





EXAGRAPH: Graph and combinatorial methods for enabling exascale applications

Seher Acer¹, Ariful Azad², Erik G Boman¹, Aydın Buluç³,
Karen D. Devine¹, SM Ferdous⁴, Nitin Gawande^{5,6} ,
Sayan Ghosh⁶, Mahantesh Halappanavar^{6,7} ,
Ananth Kalyanaraman^{6,7}, Arif Khan⁶, Marco Minutoli⁶,
Alex Pothen⁴, Sivasankaran Rajamanickam¹, Oguz Selvitopi³,
Nathan R Tallent⁶  and Antonino Tumeo⁶

The International Journal of High Performance Computing Applications 2021, Vol. 0(0) 1–19
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211029299
journals.sagepub.com/home/hpc


Abstract

Combinatorial algorithms in general and graph algorithms in particular play a critical enabling role in numerous scientific applications. However, the irregular memory access nature of these algorithms makes them one of the hardest algorithmic kernels to implement on parallel systems. With tens of billions of hardware threads and deep memory hierarchies, the exascale computing systems in particular pose extreme challenges in scaling graph algorithms. The codesign center on combinatorial algorithms, ExaGraph, was established to design and develop methods and techniques for efficient implementation of key combinatorial (graph) algorithms chosen from a diverse set of exascale applications. Algebraic and combinatorial methods have a complementary role in the advancement of computational science and engineering, including playing an enabling role on each other. In this paper, we survey the algorithmic and software development activities performed under the auspices of ExaGraph from both a combinatorial and an algebraic perspective. In particular, we detail our recent efforts in porting the algorithms to manycore accelerator (GPU) architectures. We also provide a brief survey of the applications that have benefited from the scalable implementations of different combinatorial algorithms to enable scientific discovery at scale. We believe that several applications will benefit from the algorithmic and software tools developed by the ExaGraph team.

Keywords

Graph algorithms, combinatorial optimization, algebraic approach, parallel computing

Interplay between algebraic and combinatorial methods in computational science and engineering

Combinatorial algorithms play a crucial enabling role in many applications in computational science and engineering. Work in this area began with the realization that graphs are critical to the design of efficient algorithms for solving sparse systems of linear equations. A graph model of sparse Gaussian elimination was first described by Parter (1961), and chordal graphs were shown to be the adjacency graphs of Cholesky factors of symmetric matrices by Rose (1972). Efficient data structures and algorithms for computing sparse matrix factorizations (Cholesky and LU) were then developed by Rose, Tarjan, Leuker, George, Eisenstat, Duff, Reid, and others. See the discussions in the recent books (Davis, 2006; Duff et al., 2017), and some older and recent

articles (Davis et al., 2016; Liu, 1990; Rose et al., 1976). This led to the development of efficient algorithms and software (SPARSPAK, Harwell Subroutine Library, Yale Sparse Matrix Package), which were followed by more recent codes such as SuperLU (Demmel et al., 1999),

¹Sandia National Laboratories, Albuquerque, NM, USA

²Indiana University, Bloomington, IN, USA

³Lawrence Berkeley National Laboratory, Berkeley, CA, USA

⁴Purdue University, West Lafayette, IN, USA

⁵Intel Corporation, Santa Clara, CA, USA

⁶Pacific Northwest National Laboratory, Richland, WA, USA

⁷Washington State University, Pullman, WA, USA

Corresponding author:

Mahantesh Halappanavar, Pacific Northwest National Laboratory, 902 Battelle Boulevard, P.O. Box 999, MSIN J4-30, Richland, WA 99352, USA.
Email: hala@pnnl.gov

MUMPS, WSMP, sparse matrix functions in Matlab, etc., for solving sparse systems of linear equations using direct or factorization-based methods (Davis et al., 2016).

Since this beginning, the role of graph algorithms in computational science and engineering has burgeoned over time. In addition to sparse direct methods, iterative solvers for linear systems employ graph algorithms to compute preconditioners and coarsen matrices for multigrid methods (Hysom and Pothen, 2001). Graph algorithms are used to partition and map meshes and computational graphs to processors in order to achieve load balance and reduce communication costs in parallel computing Boman et al. (2012b). In Automatic Differentiation, directed acyclic graphs model the decomposition of mathematical functions using unary and binary (elementary) operations, and the use of the chain rule to compute partial derivatives by differentiating these elementary operations; edge and node eliminations on this graph enable algorithms that are efficient with respect to computations and memory needed (Griewank and Walther, 2008). In data analytics, graph construction from noisy, dense data leads to computational models from which algorithms for classification, clustering, regression, etc. may be developed. Sparse subgraphs obtained using criteria such as constraints on degrees, lead to faster algorithms and more accurate classification results. Furthermore, graph algorithms continue to be critical in emerging applications such as quantum computing, epidemic modeling, network science, computer security, computational genomics and proteomics, etc.

To describe the work we have performed in the Exa-Graph project, we will discuss the development and application of several graph algorithms. Some of these applications arise in sparse matrix computations, such as sparse matrix multiplication, solving sparse systems of linear equations, ordering sparse matrices for efficient memory access, etc. In other problems, we will solve problems on graphs using algebraic techniques, as in the computation of the eigenvectors of the Laplacian matrix associated with a graph in order to partition it. Similar in spirit is the use of algebraic operations on suitably defined semirings to solve problems on graphs (Combinatorial BLAS). We will also discuss graph algorithms to obtain significant subgraphs (degree-constrained subgraphs) with applications in sparse matrix computations, data privacy, and machine learning. We will discuss graph algorithms for community detection and influence maximization in networks.

Combinatorial approaches for graph algorithms

Matchings and coverings

We discuss progress we have made in the Exascale project in the design of parallel algorithms for matching and edge

cover problems on graphs, with their applications. These problems enable the computation of degree-constrained subgraphs of a graph that might represent its significant subgraphs. Computing these subgraphs reduce the computational costs and memory required of algorithms that obtain information from the graph, such as semi-supervised classification in machine learning, or the solution of sparse systems of linear equations. The first problem we consider is the one of computing a b -MATCHING in a graph. Given a graph, and a natural number $b(v)$ for each vertex in the graph, a b -MATCHING is a subset of *at most* $b(v)$ edges incident on the vertex v in the graph. Here we assign weights to the edges, and maximize the sum of the weights of edges in a b -MATCHING.

The second problem we consider is b -EDGE COVER, where given natural numbers $b(v)$ for each vertex v , we are required to choose *at least* $b(v)$ edges incident on v to belong to the edge cover. Here, we seek to minimize the sum of weights of the edges in the cover.

Both of these problems have polynomial time algorithms to solve them; however, the asymptotic run time is too high to be practical for graphs with millions or billions of vertices and edges. Hence, we turn to the design of approximation algorithms that have near-linear time complexity in the size of the graph. We also design approximation algorithms that possess high concurrency, so that they can be implemented efficiently on parallel computers.

An *exact* algorithm for an optimization problem computes the optimum value of its objective function. An *approximation algorithm* for an optimization problem computes a value that is within some factor α (a constant or a function of the problem size) of the optimal value for all problem instances. For a maximization problem (as in b -MATCHING), the ratio of the value computed by the approximation algorithm to the maximum value is at least $\alpha < 1$ for all instances; for a minimization problem (as in b -EDGE COVER), the ratio of the value computed by the approximation algorithm to the optimal value is at most $\alpha > 1$, again for all instances. We say that this is an α -approximation algorithm for the problem, and that the approximation ratio of the algorithm is α . Note that this worst-case approximation ratio is obtained analytically by an *a priori* argument, and the approximation ratio for a specific instance might be much better than α . An algorithm for an optimization problem for which we cannot obtain an approximation ratio is called a *heuristic algorithm*. This is the situation for many problems in combinatorial Scientific Computing, and we have to evaluate an algorithm by empirically comparing the value of the objective function it computes with other algorithms on a collection of test problems.

Approximation algorithms have several advantages over exact algorithms, of which the most important for our purposes is the higher concurrency they have relative to exact algorithms. This makes it possible for us to design

parallel algorithms for the matching and edge cover problems, whereas exact algorithms do not have sufficient concurrency. Approximation algorithms are easier to implement when compared to the more sophisticated exact algorithms, which is practically an important reason for their widespread use. We have provided a comprehensive review of the design of approximation algorithms for several matching problems and edge cover problems with applications in [Pothen et al. \(2019\)](#).

A parallel algorithm for b -MATCHING. A b -MATCHING in a graph corresponds to a combinatorial structure called a 2-extendible system. Consider two b -MATCHINGS, $M_1 \subseteq M_2$, and consider the situation when there is an edge $e \in E/M_2$ such that $M_1 + e$ is a b -MATCHING, while $M_2 + e$ is not. If $e = \{u, v\}$, then removing at most *two* edges from M_2 , namely, the two edges incident on the vertices u and v in M_2 , leads to another b -MATCHING. A 2-extendible system is a relaxation of a matroid, in which the independent sets correspond to a 1-extendible system, where in the construction above, only one edge would need to be removed to preserve independence. Mestre showed that for any k -extendible system, the Greedy algorithm computes a $1/k$ -approximation to a maximization problem over its objective function. Hence, we have the result that the Greedy algorithm is a $1/2$ -approximation algorithm for a maximum weight b -MATCHING.

The Greedy algorithm considers edges to add to the matching in non-increasing order of weights, and hence there is not much concurrency in the algorithm. Hence, we have designed another algorithm called b -SUITOR, which is based on making proposals to compute the b -MATCHING. The b -SUITOR algorithm generalizes the Suitor algorithm for maximum edge-weighted matching problem designed by Manne and Halappanavar. Every vertex v makes up to $b(v)$ proposals to its neighbors, in non-increasing order of weights of its edges. Each vertex keeps track of the lowest weight proposal that it has received from a neighbor. The vertex v proposes to a neighbor u , provided the weight of the edge $\{u, v\}$ is higher than the lowest weight proposal that u currently holds. If v can beat this value, then it annuls the lowest weight proposal that u holds, made by some vertex w , and proposes to u . The vertex w will need to make another proposal to a neighbor x such that the weight of the edge (w, x) is higher than the lowest weight offer that x holds, if such a vertex exists. When two vertices u and v propose to each other, then the edge $\{u, v\}$ is added to the b -MATCHING.

