(S8586)

Writing Graph Primitives with Gunrock

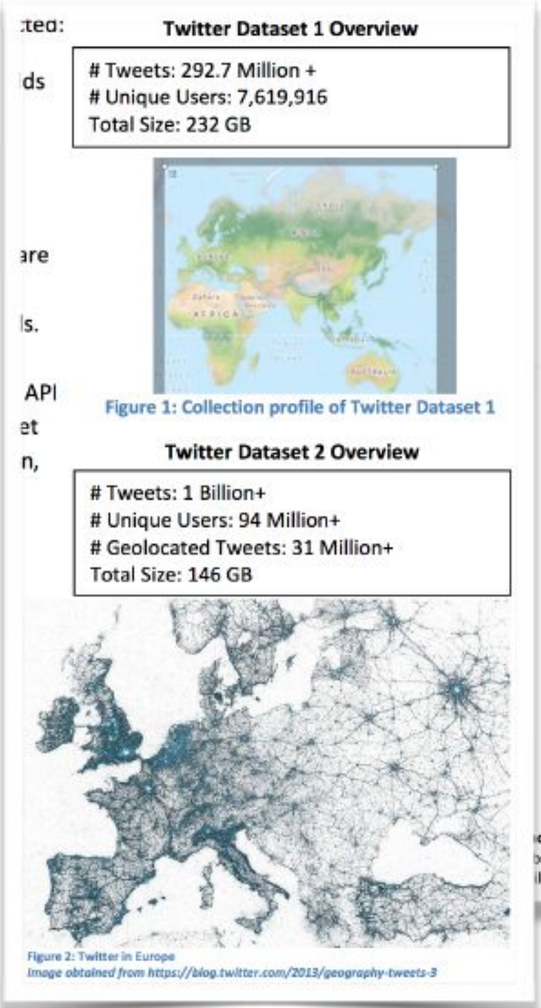
Muhammad Osama & Yuechao Pan
w/ The Gunrock Team

GUNROCK
GPU GRAPH ANALYTICS FRAMEWORK AT UC DAVIS

https://gunrock.github.io

UCDAVIS
UNIVERSITY OF CALIFORNIA

Why Graph Processing?



ted:

ds

are

ls.

API

et

n,

Twitter Dataset 1 Overview

# Tweets: 292.7 Million +
# Unique Users: 7,619,916
Total Size: 232 GB

Figure 1: Collection profile of Twitter Dataset 1

Twitter Dataset 2 Overview

# Tweets: 1 Billion+
# Unique Users: 94 Million+
# Geolocated Tweets: 31 Million+
Total Size: 146 GB

Figure 2: Twitter in Europe
Image obtained from https://blog.twitter.com/2013/geography-tweets-3

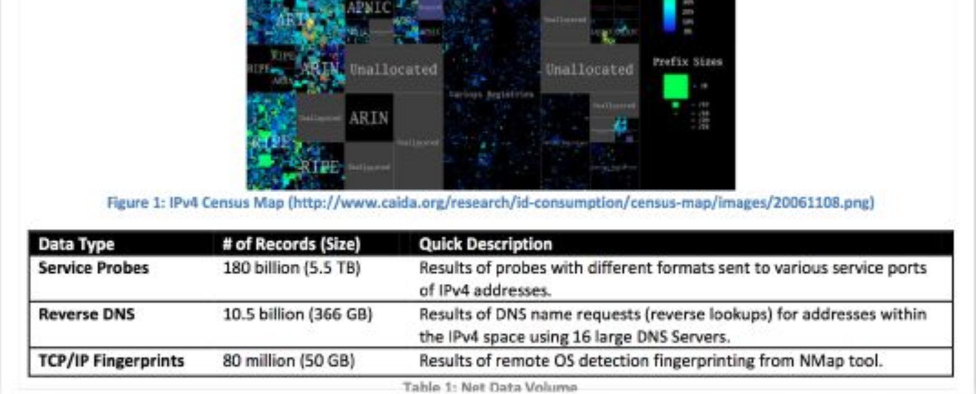


Figure 1: IPv4 Census Map (http://www.caida.org/research/id-consumption/census-map/images/20061108.png)

Data Type	# of Records (Size)	Quick Description
Service Probes	180 billion (5.5 TB)	Results of probes with different formats sent to various service ports of IPv4 addresses.
Reverse DNS	10.5 billion (366 GB)	Results of DNS name requests (reverse lookups) for addresses within the IPv4 space using 16 large DNS Servers.
TCP/IP Fingerprints	80 million (50 GB)	Results of remote OS detection fingerprinting from NMap tool.

