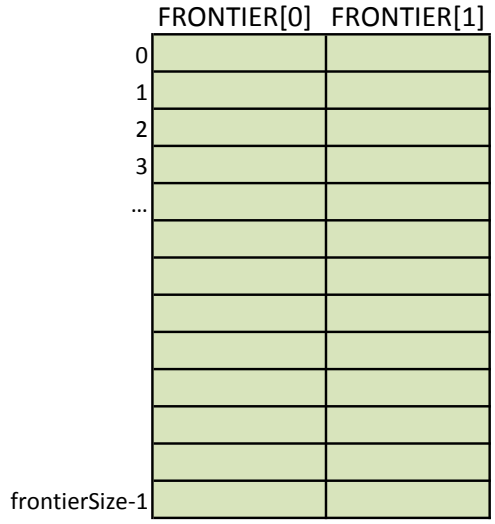
MPGraph Data Layout	
Engine Data	Engine data is the same for all algorithms. Engine data includes the frontier and the graph topology. The specific data structures for the engine data may evolve, e.g., to support compression, partitioning, etc.
User Data	User data is specific to a given algorithm. It is laid out in a structure of arrays format in order to maximize coalesced memory access.

Frontier Queues

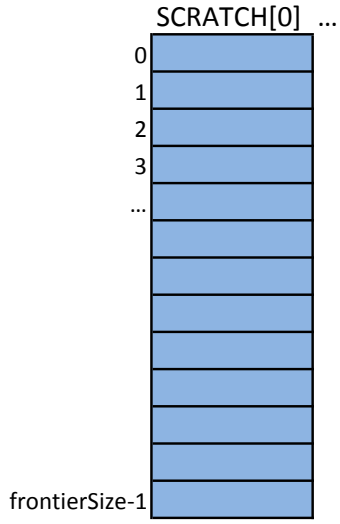
MPGraph uses compact queues for the frontier and can also maintain user data in queues that are 1:1 with the frontier. The frontier array dimensions are determined by the number of vertices in the graph *times* the frontier queue size multiplier. The frontier is in global memory, but is buffered in shared memory by some kernels.

Vertex Frontier



The frontier arrays maintain the list of active vertices. There are two arrays for double-buffering. The frontier is populated by the expand() kernel. The contract() kernel may be used to eliminate some or all of the duplicates depending on the strategy and the needs of the graph algorithm.

User Data

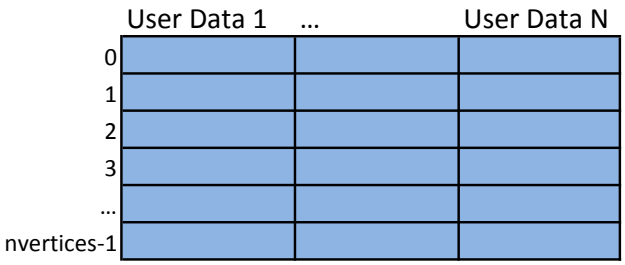


In addition to the frontier, you may allocate optional user data arrays that are 1:1 with the active vertices in the frontier. These arrays provides an important scratch area for many calculations and benefit from dense, coalesced access. The arrays are accessed from the same kernels that operate on the vertex frontier.

User Data

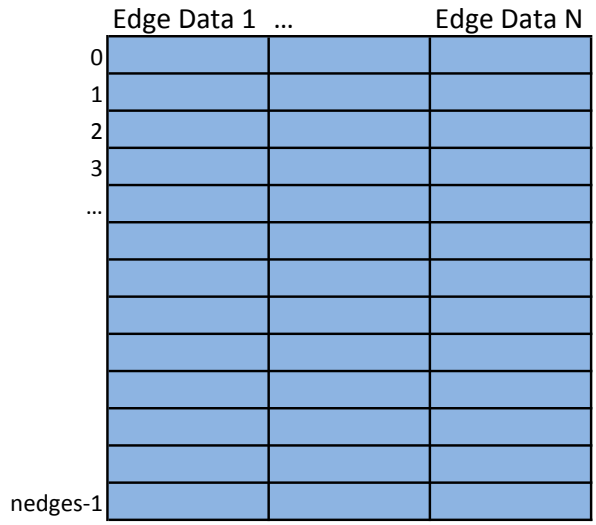
User data is specific to a given algorithm. It is laid out in a structure of arrays format in order to maximize coalesced memory access. (User data can also be 1:1 with the frontier.)

VertexList <VertexData>



The VertexList is a structure of named arrays that provides data for each vertex in the graph. The index into each array is the vertexId.

EdgeList <EdgeData>

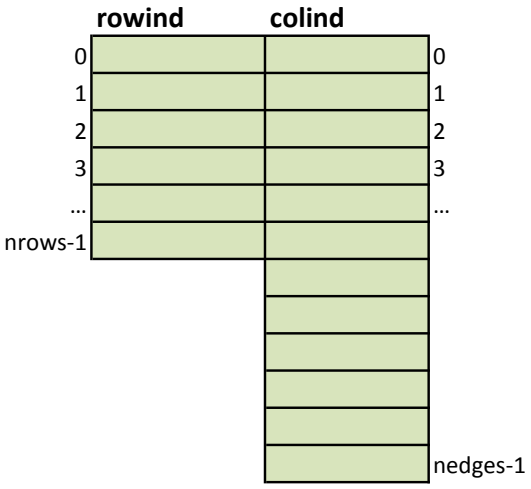


The EdgeList is a structure of named arrays that provides data for each edge in the graph. The index into each array is the edgeId. The CSR.colind[] and the CSC.edgeId[] are both 1:1 with the edge list arrays.

Topology

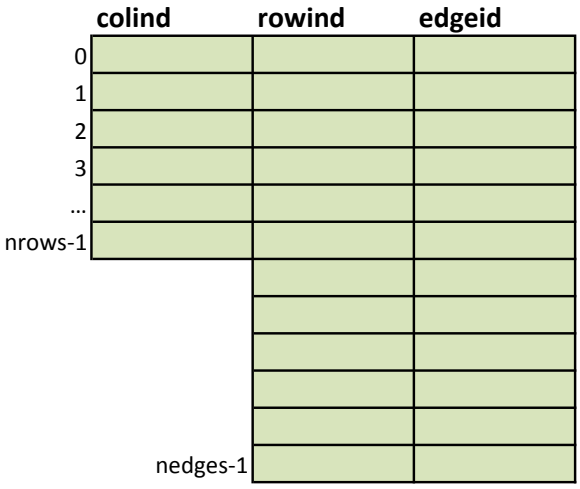
The topology of the graph is modeled by a forward and reverse sparse matrix. These data structures are not directly exposed to user algorithms and their internals may change. The examples below illustrate the use of CSR and CSC data structures to model the graph topology. Users write device functions that are invoked from kernels that process the topology using a variety of different strategies.

CSR



Compressed Sparse Row provides row based indexing into the graph. This data structure is not directly exposed to user algorithms. CSR is used for the out-edges of the graph.

CSC



Compressed Sparse Column provides column based indexing into the graph. This data structure is not directly exposed to user algorithms. CSC is used for the in-edges of the graph. The CSC edgeId array gives the index into the EdgeList arrays. The CSC data structure is only maintained if the algorithm will read over the in-edges.