The b -SUITOR algorithm computes the same matching as the Greedy algorithm, provided ties in edge weights are resolved in the same manner in both algorithms. But the b -SUITOR algorithm has a great deal of concurrency since every vertex can make proposals to its neighbors. It does so at the cost of annulled proposals which corresponds to wasted work in computing a matching. If the edge weights are chosen uniformly at random, then the expected number of

proposals in the Suitor algorithm is bounded by $n \log n$ for a complete bipartite graph, where n is the number of vertices. Edge weights could be chosen such that $O(m)$ proposals are needed for a sparse graph with m edges, but in practice such pathological distributions are not observed. The b -SUITOR algorithm has been implemented on serial, parallel shared memory, and parallel distributed-memory computers. For parallel b -SUITOR algorithm, the total work (number of operations in the parallel algorithm) is bounded by $O(\beta b(V))$, and the parallel depth (the length of the critical path) is bounded by $O(\log b(V) \log \Delta)$. Here, $b(V) = \sum_{v \in V} b(v)$, $\beta = \max_{v \in V} b(v)$, and Δ is the maximum degree of a vertex. The algorithm has been shown to strongly scale to 12,000 cores of the Cori machine at NERSC on graphs with one or two billion edges [Khan et al. \(2018a\)](#).

Parallel algorithms for vertex-weighted matching. In the vertex-weighted matching problem, there are non-negative weights on the vertices of a graph, the weight of a matching is the sum of weights on the endpoints of the matching edges, and we seek to find a matching of maximum weight. This has not been a well-studied problem in earlier work, although it occurs in applications such as internet advertising, the design of network switches, crew scheduling, etc. Underlying this problem is a matroid called the matching matroid; it consists of a ground set of the set of vertices, with a subset of vertices S defined to be *independent* if there is some matching such that the endpoints of the matching edges contain S . This is unlike the edge-weighted matching problem where the matching edges form a 2-extendible system, but not a matroid. Hence, the Greedy algorithm solves the maximum vertex-weighted matching problem ([Al-Herz and Pothen, 2019](#); [Dobrian et al., 2019](#)). However, the time complexity of this algorithm is $O(nm)$, where n is the number of vertices and m is the number of edges. Furthermore, there is no concurrency in the algorithm since the Greedy algorithm must process the vertices in non-increasing order of weights. Hence, we have turned to the design of approximation algorithms for this problem.

An alternating path with respect to a matching is a path in the graph in which alternate edges in the path belong to the matching. An augmenting path is an alternating path that begins and ends with unmatched vertices; it has one more non-matching edge than matching edges, and hence has an odd number of edges in the path (length). A weight-increasing path is an alternating path of even length that has an equal number of non-matching and matching edges, such that the unmatched endpoint of the path has higher weight than its matched endpoint. Let k be a natural number, and define a k -augmentation as an augmenting path or a weight-increasing path that has at most k edges not in the matching. If we ensure that there is no k -augmentation in the graph with respect to a matching, then the matching is $k/(k+1)$ -approximate ([Al-Herz and Pothen, 2020](#)). Such an

algorithm can be implemented in $O(\Delta^k m)$ time, where Δ is the maximum degree of a vertex. For $k = 1$, we obtain a 1/2-approximation, and for $k = 2$, we obtain a 2/3-approximation. The latter is the first instance of a practical parallel implementation of a matching algorithm with time complexity greater than 1/2. An interesting issue that arises in this work is that care needs to be taken in processing multiple augmentations in parallel to avoid deadlock, livelock, and starvation. Both these algorithms have been implemented in parallel on shared memory computers and exhibit good speedups on modest numbers of threads (Al-Herz and Pothen, 2020).

Parallel algorithms for b -EDGE COVER. Given a graph $G = (V, E)$ and a function $b(v)$ that maps each vertex $v \in V$ to a natural number, a b -EDGE COVER is a subset of edges C such that at least $b(v)$ edges in C are incident on v . We assume that $b(v) \leq \deg(v)$. If $b(v)$ is identically equal to one for all vertices v , then we have the Edge Cover problem. If the edges are weighted by a non-negative function $w(\cdot)$, then the weight of a b -EDGE COVER is the sum of weights of the edges in the cover. The problem we consider is to compute a b -EDGE COVER of minimum weight. An exact algorithm for the problem has polynomial time complexity, since it can be computed as the complement of a maximum weight b' -MATCHING, where $b'(v) = \deg(v) - b(v)$ for all $v \in V$.

We have designed a number of approximation algorithms for this problem, with approximation ratios of 3/2 and 2 (Ferdous et al. 2018; Khan et al. 2018b). The well-known nearest neighbor algorithm is a 2-approximation algorithm for the minimum weight b -EDGE COVER problem, a fact that does not seem to have been recognized in work earlier to ours. The Greedy algorithm for this problem cannot work with static edge weights as in the Matching problem; instead, the algorithm computes effective edge weights, which is the weight of the edge divided by its number of uncovered endpoints. Initially the latter is two, and it becomes one when one endpoint is covered but not the other one; finally when both endpoints are covered, then the weight of the edge becomes infinite, and such an edge will not be included in a b -EDGE COVER, unless it was chosen to cover one of both of its endpoints. We have developed b -EDGE COVER algorithms based on other paradigms for designing algorithms. One of them uses the primal-dual linear programming framework to obtain a 3/2-approximation algorithm, while two others use different reduction to b -MATCHING problems. One of these computes a b -EDGE COVER as the complement of a b' -MATCHING, where $b'(v) = \deg(v) - b(v)$. If we use a 1/2-approximation algorithm that computes a b -MATCHING by matching locally heaviest edges (i.e., edges that have weight greater than equal to other edges that share one its endpoints), such as the b -SUITOR algorithm, then we obtain a 2-approximation algorithm for b -EDGE COVER. Since the b -SUITOR algorithm has been implemented efficiently on serial

and parallel computers, this leads to a scalable parallel algorithm for the b -EDGE COVER problem.

Software libraries. We have included several matching algorithms in the MatchBox software available at www.github.com/CSCsw/MatchBox. The parallel b -SUITOR code is available at www.github.com/ExaGraph/.

Applications. An application for b -MATCHING involves a load-balancing problem for computing the Fock matrix in quantum chemistry, a problem that occurs within the NWChemEx project (Alexander et al., 2020). We have a set of tasks, each of which corresponds to the computation of a set of integrals over quartets of molecular orbitals. A computational cost is associated with each task (Chavarría-Miranda et al., 2015). We are required to map these tasks to a much smaller set of processors in a parallel computer. This corresponds to a b -MATCHING in which each processor is mapped to several tasks, but each task is mapped to a unique processor. The problem is to map the tasks to processors in order to achieve load balance. In current work, we have shown that b -MATCHINGS with submodular objective functions can be used to effectively solve this problem.

We have used the MCE algorithm to solve an adaptive anonymity problem in data privacy using a variational optimization algorithm that at each iteration computes an approximately optimal b -EDGE COVER to compute a significant subgraph. We anonymized a healthcare data set with 750,000 instances and 512 features on 8000 cores of the Cori computer at NERSC in under five minutes (Khan et al., 2018a). Strong scaling results on the Cori machine for three anonymization problems are included in Figure 1. This computation increased the size of the problems solved by three orders of magnitude.

Graph coloring

Graph coloring is a class of graph problems where the vertices need to be “colored,” that is, assigned a label, according to some constraint. In the standard vertex coloring problem (aka distance-1), no pair of adjacent vertices can have the same color. The objective is to minimize the number of colors. This is useful to find independent tasks for parallel processing. A common extension is distance-2 coloring, where vertices less than distance two apart must have different colors. Other variations are also useful in certain applications, such as algorithmic differentiation (Gebremedhin et al., 2005a). Although solving graph coloring optimally is NP-hard, greedy heuristics work well in practice. However, these heuristics are typically highly sequential and difficult to parallelize. Two approaches are commonly used in parallel. The first is based on finding independent sets using a Luby-type algorithm (Jones and Plassmann, 1993). The second is the speculative coloring

method by Gebremedhin and Manne (2000). The idea there is to have many processes (threads) color concurrently in parallel, then check for conflicts and recolor as needed until a valid coloring is found. We found the latter approach more efficient, and in previous work, it was extended to distributed-memory and implemented in the Zoltan package (Bozdağ et al., 2008).

In ExaGraph, we have extended parallel coloring in several ways. First, we optimized the speculative coloring method for manycore architectures such as GPU (Deveci et al., 2016a). We designed a novel edge-based variation of the algorithm that is faster for irregular graphs. We showed the new version was up to 139X faster than cuSparse and also gave fewer colors (4X better geometric mean). As our implementation is based on Kokkos, it is performance portable and runs on a wide range of platforms, including multicore CPU and most GPU platforms.

Second, we are currently developing a hybrid MPI+Kokkos coloring code for exascale. The target is supercomputers with many nodes (distributed-memory), and each node may have multiple GPUs. We require one MPI rank per GPU. We then use the Kokkos Kernel coloring method previously developed on each GPU (local coloring). At the MPI level, we exchange boundary information, detect, and resolve conflicts. We currently support distance-1 and distance-2 coloring (Bogle et al., 2020) while partial distance-2 coloring is in progress. Our approach is highly scalable. We have colored a graph with 76.7 billion edges on 128 GPUs in less than half a second (Bogle et al., 2020). Note that Sallinen et al. (2016) have colored graphs of similar size on a CPU-based parallel computer. Our code is unique in that it runs efficiently on both CPU- and GPU-based computers. Weak scaling is very good (Figure 2).

Generative models. A question that arises in evaluating the quality and performance of parallel coloring algorithms is how to generate graphs at scale with known chromatic numbers. Generally the number of colors computed by a coloring algorithm increases with the size of graphs. We cannot know if this is a property of the graph being colored, or it is because of the performance of the algorithm deteriorating, unless we know the dependence of the chromatic number on graph size. If we can generate graphs with known chromatic numbers at scale, we can use them to test the performance of scalable coloring algorithms. In this section, we present our work on generative models that are broadly applicable to ECP applications and beyond.

In recent work by Cheng et al. (2020b), we have identified several classes of graphs that can be generated to have arbitrary sizes, and whose distance-1 chromatic numbers (or good lower bounds) are known. We consider three classes of graphs for this problem. The first is the well-known Erdős–Rényi graph in the probability regime where the expected mean degree is specified. Here, the chromatic

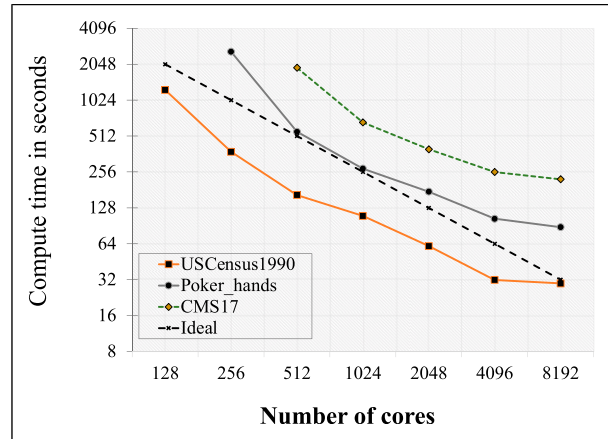


Figure 1. Strong scaling on Cori for three adaptive anonymity problems.

