Latest Development of Gunrock: a Graph Processing Library on GPUs

Yuechao Pan, with the Gunrock team
ychpan@ucdavis.edu

GTC 2018, 28 March 2018, San Jose, California, U.S.A.

GUNROCK
GPU GRAPH ANALYTICS FRAMEWORK AT UC DAVIS

https://gunrock.github.io

UC DAVIS
UNIVERSITY OF CALIFORNIA

What is the Gunrock Library?

A CUDA-based graph processing library, aims for:

- Generality

  covers a broad range of graph algorithms

- Programmability

  makes it easy to implement graph algorithms

  extends to multi-GPUs as simple as possible

- Performance

  maintains good performance

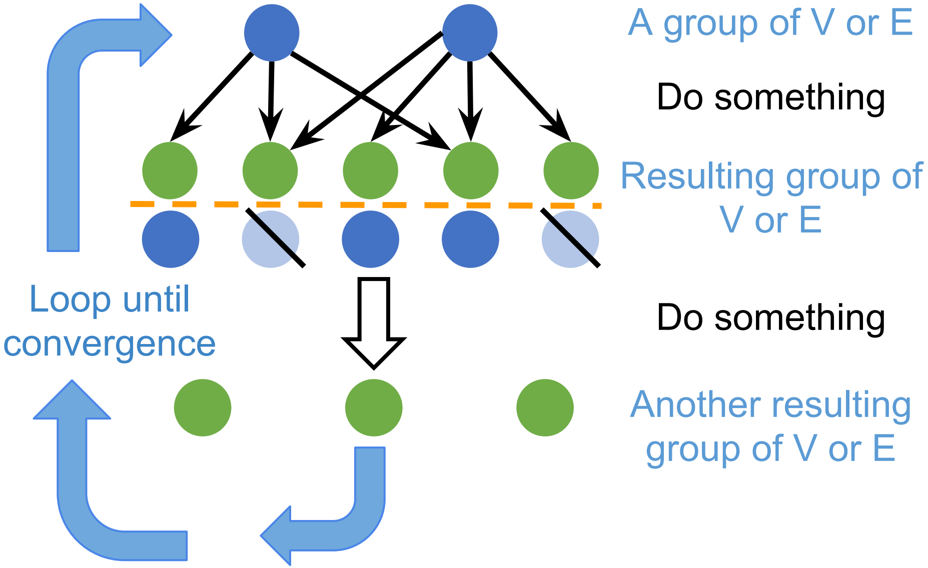
- Scalability

  fits in (very) limited GPU memory space

  performance scales when using more GPUs

Programming Model

A generic graph algorithm:



Data-centric abstraction

- Operations are defined on a group of vertices or edges $\stackrel{\text{def}}{=}$ a frontier

=> Operations = manipulations of frontiers

Bulk-synchronous programming

- Operations are done one by one, in order
- Within a single operation, computing on multiple elements can be done in parallel, without order

Yangzihao Wang, Yuechao Pan, Andrew Davidson, Yuduo Wu, Carl Yang, Leyuan Wang, Muhammad Osama, Chenshan Yuan, Weitang Liu, Andy T. Riffel, and John D. Owens. “Gunrock: GPU Graph Analytics”, TOPC 4(1), 2017

How to Write a Graph Primitive with Gunrock?

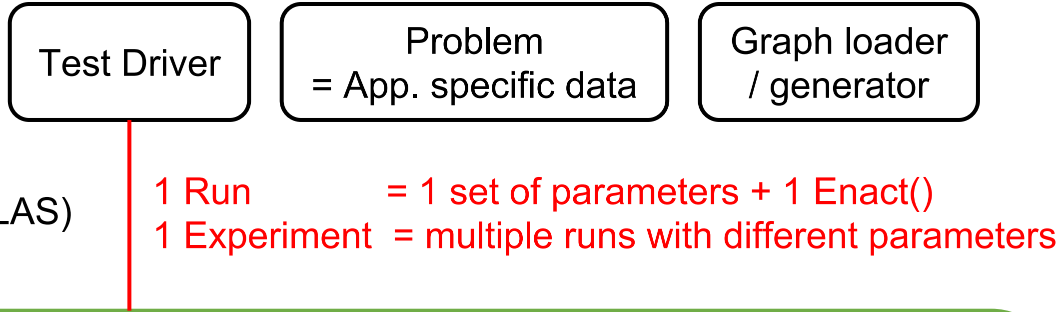
=> Section S8586, Writing Graph Primitives with Gunrock

