For most of the past 60 years, the world's largest computers have been used to model physical phenomena like weather, mechanics, electronics, and so on, mostly described by differential equations. Linear algebra has played a crucial (one might almost say magical) role as "middleware" between continuous physical models to be computed and the simple arithmetic operations implemented by actual computer hardware. Over the years, the field of computational linear algebra has developed standard algorithmic primitives, high-performance software libraries, and a deep understanding of how to map problems to high-performance computer architectures.

Nowadays, many of the largest computations concern networks and graphs -- connections among web pages, financial transactions, genomes, people. From a mathematical point of view, graph theory occupies a similar position as middleware between models of discrete structures and actual computers (Figure 1). The field of computational graph theory is, however, in its infancy.
Here is an illustration. The current champion among computers by the "Top500" linear algebra benchmark does 33.8 petaflops, which is 33,800,000,000,000,000 arithmetic operations per second. By comparison, the current champion on the "Graph500" benchmark does 15.3 terateps, which is 15,300,000,000,000 edge traversals per second. The ratio is about 2,200 : 1 (Figure 2). Strikingly, though, if we look back only to 2010, that ratio was about 380,000 : 1 -- that is, in just three years the state of the art of computational graph theory has improved relative to computational linear algebra by more than two orders of magnitude.

The goal of our project, which began in 2006, has been to develop, implement, and make available to the scientific and engineering community the same kind of tools for computational graph theory -- standard algorithmic primitives, high-performance software libraries, and mappings of problems to computer architectures -- that computational linear algebra has developed over the years. Our primary vehicles for this have been two high-performance software libraries, the Combinatorial BLAS and the Knowledge Discovery Toolbox (Figure 3). Both are available as open-source software for public use, from the web site http://kdt.sourceforge.net/.

The Combinatorial BLAS (CombBLAS) is a low-level library of graph primitives based on algebraic semirings and parallel sparse matrix data structures and algorithms. It is implemented portably in C++ and MPI, and achieves excellent performance on large, distributed-memory high-performance computers. Combinatorial BLAS was used, for example, in Beamer et al.'s development of the direction-optimizing breadth-first search algorithm that has been adopted by all the Graph500 champion implementations.

The Knowledge Discovery Toolbox (KDT) is a higher-level system, which allows a domain scientist or analyst to use the high-level Python language to perform analytic computations on attributed semantic graphs using parallel supercomputers. Many emerging data analytics applications use attributed semantic graphs. A semantic graph has vertices representing entities and edges representing relationships among the entities; vertices and edges carry labels called "attributes". For example, in a credit card fraud analytics application, the graph might have vertices for banks, credit cards, customers, and merchants, and an edge for each financial transaction; edge attributes might include dates, times, amounts, and locations. Graph algorithms like clustering, shortest paths, and centrality are used to search for patterns that could represent fraud. CombBLAS contains primitives that allow such an algorithm to run fast on a parallel computer; the algorithm's need to take account of attributes (vertex types and edge labels) is satisfied by defining a suitable underlying semiring for the algebra. This is quite flexible and efficient, but not very friendly to a user who wants to think in terms of graph algorithms

rather than semiring operations. KDT, therefore, is a higher-level system that serves as one example of how to use the CombBLAS primitives in a user-friendly way.

KDT includes a mechanism to allow the user to filter semantic graph queries through predicates on edge and vertex attributes that limit the query to a portion of the graph, without explicitly instantiating the subgraph. This implicit filtering is important in applications where the underlying semantic graph is extremely large, and it is too expensive to make a complete copy of a subgraph containing only the information relevant to a particular analytic query.

A bottleneck in the early versions of KDT was that high-level Python filters (or other per-edge and per-vertex actions) slow down the computation, because the efficient low-level CombBLAS library needed to call back to Python each time it manipulates an edge or vertex. The current release of KDT removes this bottleneck. Here we collaborated with Armando Fox and Shoaib Kamil of UC Berkeley and MIT, whose "SEJITS" (Selective Embedded Just-In-Time Specialization) system makes it possible to define a domain-specific subset of Python that can be compiled at runtime into efficient C code for semiring operations within CombBLAS. Integrating KDT with SEJITS effectively removes the performance penalty of Python user-defined filters and edge/vertex operations.