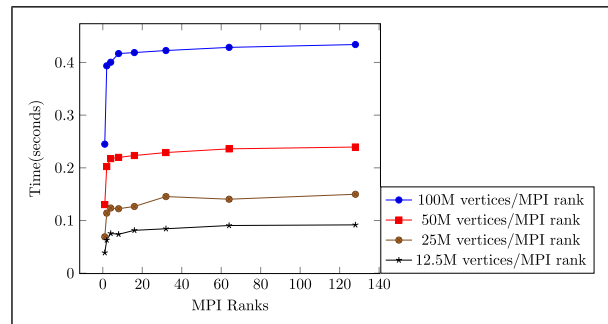


Figure 2. Weak scaling for coloring a mesh on up to 128 GPUs (Bogle et al. 2020).

number of the graph is a constant that depends on the mean expected degree. Then, for large enough values of the expected degree and the number of vertices, with high probability, the chromatic number is a constant depending on the mean expected degree but not on the size of the graph. The second is a random geometric graph embedded in hyperbolic space where the size of a maximum clique is a tight lower bound on the chromatic number, and the asymptotic behavior of this value is known and we can compute a good estimate of it. The third class is the Mycielski graph, which is recursively constructed to have a chromatic number that increases logarithmically with graph size although the maximum clique size remains two. We have computationally verified that these graphs can be colored in nearly optimal number of colors by the Greedy coloring algorithm.

We have extended this work to the distance-2 coloring problem on random bipartite graphs, a problem that corresponds to estimating Jacobian matrices using either the set of rows or the set of columns, but not both. Here, only one vertex set in the bipartite graph is colored. We obtain a lower bound of $\ln n / \ln \ln n$, where n is the number of vertices in

the vertex set being colored, on the partial distance-2 chromatic number of random bipartite graphs by using intersection graphs and “a balls and bins” analysis. Note that these results show that the chromatic number increases with graph size, unlike the Erdős-Rényi graphs where the distance-1 chromatic number depends on the average expected degree but not the graph size. We also obtain upper bounds close to the lower bound in parameter regimes of interest to us, and report empirical results that show that the lower bound is reasonably tight.

Earlier theoretical analyses show that the Greedy algorithm for coloring (or some variant) colors Erdős-Rényi graphs using a number of colors that is at most twice the chromatic number of the graph. Our computational results confirm these analyses. ColPack, a software library that we have developed for solving several coloring problems and associated estimation problems in computing Jacobians and Hessians over a period of years, has been extended with shared memory implementations in recent work by Cheng et al. (2020a).

Software libraries. Zoltan2 (Boman et al. 2012a) is a toolkit of parallel algorithms for partitioning, coloring, ordering, and task placement on modern computing architectures. Zoltan2 is in part funded by ExaGraph. Like its predecessor, the widely used Zoltan toolkit G (Boman et al., 2012a; Boman et al. (2007), Zoltan2 focuses on the graph-algorithm needs of large-scale, distributed-memory applications. However, Zoltan2 advances Zoltan’s capabilities in three key ways.

First, Zoltan2 supports modern computer architectures such as pre-exascale systems with multicore and accelerator-based nodes. Zoltan2 includes hybrid MPI+X algorithms for partitioning (e.g., Acer et al., 2020; Deveci et al., 2016b) and coloring (e.g., Bogle et al., 2020) on these multicore and GPU platforms. Performance portability is achieved through use of the Kokkos library (Edwards et al. 2014), which provides a single interface for on-node parallel operations with specialized back-ends for each architecture. Zoltan2 also provides task placement algorithms that assign MPI ranks to compute cores in a way that reduces application communication cost and network congestion in extreme-scale simulations (Deveci et al., 2019). All of these capabilities are crucial for support applications on next-generation architectures.

Second, Zoltan2 has a more natural user interface for application developers, separating the input adapters from the model (e.g., graph, hypergraph) being used.

Third, Zoltan2 is better integrated with the Trilinos solver framework (Heroux et al., 2005; Trilinos Project Team). However, the design allows Zoltan2 also to be used by non-Trilinos users.

Kokkos Kernels is a library for sparse or dense linear algebra kernels, and graph kernels that are performance

portable to CPUs, KNLs, and GPUs. Kokkos Kernels is part of the Kokkos ecosystem, and it depends on Kokkos Core for performance portability. Kokkos Kernels by design has sparse linear algebra kernels that are tuned for both scientific applications and data science applications. This allows the library to serve as the foundation for both combinatorial approaches and algebraic approaches for graph algorithms.

Kokkos Kernels implements combinatorial algorithms for distance-1 (Deveci et al., 2016a) and distance-2 coloring, reverse Cuthill-McKee (RCM) ordering methods, and graph clustering for better preconditioning. We also have an algebraic approach for triangle counting (Wolf et al., 2017) based on fast SpGEMM implementations (Deveci et al., 2018, 2017). Kokkos Kernels algorithms are used within the Trilinos software framework to improve the performance of multigrid methods, preconditioners such as Gauss-Seidel or ILU(k) preconditioner. Kokkos Kernels is funded by multiple projects, and ExaGraph uses it as a delivery vehicle for on-node (shared memory) graph algorithms.

A common theme of all these algorithms in Kokkos Kernels is effective use of large-scale parallelism, by using algorithms that are edge-based or nonzero-based, and algorithms that can exploit hierarchical parallelism. Kokkos allows expressing such hierarchical-parallel and maps the hierarchical parallelism to multiple GPUs, architectures like KNLs or traditional CPUs. This allows Kokkos Kernels-based graph algorithms to be performance portable.

Applications. Graph coloring is a graph kernel that has several applications. Within ECP, there are at least two application customers. First, the MueLu multigrid solver in Trilinos is part of the National Security Applications (ATDM) (Alexander et al., 2020). The matrix coarsening step is often given as $A_C = RAP$ (where R is a restriction operator and P a prolongator) but can also be viewed as a graph algorithm as A , R , and P are sparse. MueLu uses smoothed aggregation for coarsening. First, a set of *seed vertices* are chosen, and then the aggregates formed around the seeds. Typically, the seed vertices form an independent set or a distance-2 independent set. We can solve these problems using graph coloring, as each color class is an independent set.

Second, the National Security Application SPARC uses automatic differentiation to get the Jacobian matrix. The Jacobian matrix is sparse and can be formed more efficiently by compressing the columns. How to find sets of columns that are structurally orthogonal is a well-known coloring problem. It has been shown that partial distance-2 coloring is an efficient method for this problem (Gebremedhin et al., 2005b). As the Jacobian is typically large and distributed across many processes, a distributed-memory algorithm/software is needed.

Coloring has also been applied as an effective heuristic for parallelization in other graph algorithms developed in ExaGraph such as community detection (Lu et al., 2017).

Community detection

Given a graph $G = (V, E, \omega)$, where ω are nonzero positive weights associated with the edges, the goal of *community detection* is to identify tightly knit groups (or clusters) of vertices that strongly correlate to one another within their own group, but sparsely so with the rest of the graph. The problem is not rigorously defined and is known to be NP-hard (Fortunato, 2010). Consequently, a diverse set of methods have been devised for identifying communities in a graph. The emergence of large-scale graphs from domains such as bioinformatics and social network science has accelerated the need for fast and accurate algorithms for community detection.

Modularity was proposed by Newman as a statistical measure defined for a given partitioning of an input graph that measures the connectivity with the partitions (clusters) with respect to a probabilistic estimate on the same fraction in a random graph with identical degree sequence (Newman and Girvan, 2004). Modularity optimization is an NP-Complete problem. However, heuristics such as the Louvain method proposed by Blondel *et al.* has been demonstrated to compute high quality solutions efficiently (Blondel et al., 2008). While efficient, problems such as non determinism and resolution limit are well-known limitations of this approach.

Our work builds on the Louvain method as the serial template. We employ a number of heuristics such as graph coloring, minimum label heuristic, threshold scaling, and early termination to efficiently parallelize the inherently sequential Louvain method. By applying these heuristics, we have not only demonstrated excellent scaling results, but also for the quality of solutions that are competitive with the serial Louvain method (Ghosh et al., 2018a). Figure 3 shows the performance of VITE on ALCF Theta supercomputer using four real-world graphs of different structure. We have also developed a preliminary version of VITE that can make use of multi-GPU systems, namely, CUVITE, which uses the CUDA API to offload the most compute intensive parts on the GPUs. Figure 4 shows the performance of CUVITE on multiple Nvidia V100 GPUs. Here, the performance of the distributed-memory CPU code (VITE) and CUVITE are compared for two datasets (*nlpkkt240* with 387M edges and *rgg134* with 794M edges) on 2–8 nodes of the OLCF Summit supercomputer. Both the implementations use 6 MPI processes per node, and 14 OpenMP threads (7 cores \times 2-threads per core) per process, with the GPU version using a GPU per process in addition to the OpenMP threads. The performance of CUVITE was found to be about 1.3–2.3 \times of VITE for the specific input datasets using up to 8 nodes of OLCF Summit (48 GPUs) relative to two nodes. On a single GPU, CUVITE obtained a performance improvement of up to 19 \times relative to NVIDIA RAPIDS CUGRAPH (NVIDIA, 2020) and greater than 10 \times relative to multithreaded CPU implementation GRAPPOLO (Halappanavar et al., 2017).

A faster GPU implementation of Louvain method may be possible for certain datasets that exhibit low communication overhead and load imbalance over the partitioned distributed graph. Currently, the performance of CUVITE is limited by the frequent data movement and transformations between host and device, adding significant overhead for a number of cases. Future research will look into uniform data representations across heterogeneous systems to minimize data transfers across memory domains.

Software libraries. GRAPPOLO is a C++ library targeting multicore systems and offers the most features for community detection and is implemented using OpenMP shared-memory programming model. A distributed implementation using MPI+OpenMP is available using a library called VITE. A simpler variant of VITE is available as an ECP proxy application called MINIVITE. We have also developed a single GPU version using OpenMP+CUDA in a tool named Rundemaanan, and distributed multi-GPU implementation using MPI+OpenMP+CUDA is currently under active development via a library named CUVITE. We are also currently developing distributed multi-GPU versions

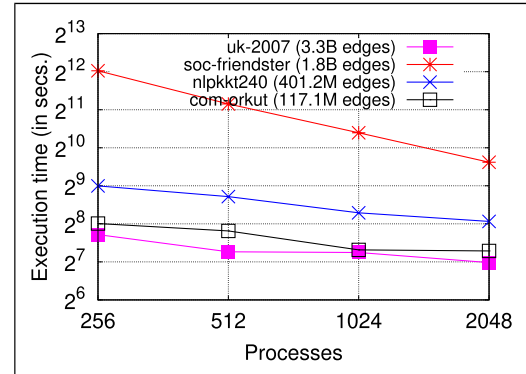


Figure 3. Strong scaling results for VITE using four real-world graphs on ALCF Theta supercomputer. VITE is a distributed community detection library using MPI+OpenMP.