Table 1: Net Data Volume

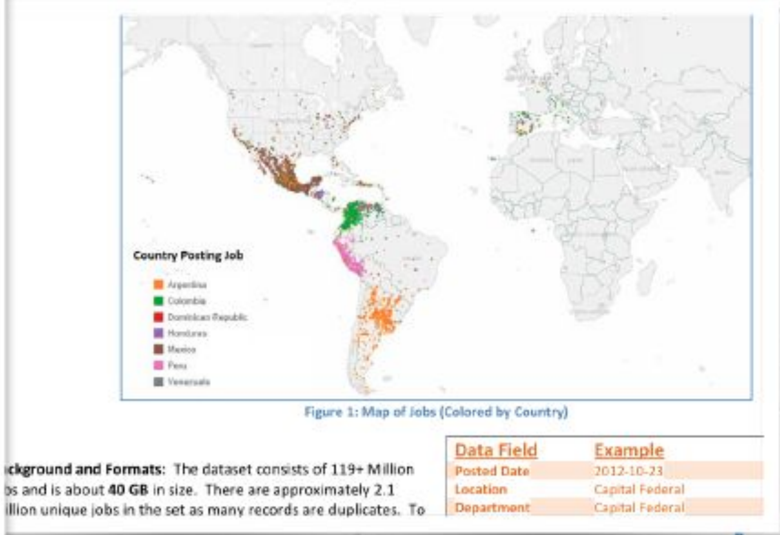


Figure 1: Map of Jobs (Colored by Country)

ckground and Formats: The dataset consists of 119+ Million bs and is about **40 GB** in size. There are approximately 2.1 illion unique jobs in the set as many records are duplicates. To

Data Field	Example
Posted Date	2012-10-23
Location	Capital Federal
Department	Capital Federal

Bitcoin Data Set Overview (May 15, 2013)

# Transactions:	15.8 Million+
# Edges:	37.4 Million +
# Senders:	5.4 Million+
# Receivers:	6.3 Million+
# Bitcoins Transacted:	1.4 Million +

Why use GPUs for Graph Processing?

Graphs

- Found everywhere
  - Road & social networks, web, etc.
- Require fast processing
  - Memory bandwidth, computing power and GOOD software
- Becoming very large
  - Billions of edges
- Irregular data access pattern and control flow
  - Limits performance and scalability

GPUs

- Found everywhere
  - Data center, desktops, mobiles, etc.
- Very powerful
  - High memory bandwidth (900 GBps) and computing power (15.7 TFlops)
- Limited memory size
  - 16 GB per NVIDIA V100
- Hard to program
  - Harder to optimize

What is Gunrock?

What is the Gunrock Library?

A CUDA-based graph processing library, aims for:

- Generality

  covers a broad range of graph algorithms

- Programmability

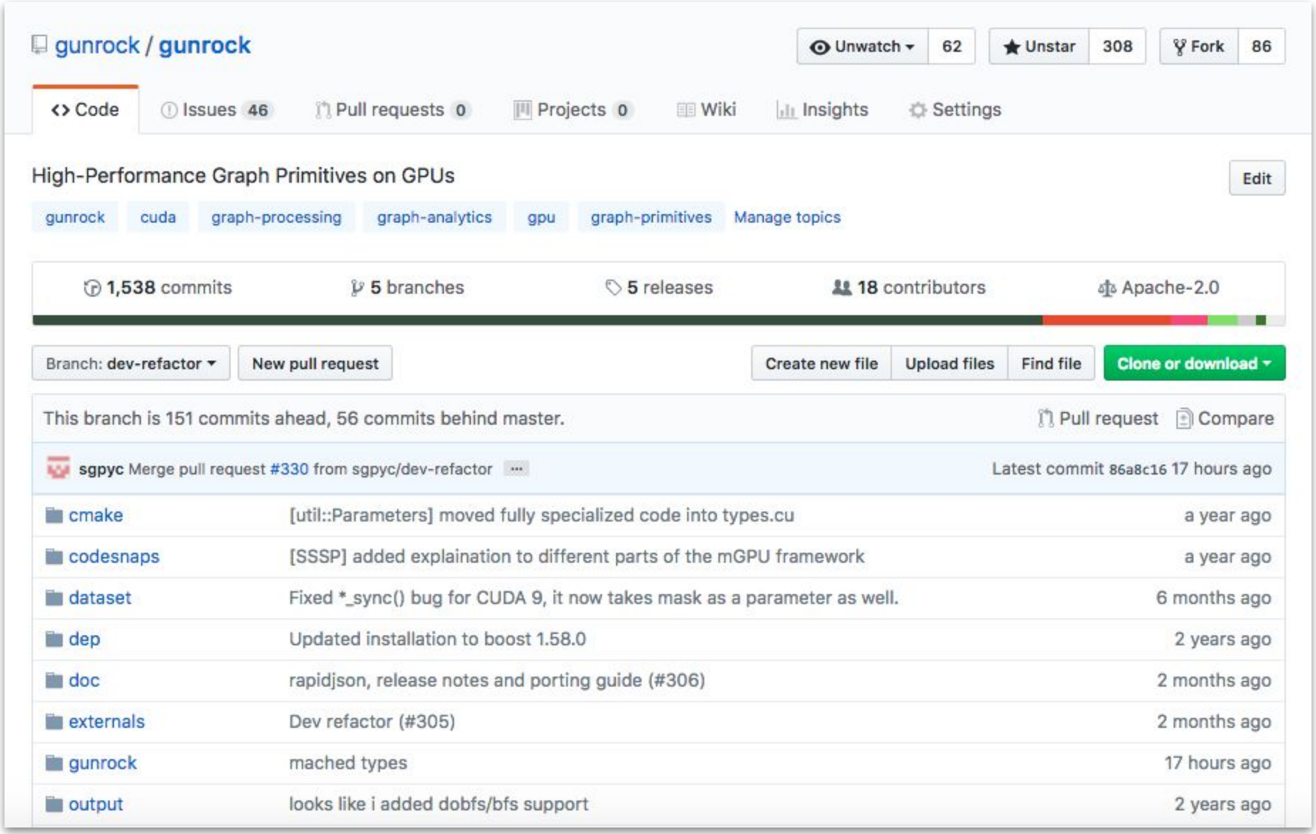
  Makes it easy to implement graph algorithms
  extends from 1-GPU to multi-GPUs as simple as possible