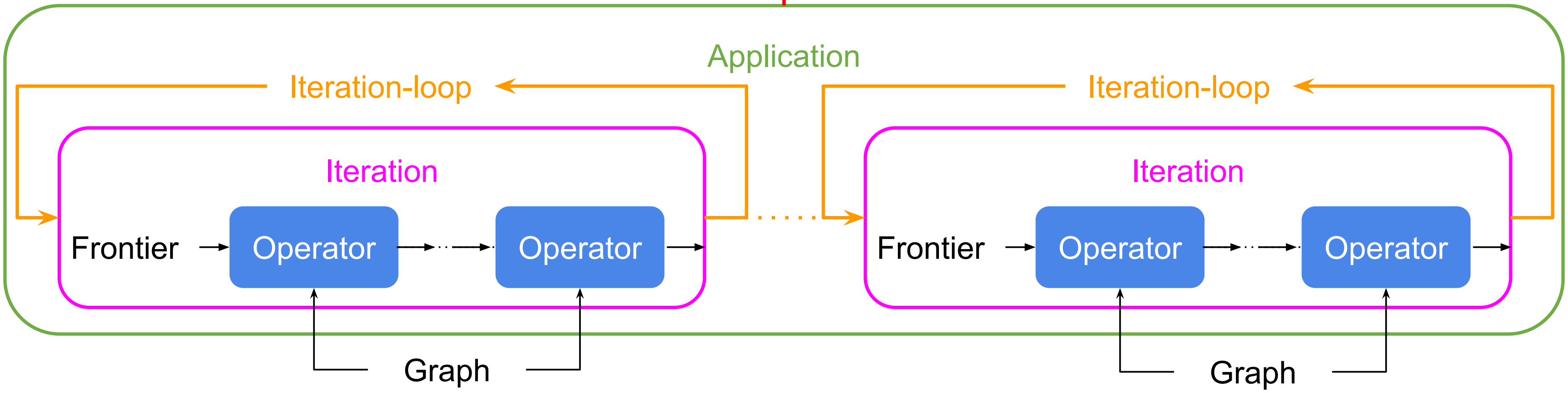
Key items for a graph primitive / app:

- Problem : data used by the algorithm
- Enactor : operations on the data
- App : higher level routines
- Test : CPU reference and result verification

New APIs

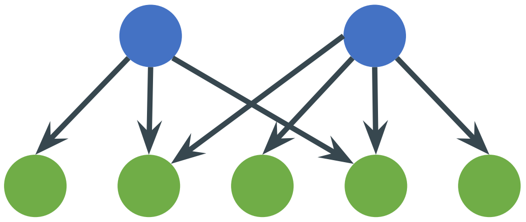
External interfaces
app. : callable from external
graph: external data structures (e.g. GoAI)
operator: calls external operators (e.g. GraphBLAS)





New APIs - oprtr::Advance

```
cudaError_t gunrock::oprtr::Advance
  <FLAG> (           // type (V2V, V2E, etc.) and
                     // option (Idempotence, Mark_Preds, ...)
  graph,             // graph representation
  input_frontier,    // input set of elements
  output_froniter,   // output set of elements
  oprtr_parameters,  // operator parameters (stream, etc.)
  advance_op,        // per-element advance lambda
  filter_op)         // per-element filter lambda (optional)
```



Advance:
visit neighbor lists

- Only 7 parameters, down from 20+
- Interface independent of graph representations
  => App. implementation isolated from graph representations
  => Operator will select a suitable implementation
     based on the given graph representation(s)
- Advance and filter operator share the same interface
- Lambda operator signatures are fixed for advance and filter
- Merged Cond. and Apply functors in older API