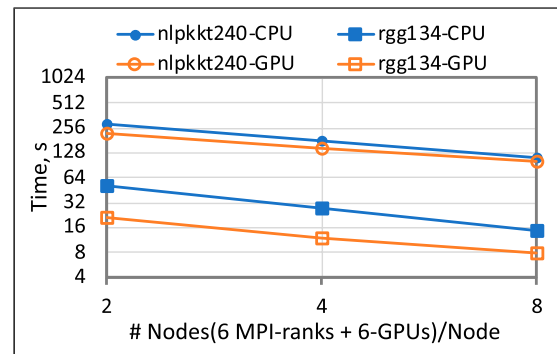


Figure 4. Strong scaling results for GPU code CUVITE and comparison with VITE on OLCF Summit supercomputer.

using OpenMP offloading features. Some of these efforts have won IEEE HPEC Graph Challenge Awards (Ghosh et al., 2018a, 2019; Halappanavar et al., 2017).

Applications. Community detection is one of the most widely used graph algorithms with applications in diverse domains. One particular application we highlight here is in bioinformatics. With the advent of high-throughput devices, single-cell experiments are capable of producing datasets with millions of cells. Community detection is often employed to identify cell populations in single-cell analysis. However, for large datasets the computation time can be prohibitive, and therefore, development of scalable algorithms becomes necessary. Existing approaches such as subsampling of cells or sparsification of data greatly reduces the accuracy, and are therefore undesirable. Consequently, efficient parallel algorithms can play a significant role in enabling single-cell analysis. The ExaGraph team integrated their scalable community detection method with a newly developed tool named FastPG (Bodenheimer et al., 2020). FastPG builds on the state-of-the-art-method graph-based algorithm PhenoGraph by parallelizing key steps in the workflow, where the final step is clustering using GRAPPOLO. Using a set of standard datasets with known ground truth, the team demonstrates that FastPG has the same cell assignment accuracy but is on average $27\times$ faster than other tools. FastPG also has higher cell assignment accuracy than two other fast clustering methods, FlowSOM and PARC (Bodenheimer et al., 2020).

The ExaGraph team has also worked on the application of community detection for model reduction in the context of electric power grid domain (Purvine et al., 2017), and more recently for the purpose of fast and efficient ordering of vertices in graphs for efficient execution on hierarchical memory systems (Barik et al., 2020).

Benchmarking efforts. An indispensable tool in the codesign and benchmarking of novel supercomputing systems is mini/proxy applications from larger scientific codes. Proxy applications represent performance-critical kernels that capture both programming models and computational characteristics of their original application, enabling hardware and system software designers to optimize their layers of the stack without requiring to simulate or emulate an entire application. Most existing proxy apps are extracted from older scientific codes, and include kernels that operate on dense data structures or exhibit structured communication patterns. Modern applications however are varied, operating on an increasingly large and unstructured data coming from newer instruments and analytics kernels. Mixed workloads that combine phases of structured simulations with data analytics and machine learning are increasingly common and challenging to support. Consequently, there is an urgent need to extend the set of proxy apps to include

kernels that use sparse data structures and graph algorithms as key computational elements. One such example is the MINIVITE proxy application developed by the ExaGraph team Ghosh et al. (2018b), which is part of the ECP Proxy Applications suite (Richards et al., 2018). Several vendors have been using MINIVITE as one of the kernels to evaluate their novel architecture designs and optimization. Tools developed by the team are providing a valuable service in the codesign of computing, networking, runtime, and programming models.

Influence maximization

Given a graph $G = (V, E, \omega)$, a diffusion process, and a budget k , the *influence maximization* problem is to select a set of k seed vertices that will result in maximizing the expected outcome of the diffusion process over the graph when chosen as the seed set. The problem was first proposed by Domingos and Richardson (2001) in the context of viral marketing campaigns. However, it has broad applicability in (social) network analysis, controlling spread of epidemics, and optimal sensor placement. Optimal algorithms for IM are known to be NP-hard. However, exploiting a certain combinatorial structure known as *submodularity*, greedy algorithms with approximation guarantees exist. In particular, the greedy algorithms provide an approximation guarantee of $(1 - 1/e - \epsilon)$, where $e = 2.72$ and $0 \leq \epsilon \leq 0.63$ is an input parameter. First, such algorithm was proposed by Kempe et al. (2003), which was subsequently improved by several authors, with the most efficient algorithm in recent times attributed to Tang et al. (2015).

While newer approaches have enabled many applications, the prohibitive computational costs have prevented a broader adoption of IM algorithms. The ExaGraph team worked on efficiently parallelizing the state-of-the-art algorithms for influence maximization on a variety of distributed computing platforms, including accelerator-enabled leadershipclass platforms. From an algorithmic perspective, the best known algorithms are based on the concept of reverse reachability in graph. Intuitively, instead of asking “Who am I influencing?” one asks “Who is influencing me?” from the perspective of a vertex in the graph. Using statistical analysis, these methods bound the amount of computation that needs to be performed, and are therefore efficient. Parallelizing these algorithms involves two basic phases—an embarrassingly parallel step of constructing the samples, followed by a selection step that involves serialization and synchronization among the participating processes. Since the reverse reachable approaches limit the diffusion models that can be used, there is still a need for computationally expensive methods such as the greedy hill climbing algorithm of Kempe et al. (2003). Although relatively expensive, the greedy hill climbing algorithm is relatively amenable to parallelization and can

be generalized to include different diffusion models (Minutoli et al., 2020b).

Software. Through careful design and optimizations, the team has developed a distributed software library named RIPPLES using MPI+OpenMP as the programming model. A specialized library targeting distributed multi-GPU platforms using MPI+OpenMP+CUDA is also available through CURIPPLES (Minutoli et al., 2020a). The tools not only enable computation of solutions on large-scale problems but also provide the ability to compute high quality solutions. Note that the computational complexity grows nonlinearly with the quality of solution. The team has demonstrated up to $790\times$ speedups (Figure 5) relative to the previous state-of-the-art, and with unprecedented accuracy for large-scale problems.

Applications. In the context of an epidemic outbreak, intervention techniques such as vaccination and social distancing become critical in containing its spread. Graph-theoretic (or network) models have been explored extensively in literature (Marathe and Vullikanti, 2013). The team’s recent work on applying RIPPLES to the problem of optimal intervention (vaccination) to reduce the spread of an epidemic outbreak has demonstrated significant impact (Figure 6), where previously known state-of-the-art methods such as SAAROUND (Sambaturu et al., 2020) are computationally infeasible. While the previous methods are based on mathematical programming, RIPPLES is able to exploit the underlying structure through submodular optimization and scalable implementations (Minutoli et al., 2020b). The team has also worked with subject matter experts to demonstrate the applicability in domains such as

cancer research and soil microbiome research (Minutoli et al., 2019). Many more applications including observability and controllability of complex systems will target for future exploration.

Algebraic approaches for graph algorithms and combinatorial problems

Basic linear algebraic operations on sparse matrices and vectors can be used to efficiently implement several data analysis pipelines, with the canonical example being graph algorithms as codified by the GraphBLAS effort (Buluç et al., 2017; Kepner et al., 2016). Recently, sparse matrix operations on user-defined semirings are used to accelerate certain machine learning and biology problems. Here, we describe the advances enabled through ExaGraph, either via direct funding or through collaborations or providing computational resources.

Bipartite matching for sparse solvers

A matching in a graph is an independent set of edges. A bipartite graph is a special type of graph where the vertices can be split into two sets such that no edges connect vertices in the same set. Finding matchings in bipartite graphs has a celebrated history. In the context of scientific computing, one well-known application is in direct solvers for square non-singular sparse matrices. Due to the high-cost of exchanging rows (also known as *pivoting*) dynamically in distributed-memory platforms, these solvers (Li and Demmel, 2003; Rouet et al., 2016) relied on prepermuting the sparse matrix prior to factorization. In this method, known as *static pivoting* (Li and Demmel, 1998), the goal of

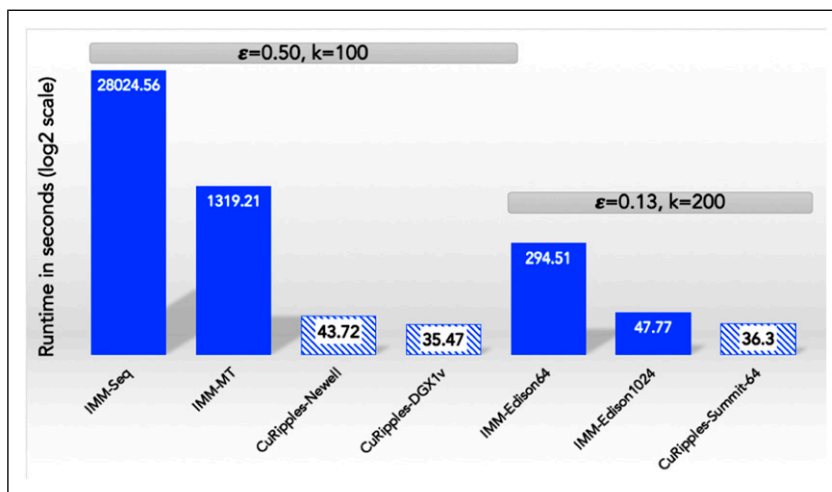


Figure 5. Scaling results for CURIPPLES. We achieve a speedup of up to $790\times$ over a state-of-the-art serial implementation (left), while also significantly improving the approximation factor (to $\epsilon = 0.13$) and doubling the number of seeds (right). The input network is com-Orkut. Details are available in Minutoli et al. (2020a).

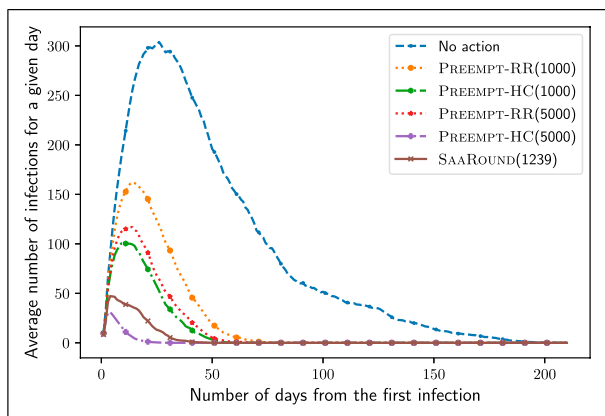


Figure 6. Temporal evolution of infections under different intervention strategies. Algorithms considered are: PREEMPT-RR (reverse reachable), PREEMPT-HC (greedy hill climbing), and SAAROUND (Sambaturu et al., 2020) for contact network Portland-141k. Details are available in Minutoli et al. (2020b).

the permutation is to make sure that the matrix elements in the diagonal are all nonzero and their magnitudes are as large as possible to ensure numerical stability.

