Fast, efficient graph analysis is important and pervasive. However, achieving high levels of performance is exceedingly difficult especially in the era of complex heterogeneous high-performance computing (HHPC) architectures. By defining a set of graph primitives and operations, we are able to separate the concerns between the graph expertise needed to develop advanced graph analytics and the hardware expertise needed to achieve high levels of performance on the ever-increasing complexity of the underlying hardware.

**Graph Algorithms: Simplified by GraphBLAS API**

**Algorithms Implemented with Less Code.** We are developing a library of graph algorithms that are implemented in terms of the new operations and data primitives currently defined by the GraphBLAS API. Classes of algorithms include:
- Metrics: e.g., degree, diameter, centrality, triangle counting
- Traversals: Breadth-First Search (BFS)
- Shortest Path/Cost Minimization
- Community Detection/Clustering
- Connected Components
- (Minimum) Spanning Tree
- Maximum Flow
- PageRank

**Separation of Concerns: GraphBLAS API Spec Standardization In Progress.** Researchers from the SEI, industry, academia, and the U.S. government are developing the API specification:
- The mathematical properties are defined by semi-ring algebra.
- Nine operations are specified currently (see right).
- The key primitive type is the sparse matrix.
- We are exploring extensions to this set of operations that can offer greater expressivity and greater opportunities for tuning.

**Graph Primitives: Tuned for GPU Architectures**

**Collaboration with Indiana University.** Researchers including Andrew Lumsdaine from the Center for Research in Exascale Technologies have been collaborating with the SEI on this project to explore efficient implementations of graph primitives. The graph at the right shows the performance of our BFS algorithm (orange) using a compressed, sparse row matrix format on the newest generation of GPU cards using implementations of graph primitives. The graph at the right shows the performance of our BFS algorithm (orange) using a compressed, sparse row matrix format on the newest generation of GPU cards using implementations of graph primitives.

**Future Work.**
- Continued participation in the GraphBLAS standardization effort
- Addressing scaling issues for large graphs
- Developing distributed primitives to support multiple GPU nodes.
- Tuning for a variety of different sparse matrix formats that will be required for high performance across a wide range of algorithms
- Future versions that include sparse solvers to support other important algorithms (e.g., PCA, graph partitioning)

**The BFS algorithm implemented using only five GraphBLAS operations.** With the masking extension proposed for matrix multiplies, BFS could be implemented with only three operations.

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildMatrix</td>
<td>Build a sparse matrix from row, column, value tuples</td>
</tr>
<tr>
<td>ExtractTuples</td>
<td>Extract the row, column, value tuples from a sparse matrix</td>
</tr>
<tr>
<td>MxM, MxV, VxM</td>
<td>Perform sparse matrix multiplication (e.g., BFS traversal)</td>
</tr>
<tr>
<td>Extract</td>
<td>Extract a sub-matrix from a larger matrix (e.g., sub-graph selection)</td>
</tr>
<tr>
<td>Assign</td>
<td>Assign to a sub-matrix of a larger matrix (e.g., sub-graph assignment)</td>
</tr>
<tr>
<td>Union/UnionAll</td>
<td>Elementwise addition and multiplication of matrices (e.g., graph union, intersection)</td>
</tr>
<tr>
<td>Apply</td>
<td>Apply unary function to each element of matrix (e.g., edge weight modification)</td>
</tr>
<tr>
<td>Reduce</td>
<td>Reduce along column or rows of matrices (vertex degree)</td>
</tr>
<tr>
<td>Transpose</td>
<td>Transposes the rows and columns of a sparse matrix (e.g., reverse directed edges)</td>
</tr>
</tbody>
</table>

**Contact:** Scott McMillan  smcmillan@sei.cmu.edu

---

**Software Engineering Institute | Carnegie Mellon University**