New APIs - oprtr::Advance

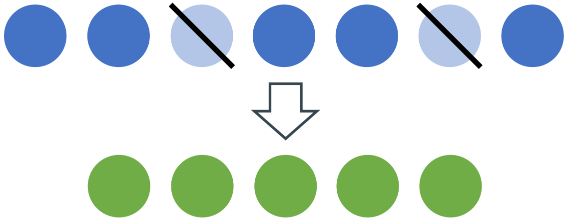
Example: SSSP advance

```
auto advance_op = [distances, weights, preds] __host__ __device__ (
  const VertexT &src, VertexT &dest, const SizeT &edge_id,
  const VertexT &input_item, const SizeT &input_pos, SizeT &output_pos) -> bool
{
  ValueT src_distance = Load<cub::LOAD_CG>(distances + src);
  ValueT edge_weight  = Load<cub::LOAD_CS>(weights + edge_id);
  ValueT new_distance = src_distance + edge_weight;
  if (new_distance >= atomicMin(distances + dest, new_distance))
    return false;
  Store(preds + dest, src);
  return true;
};

// Call the advance operator, using the advance operation
oprtr::Advance<oprtr::OprtrType_V2V>(
  graph.csr(), frontier.V_Q(), frontier.Next_V_Q(),
  oprtr_parameters, advance_op, filter_op);
```

New APIs - oprtr::Filter

```
cudaError_t gunrock::oprtr::Filter
  <FLAG> (           // type (V2V, V2E, etc.) and
                     // option (Idempotence, Mark_Preds, ...)
  graph,             // graph representation
  input_frontier,    // input set of elements
  output_froniter,   // output set of elements
  oprtr_parameters,  // operator parameters (stream, etc.)
  advance_op,        // per-element advance lambda (optional)
  filter_op)         // per-element filter lambda
```



Filter:
select and reorganize

Example: SSSP filter

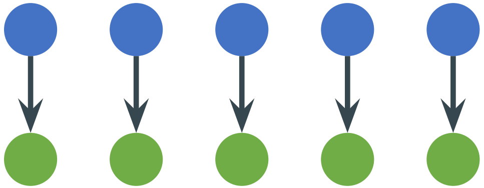
```
auto filter_op = [labels, iteration] __host__ __device__ (
  const VertexT &src, VertexT &dest, const SizeT &edge_id,
  const VertexT &input_item, const SizeT &input_pos,
  SizeT &output_pos) -> bool
{
  if (!util::isValid(dest)) return false;
  if (labels[dest] == iteration) return false;
  labels[dest] = iteration;
  return true;
};
```

```
// Call the filter operator, using the filter operation
oprtr::Filter<oprtr::OprtrType_V2V>(
  graph.csr(), frontier.V_Q(), frontier.Next_V_Q(),
  oprtr_parameters, filter_op);
```

New APIs - Compute Operators

```
cudaError_t gunrock::util::Array1D<...>::ForEach(
    compute_op,   // per-element computation lambda (w/o pos)
    num_elements, // number of elements
    target,  // where to perform the computation, CPU or GPU
    stream)       // cudaStream

cudaError_t gunrock::util::Array1D<...>::ForAll(
    compute_op,   // per-element computation lambda (with pos)
    num_elements, // number of elements
    target,       // where to perform the computation
    stream)       // cudaStream
```



Compute:
per-element operation

```
rank_next.ForEach([normalize, delta]__host__ __device__(ValueT &rank)
{
  rank = normalize ? (ValueT)0.0 : (ValueT)(1.0 - delta);
}, graph.nodes, target, stream);

degrees.ForAll([graph]
__host__ __device__(SizeT *degrees, const SizeT &pos)
{
  degrees[pos] = sub_graph.GetNeighborListLength(pos);
}, graph.nodes, target, stream);
```

- No need to write a kernel for simple operations
- Target independent

  => same code works on either CPU or GPU

<= Example: PR reset

New APIs - Graph Primitives / Apps

```
template <typename GraphT, ..., ProblemFlag FLAG>
struct Problem : ProblemBase<GraphT, FLAG>
{
  Problem(util::Parameters &parameters,
    ProblemFlag flag = Problem_None);

  cudaError_t Init(GraphT &graph,
       util::Location target);

  cudaError_t Reset(src, target);

  cudaError_t Extract(distances, preds, target);

  cudaError_t Release(target);
};

template <typename Problem, ...>
struct Enactor : public EnactorBase<...>
{
  Enactor();

  cudaError_t Init(Problem &problem, target);

  cudaError_t Reset(src, target);

  cudaError_t Enact(src, target);

  cudaError_t Release(target);
};
```