This problem is conveniently modeled as a matching problem on a bipartite graph. Each existing nonzero in the matrix is an edge and the bipartite vertex sets are composed of rows and columns of the matrix. The requirement of finding a diagonal that is free of nonzeros means that the matching needs to be perfect. The magnitudes of the diagonal entries being as large as possible is a softer requirement that can be relaxed. Previous attempts on parallelizing the maximum-weight perfect matching algorithms have not been successful at producing a highly-scalable solution. In our recent work, we relaxed the maximum-weight requirement and opted for a heuristic that achieves a heavy-weight perfect matching (HWPM) instead (Azad et al., 2020).

Our algorithm starts with a maximum cardinality matching, which is by definition perfect due to the matrix being nonsingular. The input matrix is assumed to be nonsingular because otherwise a Gaussian elimination-based solver would fail. This maximum cardinality matching, also called a maximum transversal in the literature (Duff and Koster, 1999), is computed in distributed-memory using Combinatorial BLAS (CombBLAS) primitives (Azad and Buluç, 2016), chief among them sparse matrix-sparse vector multiplication (SpMSpV). While computing the maximum cardinality matching, if there are multiple paths that can be augmented at the same time, we choose the one with the larger weight. This modification is almost trivial and amounts to modifying the semiring addition function due to the flexibility of CombBLAS. Our algorithm then proceeds to find weight-increasing alternating cycles. We chose to only find cycles of length 4 for

limiting the critical path of our computation and increasing scalability. The resulting matchings are often already maximum weight or extremely close to maximum weight. Our resulting HWPM algorithm scales up to 17,408 cores.

Protein similarity network construction

Another important application that ExaGraph helped accelerate is the protein family identification. Proteins can be divided into families based on their evolutionary relationship. Proteins in the same family share a relatively recent common ancestor. Members of a given protein family are said to be homologous and often perform similar functions. In the absence of high-quality data that reveals protein structure and function, the method of choice is to rely on sequence information for identifying protein families. The common pipeline (Enright et al., 2002) constructs a similarity network among proteins using their pairwise sequence similarity and then clusters that network using a graph clustering algorithm such as the Markov Cluster Algorithm (MCL) (Van Dongen, 2000).

The first step of similarity network construction has traditionally been done with BLAST (Altschul et al., 1990) but the increases in protein sequence data now requires much faster tools. This has led to several popular alternatives such as LAST, DIAMOND (Buchfink et al., 2015), and MMseqs2 (Steinegger and Söding, 2017), which are widely used in practice. However, LAST and DIAMOND only run on single CPU node, whereas MMseqs2 does not scale well to high node counts. To unleash the power of Exascale supercomputers on billions of protein sequences, we recently developed PASTIS (Selvitopi et al., 2020a), a distributed many-to-many protein sequence aligner that relies on sparse matrices. PASTIS uses the 2D sparse matrix-matrix multiplication (SpGEMM) algorithm Buluç and Gilbert (2012) implemented in CombBLAS to quickly identify pairs of protein sequence that either share an exact or “similar” short subsequence. It performs expensive pairwise alignment only on those pairs that share exact or similar subsequences. PASTIS, an ECP application jointly developed by ExaGraph and ExaBiome, scales up to 2000 nodes on NERSC’s Cori supercomputer while being comparable in accuracy with other sequence aligners.

The output of PASTIS is fed to a high-performance distributed implementation of the Markov Cluster algorithm named HipMCL (Azad et al., 2018), an application initially developed by the ExaBiome project. ExaGraph identified and optimized key graph and sparse matrix kernels within HipMCL, which we describe below. HipMCL is an iterative algorithm that relies on SpGEMM as its workhorse at each iteration. The ExaGraph project ported the hash-based SpGEMM algorithm, which was originally developed for GPUs by collaborators (Nagasaka et al., 2017), into multicore CPUs and Intel KNLs

(Nagasaka et al., 2019). For GPU-equipped clusters, we developed a model to choose the fastest GPU-based SpGEMM depending on the sparsity of the current MCL iteration and utilized a pipelined communication scheme that hides the cost of CPU-to-GPU data transfers. These advances, coupled with a distributed-memory implementation of randomized output structure prediction algorithm, resulted in orders of magnitude speedup compared to the original HipMCL (Selvitopi et al., 2020b).

The HipMCL pipeline interprets the converged matrix after MCL iterations by finding its connected components. To find connected components in distributed-memory, we developed LACC (Azad and Buluç, 2019), which is based on the Awerbuch–Shiloach algorithm (Awerbuch and Shiloach, 1987) but with the added focus on exploiting sparsity. Our CombBLAS-based implementation of LACC heavily relies on SpMSPV like the HWPM algorithm we described earlier but it also utilizes SpMV (sparse matrix-dense vector multiplication) in its earlier iterations. Later, LACC has been improved (Zhang et al., 2020a) with the introduction and integration of another linear-algebraic connected components implementation named FastSV (Zhang et al., 2020b), which is based on the Shiloach–Vishkin algorithm (Shiloach and Vishkin, 1982).

In Table 1, we summarize the tools we have covered in this subsection with the key linear-algebraic primitive(s) they rely on. ExaGraph project also allowed us to increase the computational efficiency of single node implementations of these primitives. SpGEMM is one kernel where the ExaGraph team has made many advances in the last several years. We introduced a two-phase, performance portable, SpGEMM implementation in Kokkos Kernels using a team-level hash map accumulator (Deveci et al., 2017, 2018). We also developed a multicore SpGEMM algorithm (Nagasaka et al., 2019) that uses an efficient hash table for computing row accumulations. The latest advances include an outer product algorithm that uses propagation blocking (Gu et al., 2020). Some of the improvements in the SpGEMM and other linear algebra kernels directly translated to improvements in graph algorithms. For example, the Kokkos Kernels SpGEMM implementation led to highly optimized implementation of the triangle counting kernel (Wolf et al., 2017). Since then, there are other fast implementations of triangle counting kernels based on Cilk (Yaşar et al., 2018) and using both CPUs and GPUs (Yaşar et al., 2019) and distributed memory systems (Acer et al., 2019).

Graph learning

Graph learning is an emerging subfield of machine learning that is concerned with learning on graph-structured data such as graphs describing chemical bonds of molecules, power grid, or transportation networks. ExaGraph

Table 1. ExaGraph supported codes and the basic linear-algebraic primitives they use. SpGEMM: sparse matrix–sparse matrix multiplication, SpMSPV: sparse matrix–sparse vector multiplication, SpMV: sparse matrix–dense vector multiplication. HWPM and PASTIS are developed primarily by the ExaGraph project, whereas HipMCL is primarily developed by the ExaBiome project with pieces supported by ExaGraph.

	SpGEMM	SpMSPV	SpMV
HWPM		✓	
PASTIS	✓		
HipMCL	✓	✓	✓
Kokkos Kernels			
Triangle Count	✓		

supported research on high-performance graph learning through knowledge transfer on graph methods and the use of compute resources. The marginalized graph kernels computation on GPUs enable fast similarity computation among all pairs of graphs in a large database (Tang et al., 2020). The techniques include identification and optimization of the fused sparse Kronecker product times dense vector multiplication operation as well as using state-of-the-art graph partitioning algorithms for minimizing the number of nonzero tiles.

Graph Neural Networks are versatile learners that can be used for classifying or embedding vertices, edges, or entire graphs. Our recent work on distributed-memory GNN training Tripathy et al. (2020) showed that communication-avoiding algorithms greatly accelerate GNN training at the expense of increasing memory requirements. The primary workhorse of GNN training and inference is the sparse matrix-dense matrix product (Yang et al., 2018; Huang et al., 2020). The algorithmic research on marginalized graph kernels and communication-avoiding distributed GNN training has been primarily supported by the ASCR Applied Math program. In related work, the robustness of GNN was studied (Fox and Rajamanickam, 2020).

Software libraries. Our primary software library is the CombBLAS (Buluç and Gilbert, 2011), which provides distributed-memory linear-algebra primitives for implementing complex data analysis applications. Using CombBLAS, we have supported HipMCL, PASTIS, and HWPM libraries described in this section. A high-performance GPU library that provides an implementation of the GraphBLAS-like primitives is the GraphBLAST (Yang et al., 2019), which will be useful for porting the ExaGraph supported codes to GPU-equipped exascale supercomputers.

Applications. The linear-algebraic primitives developed and optimized by ExaGraph have been used in many graph algorithms as well as computational biology and machine

learning pipelines. They also have applications in large sparse solvers as detailed in this subsection.

Graph partitioning

Graph partitioning is a fundamental problem in combinatorial scientific computing. Typically, the objective is to minimize the edge cut (e.g., load balancing for parallel computing), but another variation is to minimize the vertex separator size (for fill-reducing ordering).

The most popular graph partitioning methods (software) are currently based on the multilevel method (Hendrickson and Leland, 1993; Karypis and Kumar, 1998, 1997). However, these methods are hard to parallelize, especially on GPU. In ExaGraph, we chose to revisit the *spectral partitioning* method. Spectral methods solve the partitioning problem using an algebraic approach, based on linear algebra. First, the graph Laplacian matrix for a graph is formed. Then, a small number of eigenpairs are computed. These are used to define a low-dimensional space to embed the graph. A fast geometric partitioning method is then applied to the embedded graph. In the bisection case, only the second-lowest eigenvector is needed, but it may be beneficial to use more than one eigenvector to do multi-section (Hendrickson and Leland, 1995).

The idea of the spectral partitioning goes back to spectral graph theory in the 1970s (Donath and Hoffman, 1972; Fiedler, 1973), and it was later developed into a practical algorithm and used in the context of scientific computing (Pothen et al., 1990). It is an early example of using linear algebra to solve a graph/combinatorial problem. The key insight is that the graph edge cut can be reformulated as $1/4x^T Lx$, where L is the Laplacian matrix and x is an indicator vector where x_i is ± 1 in the bisection case. Relaxing the integer constraint gives the symmetric eigenvalue problem. Hence, the continuous problem approximates the discrete problem. Spectral partitioning for balanced edge cuts was an option in Chaco (Hendrickson and Leland, 1993). Spectral methods have also become popular for data

clustering (Shi and Malik, 2000). A parallel spectral code for clustering is available (Chen et al., 2011), and there is also a GPU code (Naumov and Moon, 2016), but it is limited to a single GPU.

