput.*, vol. 38, pp. 576–594, 2012.

[26] A. Gebremedhin, F. Manne, and A. Pothen, "What color is your Jacobian? Graph coloring for computing derivatives," *SIAM Rev.*, vol. 47, no. 4, pp. 629–705, 2005.

[27] T. A. Davis and Y. Hu, "The university of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1–25, Dec. 2011.

[28] J. Daily, A. Kalyanaraman, S. Krishnamoorthy, and A. Vishnu, "A work stealing based approach for enabling scalable optimal sequence homology detection," *J. Parallel Distrib. Comput.*, vol. 79–80, pp. 132–142, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001518

[29] M. Penrose, *Random Geometric Graphs*. London, U.K.: Oxford Univ. Press, 2003.

[30] M. Halappanavar, "Algorithms for vertex-weighted matching in graphs," Ph.D. dissertation, Dept. Comput. Sci., Old Dominion Univ., Norfolk, VA, USA, 2009.

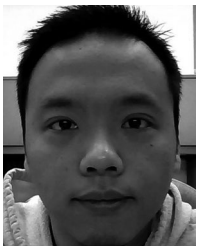
[31] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin, "Feature clustering for accelerating parallel coordinate descent," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 28–36.

[32] H.-F. Yu, et al., "Feature engineering and classifier ensemble for KDD Cup 2010, " in *Proc. JMLR Workshop Conf. Proc.*, 2011, pp. 1–16.

[33] M. Griffith, et al., "DGIdb: Mining the druggable genome," *Nature Methods*, vol. 10, no. 12, pp. 1209–1210, 2013.

[34] M. Wardeh, C. Risley, M. K. McIntyre, C. Setzkorn, and M. Baylis, "Database of host-pathogen and related species interactions, and their global distribution," *Sci. Data*, vol. 2, 2015, Art. no. 150049.

[35] D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer, "Co-clustering of biological networks and gene expression data," *Bioinformatics*, vol. 18, no. suppl 1, pp. S145–S154, 2002.



Hao Lu received the bachelor's degree in computer science from Washington State University, in 2011. Currently, he is working toward the PhD degree in computer science at Washington State University. His research interests include high-performance computing, graph algorithms, and computational biology. The focus of his dissertation is on developing parallel heuristics on community detection in graphs, and heuristics for coloring and balanced coloring. He is a member of the ACM Computer Society.

Mahantesh Halappanavar received the PhD degree in computer science from the Old Dominion University, in 2009. He is a senior research scientist and team lead in the Data Sciences Group, Pacific Northwest National Laboratory. His research focuses on developing efficient parallel graph algorithms and their applications to several domains including the analysis of electric power grids, information analytics, sparse linear algebra, and cyber security. He explores the interplay of algorithm design, architectural features, and data characteristics targeting massively multithreaded architectures and emerging multicore and manycore platforms.

Daniel Chavarría-Miranda received the MS and PhD degrees in computer science from Rice University, in 2004. He is a senior scientist with High-Performance Computing Group, Pacific Northwest National Laboratory. His expertise is in runtime systems, programming models, compilers, and languages for high performance computing. He has been a member of the ACM since 1997.

Assefaw H. Gebremedhin received the BSc degree in electrical engineering from Addis Ababa University, Ethiopia, in 1992, and the MSc and PhD degrees in computer science from the University of Bergen, Norway, in 2003 and 1999, respectively. He is currently an assistant professor in the School of Electrical Engineering and Computer Science, Washington State University (WSU), where he leads the Scalable Algorithms for Data Science Lab. Prior to joining WSU, in fall 2014, he was a research assistant professor with Department of Computer Science, Purdue University. His research interests include high-performance computing, combinatorial scientific computing, network science, data mining and machine learning, bioinformatics, and health analytics. He received the National Science Foundation CAREER Award in 2016.

Ajay Panyala received the BTech degree in computer science from Jawaharlal Nehru Technological University, Hyderabad, India, in 2007, and the PhD degree in computer science from Louisiana State University, in 2014. He is currently a post doctoral research associate with Pacific Northwest National Laboratory. His research interest include compiler optimizations for high performance computing.

Ananth Kalyanaraman received the bachelor's degree from the Visvesvaraya National Institute of Technology, Nagpur, India, in 1998, and the MS and PhD degrees from Iowa State University, Ames, in 2002 and 2006, respectively. Currently, he is an associate professor in the School of Electrical Engineering and Computer Science, Washington State University, Pullman. His research focuses on developing parallel algorithms and software for data-intensive problems originating in the areas of computational biology and graph-theoretic applications. He received the DOE Early Career Award, an Early Career Impact Award and two best paper awards. He serves on editorial boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Parallel and Distributed Computing*. He is a member of the AAAS, the ACM, the IEEE, the ISCB, and the SIAM.