Template:	data types & option switches
util::Parameters:	running parameters
src, distances, preds:	algorithm specific inputs
target:	where to do the action
Init :	initialization, only do once
Reset:	data / status reset, do for each run
Enact:	invoke the algorithm implementation
Extract:	get back the results
Release:	clean-up

New APIs - External Interfaces

```
// @brief Entry of gunrock_sssp function
// @tparam     GraphT     Type of the graph
// @tparam     ValueT     Type of the distances
// @param[in]  parameters Excution parameters
// @param[in]  graph      Input graph
// @param[out] distances  Shortest distances from source
// @param[out] preds      Predecessors of each vertex
// \return     double     Accumulated elapsed times
template <
  typename GraphT,
  typename ValueT = typename GraphT::ValueT>
double gunrock_sssp(
  gunrock::util::Parameters &parameters,
  GraphT &graph,
  ValueT **distances,
  typename GraphT::VertexT **preds = NULL)
{...}
```

<= Using gunrock data types
Using raw data pointers =>

Able to take in graphs in GPU / CPU memory

Able to take in different graph representations

=> GoAI and other libraries

```
template <
    typename VertexT = int,
    typename SizeT   = int,
    typename GValueT = unsigned int,
    typename SSSPValueT = GValueT>
double sssp(
    const SizeT        num_nodes,
    const SizeT        num_edges,
    const SizeT       *row_offsets,
    const VertexT     *col_indices,
    const GValueT     *edge_values,
    const int          num_runs,
          VertexT     *sources,
    const bool         mark_pred,
          SSSPValueT **distances,
          VertexT    **preds = NULL)
{
    gunrock::util::Parameters parameters;
    GraphT graph;
    // prepare parameters & graph
    return gunrock_sssp(parameters, graph,
        Distances, preds);
}
```

New Features - Graph Representations

- Graph representation is isolated from most parts of Gunrock
  - Only operator implementations, graph generators & converters need to know the representation
  - Application level implementations does NOT need to know

  => External graph inputs (e.g. GoAI)

  => New graph representations (e.g. mutable graphs)

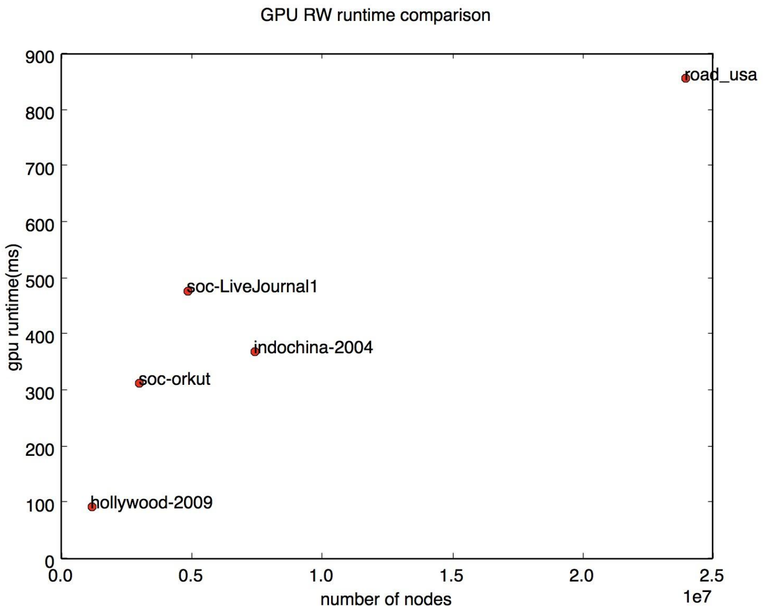
- Current status:
  - 3 basic representations: CSR, CSC, COO
  - SSSP on CSR, PR on COO

New Primitives - Random Walks

Find x random paths of given length y

- Algorithm

Q_0 <- {x randomly select source vertices}
Do y iterations:
    Q_1 <- {}
    For each vertex v in Q_0:
        Randomly select a neighbor u of v
        Put u in Q_1
    Q_0 <- Q_1



Running time of GPU random walk

New Primitives - Graph Coloring

Find the minimum number of vertex sets (i.e. colors), such that no edge has endpoints in the same set

- Algorithm

  Assign each vertex a random number
  Do until no more independent sets are found:
  Assign vertices with the minimum and the maximum number among neighbors to two sets, respectively