As part of ExaGraph, we have developed a new spectral partitioner, Sphynx (Acer et al. 2020, 2021). The algorithm is based on spectral partitioning but the implementation is targeted highly parallel exascale systems. Sphynx is based on Trilinos (Boman et al., 2007), a state-of-the-art toolkit for scientific computing. The Tpetra linear algebra layer is used for distributed matrices and graphs. The Anasazi package is used for eigensolvers. We found LOBPCG (Knyazev, 2001) works well, and it also supports preconditioning. We have recently integrated several preconditioners from Trilinos to accelerate the convergence. Sphynx has been integrated into the Zoltan2 Trilinos package so it is easily available to Trilinos users.

In summary, the Sphynx spectral partitioner has three phases: (i) Compute the graph Laplacian matrix, (ii) compute a few eigenpairs corresponding to the lowest eigenvalues, and (iii) partition the (embedded) vertices using the geometric MultiJagged method. This is illustrated in Figure 7.

We note one difference from previous work. First, we do not use recursive bisection or octasection. Instead, we compute the eigenvectors once for the global problem and use multiple eigenvectors in the geometric step. The advantage is speed and simplicity, we do not need to move subproblems onto subsets of processors. The downside is partitioning quality could be worse, as we do not capture local structure well.

We have tested and evaluated Sphynx on both regular graphs such as meshes, and highly irregular graphs such as social networks and web graphs. We have also evaluated the use of several different preconditioners: Jacobi, polynomial, and algebraic multigrid (AMG). The Jacobi and polynomial preconditioners run well on GPUs, but are not scalable (in the sense that #iterations grow with problem size). Algebraic multigrid is scalable, but has a high set-up cost (especially on GPU). We observed that no single preconditioner was always

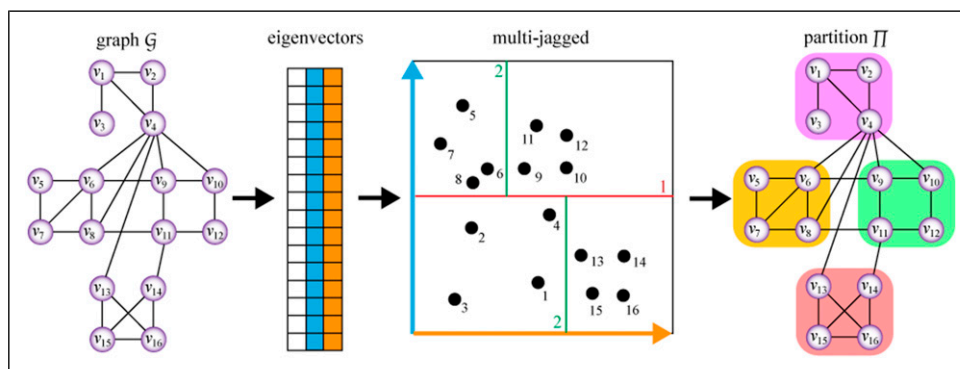


Figure 7. Example of the Sphynx spectral partitioner workflow for a graph with 16 vertices, partitioned into 4 parts.

best, but AMG was often the best choice for regular (mesh-like) graphs while the polynomial preconditioner did well on irregular graphs (such as web graphs and social networks). An advantage of using a preconditioned eigensolver, is that we can easily benefit from new developments in preconditioners for linear solvers. Sphynx attempts to analyze the graph structure and automatically pick a suitable preconditioner.

Sphynx is distributed-memory parallel and can run on multi-GPU systems. We believe this is a unique capability. No other commonly used graph partitioner currently runs on multi-GPU systems.

Sphynx is a single level partitioner, though it can use a multigrid preconditioner. In collaboration with Penn State University, we have recently studied graph coarsening on GPU (Gilbert et al., 2021). We believe this may provide an approach to a faster (perhaps also better quality) multilevel partitioner on GPU.

Software libraries. Sphynx has been released as part of the Zoltan2 package in Trilinos. Sphynx depends on several Trilinos packages. The required dependencies are Tpetra, Teuchos, and Anasazi. Optional dependencies are Belos, Itpack2, and MueLu (for preconditioning).

Applications. Graph partitioning is fundamental algorithmic operation with numerous applications in scientific computing. The most typical use case is load balancing for parallel computing.

The National Security Application Empire and ExaWind ECP projects (Alexander et al., 2020) have mesh partitioning needs and are the intended early users. A potential future extension is to compute vertex separators and nested dissection orderings, which would benefit the Strumpack/SuperLU project.

Summary and future work

Combinatorial and graph algorithms are key enablers for several applications in scientific computing and are increasingly applied for data analysis in the emerging areas of graph analytics and artificial intelligence. Ubiquity of large-scale datasets as well as accelerator platforms, especially in the context of exascale computing has necessitated a fundamental reconsideration of graph algorithms. A codesign center focusing on combinatorial kernels was established in the Exascale Computing Project to address the needs of scientific computing applications. In this paper, we presented a brief survey of the algorithmic and software development efforts spanning combinatorial and algebraic approaches for graph algorithms, and the applications that have benefited from this work.

Our future work will focus on adapting our libraries to exascale platforms including optimizations for performance.

We plan to explore scalable algorithms and software development for graph problems such as network alignment and vertex orderings techniques. We also intend to work closely with application teams and integrate our tools in application workflows.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.


Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. We used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory (Contract DE-AC05-00OR22725), Argonne Leadership Computing Facility at Argonne National Laboratory (Contract DE-AC02-06CH11357), and the National Energy Research Scientific Computing Center (Contract DE-AC02-05CH11231). The Pacific Northwest National Laboratory is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

ORCID iD

Nitin Gawande  <https://orcid.org/0000-0002-5761-1027>

Mahantesh Halappanavar  <https://orcid.org/0000-0002-2323-4753>

Nathan R. Tallent  <https://orcid.org/0000-0003-4297-3057>

References

- Acer S, Yaşar A, Rajamanickam S, et al. (2019) Scalable triangle counting on distributed-memory systems. In: IEEE High Performance Extreme Computing Conference (HPEC), Athens, Greece, 12-15 August 2018, pp. 1–5. IEEE.
- Acer S, Boman EG, and Rajamanickam S (2020) SPHYNX: Spectral partitioning for hybrid and accelerator-enabled systems. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, 18-22 May 2020, pp. 440–449.
- Acer S, Boman EG, Glusa CA, et al. (2021) Sphynx: a parallel multi-GPU partitioner for distributed memory systems. *Parallel Computing*.
- Alexander F, Almgren A, Bell J, et al. (2020) Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society. A, Mathematical, Physical and Engineering Sciences* 378(2166): 1. DOI: [10.1098/rsta.2019.0056](https://doi.org/10.1098/rsta.2019.0056).

- Al-Herz A and Pothen A (2019) A 2/3-approximation algorithm for vertex-weighted matching. *Discrete Applied Mathematics*. Epub ahead of print 19 October 2019. DOI: [10.1016/j.dam.2019.09.013](https://doi.org/10.1016/j.dam.2019.09.013).
- Al-Herz A and Pothen A (2020) A parallel 2/3-approximation algorithm for vertex-weighted matching. In: Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, pp. 12–21.
- Altschul SF, Gish W, Miller W, et al. (1990) Basic local alignment search tool. *Journal of molecular biology* 215(3): 403–410.
- Awerbuch B and Shiloach Y (1987) New connectivity and MSF algorithms for shuffle-exchange network and PRAM. *IEEE Transactions on Computers* 10(C-36): 1258–1263.
- Azad A and Buluç A (2016) Distributed-memory algorithms for maximum cardinality matching in bipartite graphs. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, USA, 8-11 September 2015, pp. 32–42. IEEE.
- Azad A and Buluç A (2019) LACC: a linear-algebraic algorithm for finding connected components in distributed memory. In: International Parallel and Distributed Processing Symposium IPDPS, Rio de Janeiro, Brazil, 20-24 May 2019, pp. 2–12. New York, NY, USA: IEEE.
- Azad A, Pavlopoulos GA, Ouzounis CA, et al. (2018) HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. *Nucleic Acids Research* 46(6): e33.
- Azad A, Buluç A, Li XS, et al. (2020) A distributed-memory algorithm for computing a heavy-weight perfect matching on bipartite graphs. *SIAM Journal on Scientific Computing* 42(4): C143–C168.
- Barik R, Minutoli M, Halappanavar M, et al. (2020) Vertex reordering for real-world graphs and applications: an empirical evaluation. In: 2020 IEEE International Symposium on Workload Characterization, Beijing, China, 27-30 October 2020.
- Blondel V, Guillaume J-L, Lambiotte R, et al. (2008) Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008(10).
- Bodenheimer T, Halappanavar M, Jefferys S, et al. (2020) Fastpg: fast clustering of millions of single cells. bioRxiv.
- Bogle I, Boman EG, Devine KD, et al. (2020) Distributed memory graph coloring algorithms for multiple gpus. In: Proc. of the 2020 IAAA Workshop at SC20, GA, USA, 11-11 November 2020. IEEE.
- Boman E, Devine K, Heaphy R, et al. (2007) *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*. Tech. Report SAND2007-4748W. Albuquerque, NM, USA: Sandia National Laboratories.
- Boman EG, Devine KD, Leung VJ, et al. (2012a) *Zoltan2: Next-Generation Combinatorial Toolkit*. Technical Report SAND2012-9373C. Albuquerque, NM, USA: Sandia National Laboratories.
- Boman EG, Çatalyürek ÜV, Chevalier C, et al. (2012b) The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering, and coloring. *sciprog* 20(2): 129–150.
- Bozdağ D, Gebremedhin AH, Manne F, et al. (2008) A framework for scalable greedy coloring on distributed-memory parallel computers. *Journal of Parallel and Distributed Computing* 68(4): 515–535.
- Buchfink B, Xie C and Huson DH (2015) Fast and sensitive protein alignment using DIAMOND. *Nature Methods* 12(1): 59.
- Buluç A and Gilbert JR (2011) The combinatorial blas: design, implementation, and applications. *The International Journal of High Performance Computing Applications* 25(4): 496–509.
- Buluç A and Gilbert JR (2012) Parallel sparse matrix-matrix multiplication and indexing: implementation and experiments. *SIAM Journal on Scientific Computing* 34(4): C170–C191.
- Buluç A, Mattson T, McMillan S, et al. (2017) Design of the graphblas api for c. In: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May-2 June 2017, pp. 643–652. IEEE.
- Chavarría-Miranda D, Halappanavar M, Krishnamoorthy S, et al. (2015) On the impact of execution models: a case study in computational chemistry. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, Hyderabad, India, 25-29 May 2015, pp. 255–264. DOI: [10.1109/IPDPSW.2015.111](https://doi.org/10.1109/IPDPSW.2015.111).
- Chen W, Song Y, Bai H, et al. (2011) Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33(3): 568–586.
- Cheng X, Gebremedhin A, Patwary M, et al. (2020a) Colpack: a graph coloring library for derivative matrix computation and beyond. www.github.com/CSCsw/ColPack.
- Cheng X, Maji HK and Pothen A (2020b) Graphs with tunable chromatic numbers for parallel coloring. In: Proceedings of the SIAM Workshop on Combinatorial Scientific Computing pp. 11–20.
- Davis TA (2006) *Direct Methods for Sparse Linear Systems*. SIAM.
- Davis TA, Rajamanickam S and Sid-Lakhdar WM (2016) A survey of direct methods for sparse linear systems. *Acta Numerica* 25: 383–566.
- Demmel JW, Eisenstat SC, Gilbert JR, et al. (1999) A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications* 20: 720–755.
- Deveci M, Boman EG, Devine KD, et al. (2016a) Parallel graph coloring for manycore architectures. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, USA, 23-27 May 2016, pp 892–901. IEEE.
- Deveci M, Rajamanickam S, Devine KD, et al. (2016b) Multi-jagged: a scalable parallel spatial partitioning algorithm. *IEEE Transactions on Parallel and Distributed Systems* 27(3): 803–817.
- Deveci M, Trott C and Rajamanickam S (2017) Performance-portable sparse matrix-matrix multiplication for many-core