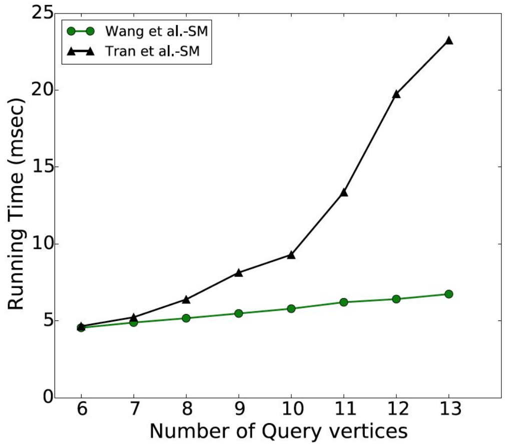
- Optimizations
  - use more colors per iteration,

    => maks the problem simpler, but loses accuracy in terms of amount of colors.

New Primitives - Subgraph Matching

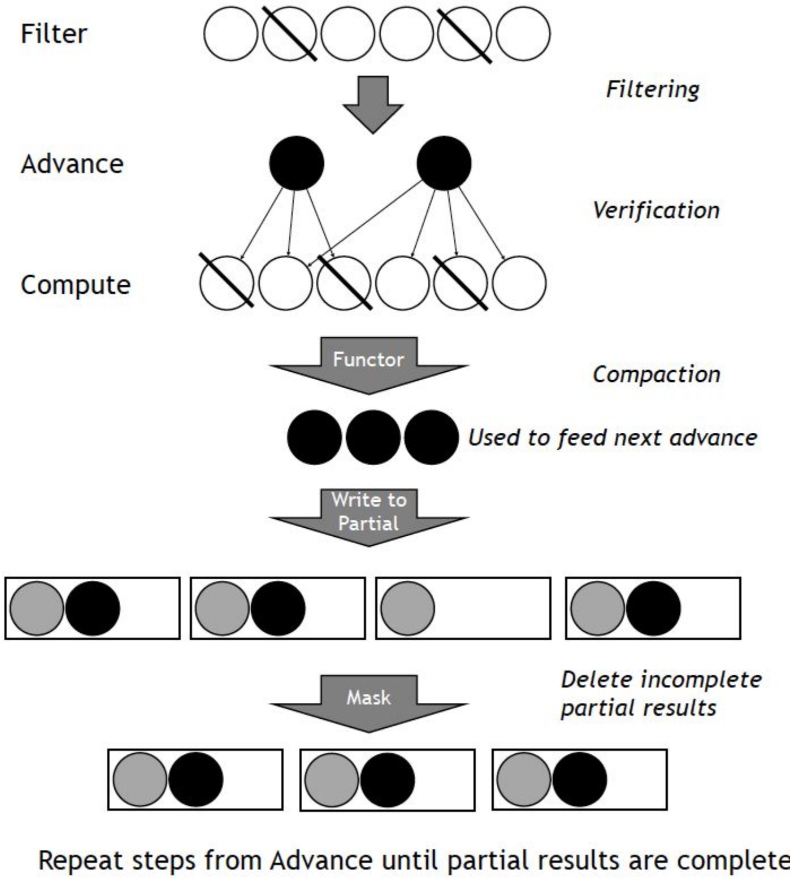
Find the occurrences of a small graph in a large graph
a.k.a. graph isomorphism, more info => poster P8290

- Design a BFS-based algorithm, to leverage Gunrock's high performance operators
- Use *k*-look-ahead, neighborhood encoding, and node equivalence, to generate fewer intermediate results



<= Speed-up of Enron dataset, compared to Tran et al.