- architectures. In 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May-2 June 2017, pp. 693–702. IEEE.
- Deveci M, Trott C and Rajamanickam S (2018) Multithreaded sparse matrix-matrix multiplication for many-core and gpu architectures. *Parallel Computing* 78: 33–46.
- Deveci M, Devine K, Pedretti K, et al. (2019) Geometric mapping of tasks to processors on parallel computers with mesh or torus networks. *IEEE Transactions on Parallel and Distributed Systems* 30(9): 2018–2032.
- Dobrian F, Halappanavar M, Pothén A, et al. (2019) A 2/3-approximation algorithm for vertex-weighted matching in bipartite graphs. *SIAM Journal on Scientific Computing* 41(1): A566–A591.
- Domingos PM and Richardson M (2001) Mining the network value of customers. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, 26-29 August 2001, pp. 57–66. ACM.
- Donath W and Hoffman A (1972) Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin* 15: 938–944.
- Duff IS and Koster J (1999) The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications* 20(4): 889–901.
- Duff IS, Erisman AM and Reid JK (2017) *Direct Methods for Sparse Matrices*. 2nd edition. Oxford: Oxford University Press.
- Edwards HC, Trott CR and Sunderland D (2014) Kokkos: enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing* 74(12): 3202–3216.
- Enright AJ, Van Dongen S and Ouzounis CA (2002) An efficient algorithm for large-scale detection of protein families. *Nucleic acids research* 30(7): 1575–1584.
- Ferdous S, Pothén A and Khan A (2018) New approximation algorithms for minimum weighted edge cover. In: Proceedings of SIAM Workshop on Combinatorial Scientific Computing, pp. 97–108.
- Fiedler M (1973) Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal* 23(98): 298–305.
- Fortunato S (2010) Community detection in graphs. *Physics Reports* 486(3-5): 75–174.
- Fox JS and Rajamanickam S (2020) How robust are graph neural networks to structural noise? In: The First International Workshop on Deep Learning on Graphs: Methodologies and Applications (DLGMA’20).
- Gebremedhin AH and Manne F (2000) Scalable parallel graph coloring algorithms. *Concurrency: Practice and Experience* 12(12): 1131–1146.
- Gebremedhin AH, Manne F and Pothén A (2005a) What color is your jacobian? graph coloring for computing derivatives. *SIAM Review* 47(4): 629–705.
- Gebremedhin AH, Manne F and Pothén A (2005b) What color is your Jacobian? graph coloring for derivatives. *SIAM Review* 47(4): 629–705.
- Ghosh S, Halappanavar M, Tumeo A, et al. (2018a) Scalable distributed memory community detection using vite. In: 2018 IEEE High Performance extreme Computing Conference (HPEC), Waltham, MA, USA, 25-27 September 2018, pp. 1–7. IEEE.
- Ghosh S, Halappanavar M, Tumeo A, et al. (2018b) Distributed Louvain algorithm for graph community detection. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, BC, Canada, 21-25 May 2018, pp. 885–895. IEEE.
- Ghosh S, Halappanavar M, Tumeo A, et al. (2019) Scaling and quality of modularity optimization methods for graph clustering. In: 2019 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 24-26 September 2019, pp. 1–6. IEEE.
- Gilbert MS, Acer S, Boman EG, et al. (2021) Performance-portable graph coarsening for efficient multilevel graph analysis. In: 2021 Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17-21 May 2021.
- Griewank A and Walther A (2008) *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd edition. SIAM.
- Gu Z, Moreira J, Edelsohn D, et al. (2020) Bandwidth-optimized parallel algorithms for sparse matrix-matrix multiplication using propagation blocking. In: Symposium on Parallelism in Algorithms and Architectures (SPAA).
- Halappanavar M, Lu H, Kalyanaraman A, et al. (2017) Scalable static and dynamic community detection using grappolo. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 12-14 September 2017, pp. 1–6. IEEE.
- Hendrickson B and Leland R (1993) The chaco user’s guide. Version 1.0. DOI: [10.2172/10106339](https://doi.org/10.2172/10106339).
- Hendrickson B and Leland R (1995) An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing* 16(2): 452–469.
- Heroux MA, Bartlett RA, Howle VE, et al. (2005) An overview of the Trilinos project. *ACM Transactions on Mathematical Software* 31(3): 397–423.
- Huang G, Dai G, Wang Y, et al. (2020) GE-SpMM: general-purpose sparse matrix-matrix multiplication on GPUs for graph neural networks. arXiv preprint arXiv:2007.03179.
- Hysom DA and Pothén A (2001) A scalable parallel algorithm for incomplete-factor preconditioning. *SIAM Journal on Scientific Computing* 22(6): 2194–2215.

- Jones MT and Plassmann PE (1993) A parallel graph coloring heuristic. *SIAM Journal on Scientific Computing* 14(3): 654–669.
- Karypis G and Kumar V (1997) *Parmetis: Parallel graph partitioning and sparse matrix ordering library*. Technical report. University of Minnesota.
- Karypis G and Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1): 359–392.
- Kempe D, Kleinberg JM and Tardos É (2003) Maximizing the spread of influence through a social network. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 24–27 August 2003, pp. 137–146. ACM.
- Kepner J, Aaltonen P, Bader D, et al. (2016) Mathematical foundations of the graphblas. In: IEEE High Performance Extreme Computing Conference (HPEC) pp. 1–9. IEEE.
- Khan A, Choromanski K, Pothen A, et al. (2018a) Adaptive anonymization of data using b -edge cover. In: Proceedings of Supercomputing, Dallas, TX, USA, 11–16 November 2018, pp. 743–753. IEEE.
- Khan A, Pothen A and Ferdous S (2018b) Parallel algorithms through approximation: b -edge cover. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, BC, Canada, 21–25 May 2018, pp. 22–33.
- Knyazev AV (2001) Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing* 23(2): 517–541.
- Li XS and Demmel JW (1998) Making sparse gaussian elimination scalable by static pivoting. In: SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Orlando, FL, USA, 7–13 November 1998, pp. 34. IEEE.
- Li XS and Demmel JW (2003) Superlu_dist: a scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software (TOMS)* 29(2): 110–140.
- Liu JW-H (1990) The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications* 11: 134–172.
- Lu H, Halappanavar M, Chavarría-Miranda D, et al. (2017) Algorithms for balanced graph colorings with applications in parallel computing. *IEEE Transactions on Parallel and Distributed Systems* 28(5): 1240–1256.
- Marathe M and Vullikanti A (2013) Computational epidemiology. *Communications of the ACM* 56(7): 88–96.
- Minutoli M, Halappanavar M, Kalyanaraman A, et al. (2019) Fast and scalable implementations of influence maximization algorithms. In: 2019 IEEE International Conference on Cluster Computing CLUSTER, Albuquerque, NM, USA, 23–26 September 2019, pp. 1–12. IEEE.
- Minutoli M, Drocco M, Halappanavar M, et al. (2020a) cuRipples: influence maximization on multi-GPU systems. In: Proceedings of the International Conference on Supercomputing 2020 (ICS20).
- Minutoli M, Sambaturu P, Halappanavar M, et al. (2020b) PREEMPT: scalable epidemic interventions using sub-modular optimization on multi-GPU system. In: To appear in The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC20), Atlanta, GA, USA, 9–19 November 2020.
- Nagasaka Y, Nukada A and Matsuoka S (2017) High-performance and memory-saving sparse general matrix-matrix multiplication for nvidia pascal gpu. In: 6th International Conference on Parallel Processing (ICPP), Bristol, UK, 14–17 August 2017, pp. 101–110.
- Nagasaka Y, Matsuoka S, Azad A, et al. (2019) Performance optimization, modeling and analysis of sparse matrix-matrix products on multi-core and many-core processors. *Parallel Computing* 90: 102545.
- Naumov M and Moon T (2016) Parallel spectral graph partitioning. NVIDIA Tech. Rep. NVR-2016-001. NVIDIA.
- Newman MEJ and Girvan M (2004) Finding and evaluating community structure in networks. *Physical Review E* 69(2): 026113.
- NVIDIA (2020) Nvidia rapids cudgraph. <https://github.com/rapidsai/cugraph>.
- Parter SV (1961) The use of linear graphs in Gaussian elimination. *SIAM Review* 3: 119–130.
- Pothen A, Simon H and Liou K (1990) Partitioning sparse matrices with eigenvectors of graphs. *The SIAM Journal on Matrix Analysis* 11(3): 430–452.
- Pothen A, Ferdous SM and Manne F (2019) Approximation algorithms in combinatorial scientific computing. *Acta Numerica* 28: 541–633.
- Purvine E, Cotilla-Sanchez E, Halappanavar M, et al. (2017) Comparative study of clustering techniques for real-time dynamic model reduction. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 10 (5): 263–276.
- Richards DF, Aaziz O, Cook J, et al. (2018) *Fy18 proxy app suite release. milestone report for the ecp proxy app project*. Technical report. Livermore, CA, USA: Lawrence Livermore National Lab.(LLNL).
- Rose DJ (1972) A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Read RC (ed) *Graph Theory and Computing*. New York: Academic Press, 183–217.
- Rose DJ, Tarjan RE and Lueker GS (1976) Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing* 5: 266–283.
- Rouet F-H, Li XS, Ghysels P, et al. (2016) A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Transactions on Mathematical Software (TOMS)* 42(4): 1–35.
- Sallinen S, Iwabuchi K, Poudel S, et al. (2016) Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In: Proceedings of the International

- Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, Salt Lake City, UT, USA, 13-18 November 2016. IEEE Press.
- Sambaturu P, Adhikari B, Prakash BA, et al. (2020) Designing effective and practical interventions to contain epidemics. In: Proc. AAMAS.
- Selvitopi O, Ekanayake S, Guidi G, et al. (2020a) Distributed many-to-many protein sequence alignment using sparse matrices. In: Proceedings of the 2020 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'20.
- Selvitopi O, Hussain MT, Azad A, et al. (2020b) Optimizing high performance Markov clustering for pre-exascale architectures. In: International Parallel and Distributed Processing Symposium IPDPS.
- Shi J and Malik J (2000) Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8): 888–905.
- Shiloach Y and Vishkin U (1982) An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms* 3(1): 57–67.
- Steinegger M and Söding J (2017) Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology* 35(11): 1026.
- Tang Y, Shi Y and Xiao X (2015) Influence maximization in near-linear time: a martingale approach. In: Proc. 2015 ACM SIGMOD International Conference on Management of Data, pp. 1539–1554. ACM.
- Tang Y-H, Selvitopi O, Popovici D, et al. (2020) A high-throughput solver for marginalized graph kernels on GPU. In: International Parallel and Distributed Processing Symposium IPDPS.
- Trilinos Project Team. *The Trilinos Project Website*.
- Tripathy A, Yelick K and Buluç A (2020) Reducing communication in graph neural network training. In: Proceedings of the 2020 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'20.
- Van Dongen SM (2000) *Graph clustering by flow simulation*. PhD thesis.
- Wolf MM, Deveci M, Berry JW, et al. (2017) Fast linear algebra-based triangle counting with kokkoskernels. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 12-14 September 2017, pp. 1–7. IEEE.
- Yang C, Buluç A and Owens JD (2018) Design principles for sparse matrix multiplication on the gpu. In: European Conference on Parallel Processing, pp. 672–687. Springer.
- Yang C, Buluc A and Owens JD (2019) Graphblast: a high-performance linear algebra-based graph framework on the gpu. arXiv preprint arXiv:1908.01407.
- Yaşar A, Rajamanickam S, Wolf M, et al. (2018) Fast triangle counting using cilk. In: 2018 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. IEEE.
- Yasar A, Rajamanickam S, Berry JW, et al. (2019) *Linear algebra-based triangle counting via fine-grained tasking on heterogeneous environments*. Technical Report. Albuquerque, NM, USA: Sandia National Lab.(SNL-NM).
- Zhang Y, Azad A and Buluç A (2020a) Parallel algorithms for finding connected components using linear algebra. *Journal of Parallel and Distributed Computing* 144: 14–27.
- Zhang Y, Azad A and Hu Z (2020b) FastSV: a distributed-memory connected component algorithm with fast convergence. In: Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing, pp. 46–57. SIAM.

Author biographies

Seher Acer is a postdoc in the Center for Computing Research at Sandia National Labs. She holds a PhD from Bilkent University, Turkey. Her research interests are in combinatorial scientific computing and graph algorithms.

Ariful Azad is an Assistant Professor in the Department of Intelligent Systems Engineering at Indiana University. He obtained his Ph.D. from Purdue University. He was a research scientist at Lawrence Berkeley National Laboratory. His research interests are in high-performance computing, graph algorithms, machine learning, and bioinformatics.

Erik G. Boman is a research scientist (PMTS) at the Center for Computing Research at Sandia National Labs. He received a PhD from Stanford University. His research interests are in combinatorial scientific computing, high-performance parallel computing, sparse linear algebra, and preconditioners. Dr. Boman is currently an associate editor of SIAM Journal on Scientific Computing (SISC) and SIAM Review.

Aydın Buluç is a Staff Scientist and Principal Investigator at the Lawrence Berkeley National Laboratory (LBNL) and an Adjunct Assistant Professor of EECS at UC Berkeley. His research interests include parallel computing, combinatorial scientific computing, high-performance graph analysis and machine learning, sparse matrix computations, and computational biology. Previously, he was a Luis W. Alvarez postdoctoral fellow at LBNL and a visiting scientist at the Simons Institute for the Theory of Computing. He received his PhD in Computer Science from the University of California, Santa Barbara, in 2010 and his BS in Computer Science and Engineering from Sabanci University, Turkey in 2005. Dr. Buluç is a recipient of the DOE Early Career Award in 2013 and the IEEE TCSC Award for Excellence for Early Career Researchers in 2015. He is also a founding associate editor of the ACM Transactions on Parallel Computing.

Karen D. Devine is a Distinguished Member of Technical Staff at the Center for Computing Research at Sandia National Labs. She earned her PhD from Rensselaer Polytechnic Institute. Her research interests include high-performance computing applications and algorithms,

parallel partitioning and load balancing, and task mapping and placement.

S M Ferdous is a PhD candidate in computer science at Purdue University, USA. Ferdous's research interest is in combinatorial scientific computing and high-performance computing. Particularly, his interest lies in applying algorithmic techniques (e.g., approximation and randomization) to solve practical, real-world problems.

Nitin Gawande is Software Application Engineer in the Architecture Workload Engineering team of the Technical Computing Engineering Group, in Architecture, Graphics, and Software at Intel Corporation. The work that led to his contribution to this publication was conducted during his tenure as a research scientist with the Data Sciences and Machine Intelligence Group of the Pacific Northwest National Laboratory at Richland WA (USA). Nitin's research interests include design and implementation of scalable high-performance computing algorithms on GPU accelerators, and he is also part of the Global Arrays, a PGAS model development team.

Sayan Ghosh is a Computer Scientist in the Data Sciences and Machine Intelligence Group at the Pacific Northwest National Laboratory in Richland, WA. His research interests are broadly in the application of parallel programming models and communication optimizations for building scalable scientific codes. He holds a master's degree from University of Houston in Houston, TX and a PhD degree (both in Computer Sciences) from Washington State University in Pullman, WA.

Mahantesh Halappanavar is a chief computer scientist and group leader of the Data Sciences and Machine Intelligence Group at the Pacific Northwest National Laboratory. He holds a joint appointment as adjunct faculty in Computer Science at the School of Electrical Engineering and Computer Science, Washington State University in Pullman. His research has spanned multiple technical foci and includes combinatorial scientific computing, parallel graph algorithms, machine learning, and application of graph theory and game theory to solve problems in application domains such as scientific computing, power grids, cybersecurity, and life sciences. He is a member of Society for Industrial and Applied Mathematics (SIAM), and Senior Member of Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE).

Ananth Kalyanaraman is a Professor and Boeing Centennial Chair in Computer Science at the School of Electrical Engineering and Computer Science, Washington State University in Pullman. He also serves as the Associate Director for the School of EECS, holds a joint appointment with Pacific Northwest National Laboratory (PNNL) as a

Senior Scientist, and holds affiliate faculty positions at the WSU Molecular Plant Sciences Graduate Program and the Paul G. Allen School for Global Animal Health. Ananth received his Ph.D. from Iowa State University. Ananth works at the intersection of parallel computing, graph analytics, and bioinformatics/computational biology. His focus is on developing algorithms and software for scalable analysis of large-scale data from various scientific domains and particularly the life sciences. He is currently serving as a Vice-Chair for the IEEE Technical Committee on Parallel Programming (TCPP). Ananth is a senior member of IEEE, and a member of ACM and SIAM.

Arif Khan is a computer scientist in the Data Sciences and Machine Intelligence Group at Pacific Northwest National Laboratory. He received his Ph.D. from Purdue University. His research interests include graph algorithm, high-performance computing, approximation algorithm along with their applications in bioinformatics, quantum computing, social network and machine learning. His goal is to explore how approximation algorithms can solve big graph problems using leadership class supercomputers. His recent research includes application of graph algorithms proteomics, quantum computing, and traffic analysis. He is a member of Society for Industrial and Applied Mathematics (SIAM), Association for Computing Machinery (ACM), and Institute of Electrical and Electronics Engineers (IEEE).

Marco Minutoli is a research scientist in the Data Sciences and Machine Intelligence Group at Pacific Northwest National Laboratory. He received his Bachelor of Science in Computer and Telecommunication Engineering from University of Messina, Messina, Italy, in 2008, his Master of Science in Computer Engineering from Politecnico di Milano, Milan, Italy, in 2014, and he is currently pursuing his PhD in Computer Science at Washington State University, Pullman, WA. His research focuses on the design of parallel graph algorithms and their acceleration. He is also active in the Design Automation community where he works on the definition of HW/SW codesign and High Level Synthesis methodologies and their compilation and optimization pipelines for the generation of custom computing devices.

Alex Pothén is a Professor of Computer Science at Purdue. His research interests are in combinatorial scientific computing (CSC), parallel computing, and bioinformatics algorithms. He is a Fellow of the Society for Industrial and Applied Mathematics (SIAM). He is the Chair of the SIAM Activity Group on Applied and Computational Discrete Algorithms. He served as the Director of the CSCAPES Institute, a pioneering research center in CSC (2006–2012), Director of Purdue's Computing Research Institute (2008–2010), and Associate Head of computer science (2015–2018).

Sivasankaran Rajamanickam is a principal member of technical staff at the Center for Computing Research at

Sandia National Laboratories. He received his PhD from the University of Florida. His research interests are in high performance computing, combinatorial scientific computing, machine learning, and sparse linear algebra. Dr. Rajamanickam is a member of SIAM, ACM, and IEEE.

Oguz Selvitopi is a Research Scientist in the Performance and Algorithms group of Computer Science Department at Lawrence Berkeley National Laboratory. His research interests are high performance computing, parallel sparse matrix computations, combinatorial scientific computing, and bioinformatics. Oguz received his Ph.D. in computer engineering from Bilkent University, Turkey in 2016.

Nathan Tallent is a computer scientist and lead of the Scalable Computing & Data Team in the High Performance Computing group at Pacific Northwest National Laboratory. His research focuses on developing techniques for characterizing, analyzing, and accelerating the performance of

current and emerging workloads in scientific modeling, analytics, and data-intensive scientific workflows. He received a Ph.D. in 2010 from Rice University. He is a member of IEEE and ACM.

Antonino Tumeo received his M.S degree in Informatic Engineering, in 2005 and his Ph.D degree in Computer Engineering, in 2009, from Politecnico di Milano in Italy. Since February 2011, he has been a research scientist in the PNNL's High Performance Computing Group. He joined PNNL in 2009 as a postdoctoral research associate. Previously, he was a postdoctoral researcher at Politecnico di Milano. His research interests are modeling and simulation of high-performance architectures, hardware–software codesign, FPGA prototyping and GPGPU computing, with a specific focus on data analytics workloads and irregular applications. He is a senior member of IEEE and of ACM.