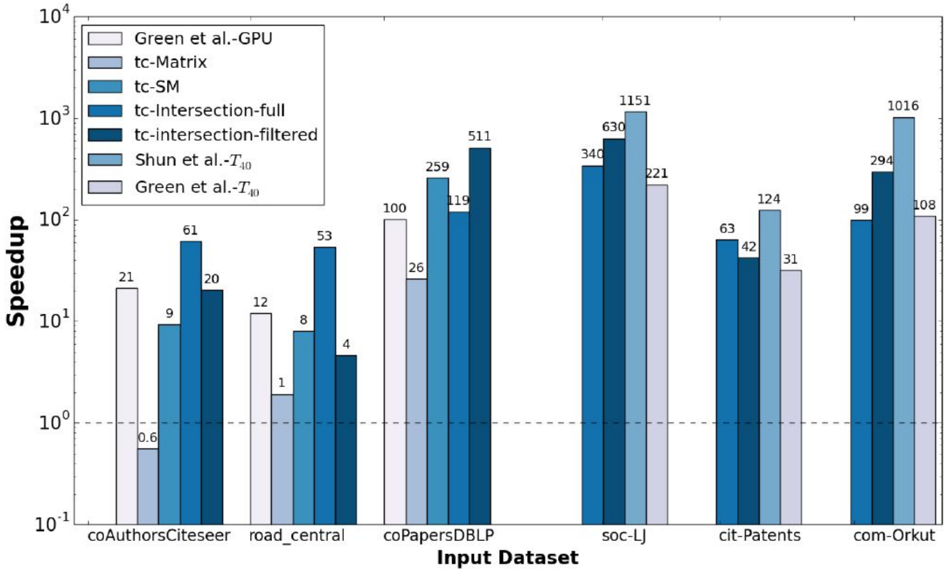
The algorithm =>



New Primitives - Triangle Counting

Find the number of triangles in a graph

- Four implementations
  - Batch set-intersection
  - Batch set-intersection + filtered edge list
  - Subgraph matching
  - Matrix-matrix multiplication and element-wise multiplication



Execution-time speedup for our four GPU implementations vs. others

Leyuan Wang, Yangzihao Wang, Carl Yang, and John D. Owens.
"A Comparative Study on Exact Triangle Counting Algorithms on the GPU", HPGP `16

Better Scaling

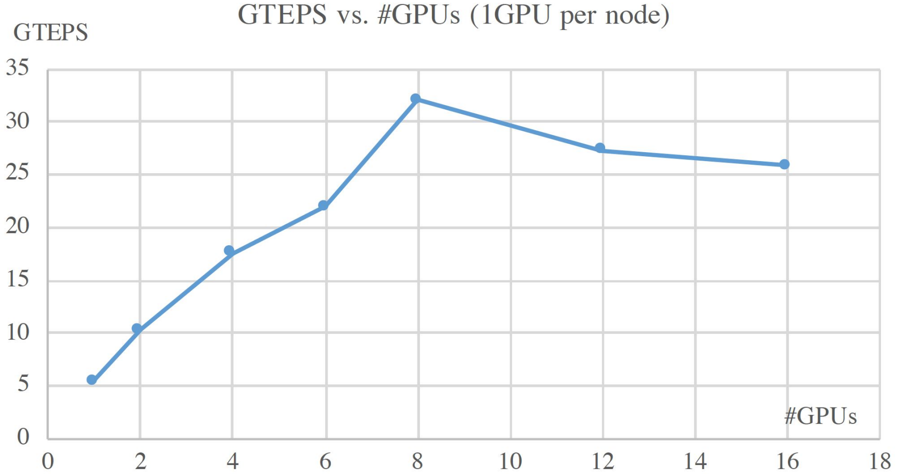
Scales beyond 1-node multiple GPUs

- Approach 1:
  - use 1-node multiple GPU framework*
  - differentiate local and remote peer GPUs
  - use MPI for communications between remote peers

- Tried with PageRank

  => workable

  => but can't scale to large number of GPUs



PageRank performance on undirected soc-orkut graph;
1 P100 + 2x 100 Gbps infiniband per node.

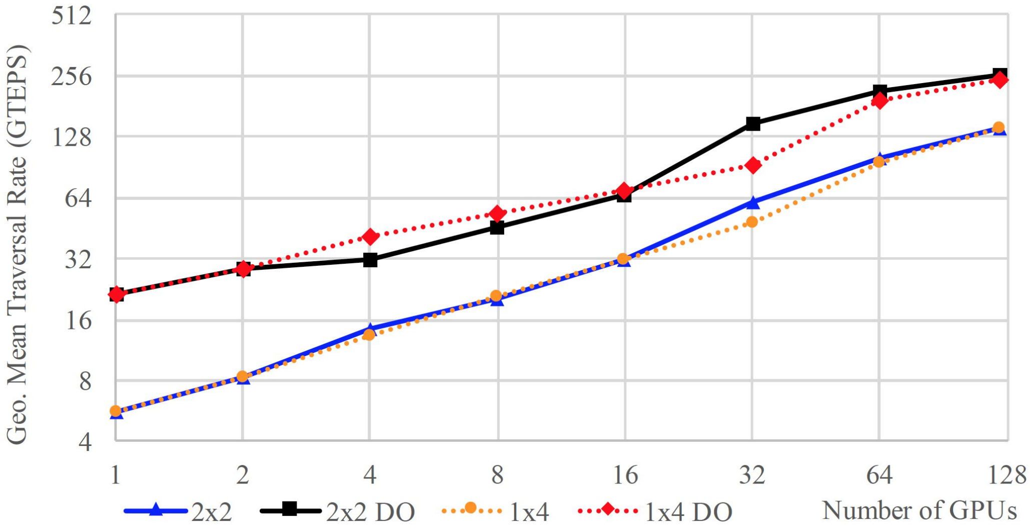
*Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and John D. Owens. "Multi-GPU Graph Analytics", IPDPS `17.

Better Scaling

Scales beyond 1-node multiple GPUs

- Approach 2:
  * Separate high and low degree vertices
  * Use (I)AllReduce for high-d vertices
  * Use p2p Isend / Irecv for low-d vertices
  * Use bit-masks for high-d vertices
  * Use different computation kernels
    for different edge types

- Tried with (DO)BFS
  => works well
  => bit-masks only works for BFS, other apps will have higher communication costs

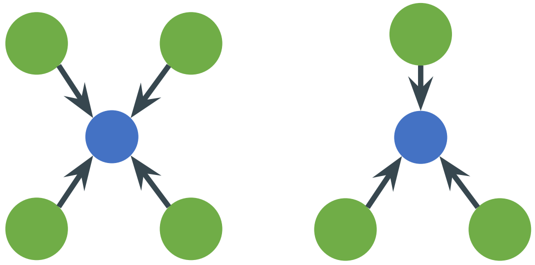


(DO)BFS weak scaling with a scale-26 RMAT graph per P100; 2x 100 Gbps + 4 P100s per node.

Yuechao Pan, Roger Pearce and John D. Owens.
“Salable Breadth-First Search on a GPU Cluster”, IPDPS ‘18

Upcoming - NeighborReduce

```
cudaError_t gunrock::oprtr::NeighborReduce
  <FLAG> (          // type (V2V, V2E, etc.) and
                    // option (Idempotence, Mark_Preds, ...)
  graph,            // graph representation
  input_frontier,   // input set of elements
  output_froniter,  // output set of elements
  oprtr_parameters, // operator parameters (stream, etc.)
  advance_op,       // per-element advance lambda
  reduce_op)        // neighborhood reduction lambda
```



NeighborReduce:
visit neighbors and reduce

```
auto reduce_op = [] __host__ __device__ (
  const VertexT &src, VertexT &dest, const SizeT &edge_id,
  const VertexT &input_item, const SizeT &input_pos,
  ValueT &val2, ValueT &val2, double probability) -> ValueT
{
  // return reduce(val1, val2);
}
```

Upcoming - More Graph Apps

- Clustering / Partitioning
- Abnormality Detections
- Asynchronous Graph Algorithms
- Cooperate with Machine Learning
- Mutable Graphs

Acknowledgments

The Gunrock team

All code contributors to the Gunrock library

NVIDIA

For hardware support, GPU cluster access, and all other support and discussion

Funding support:

- DARPA HIVE program
- DARPA XDATA program under US Army award W911QX-12-C-0059
- DARPA STTR awards D14PC00023 and D15PC00010
- NSF awards OAC-1740333 and CCF-1629657
- Adobe Data Science Research Award

Q & A

Q: How can I find Gunrock?
A: https://gunrock.github.io/

Q: Is it free and open?
A: Absolutely (under Apache License v2.0)

Q: Papers, slides, etc.?
A: http://gunrock.github.io/gunrock/doc/latest/index.html#Publications

Q: Requirements?
A: CUDA ≥ 8.0, GPU compute capability ≥ 3.0, Linux || Mac OS

Q: Language?
A: C/C++, with a simple wrapper to connect to Python

Q: … (continue)