- Performance

  maintains good performance

- Scalability

  fits in (very) limited GPU memory space
  performance scales when using more GPUs



gunrock / gunrock

High-Performance Graph Primitives on GPUs

gunrock cuda graph-processing graph-analytics gpu graph-primitives Manage topics

1,538 commits | 5 branches | 5 releases | 18 contributors | Apache-2.0

This branch is 151 commits ahead, 56 commits behind master.

sgpyc Merge pull request #330 from sgpyc/dev-refactor ... Latest commit 86a8c16 17 hours ago

cmake	[util::Parameters] moved fully specialized code into types.cu	a year ago
codesnaps	[SSSP] added explaination to different parts of the mGPU framework	a year ago
dataset	Fixed *_sync() bug for CUDA 9, it now takes mask as a parameter as well.	6 months ago
dep	Updated installation to boost 1.58.0	2 years ago
doc	rapidjson, release notes and porting guide (#306)	2 months ago
externals	Dev refactor (#305)	2 months ago
gunrock	mached types	17 hours ago
output	looks like i added dobfs/bfs support	2 years ago

https://gunrock.github.io

How does Gunrock work?

Gunrock: Programming Model

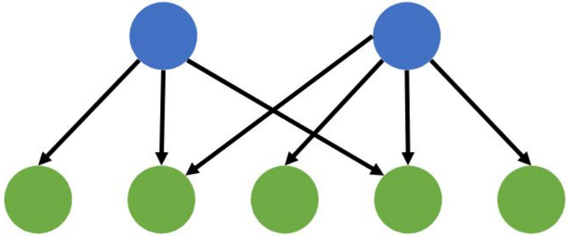
Data-centric abstraction

- A **frontier**; group of vertices or edges
- Manipulation of frontiers is an **operation**

Bulk-synchronous programming

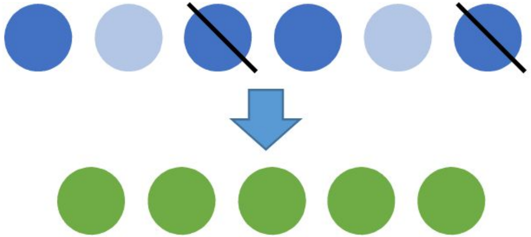
- Series of **parallel operations** separated by **global barriers**

Gunrock: Programming Model (cont.)



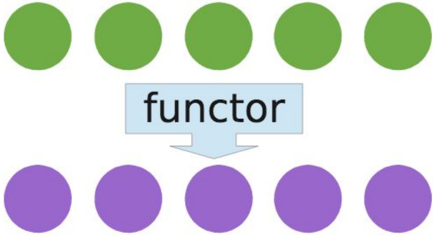
Advance

Generates new frontier by visiting the neighbors.



Filter

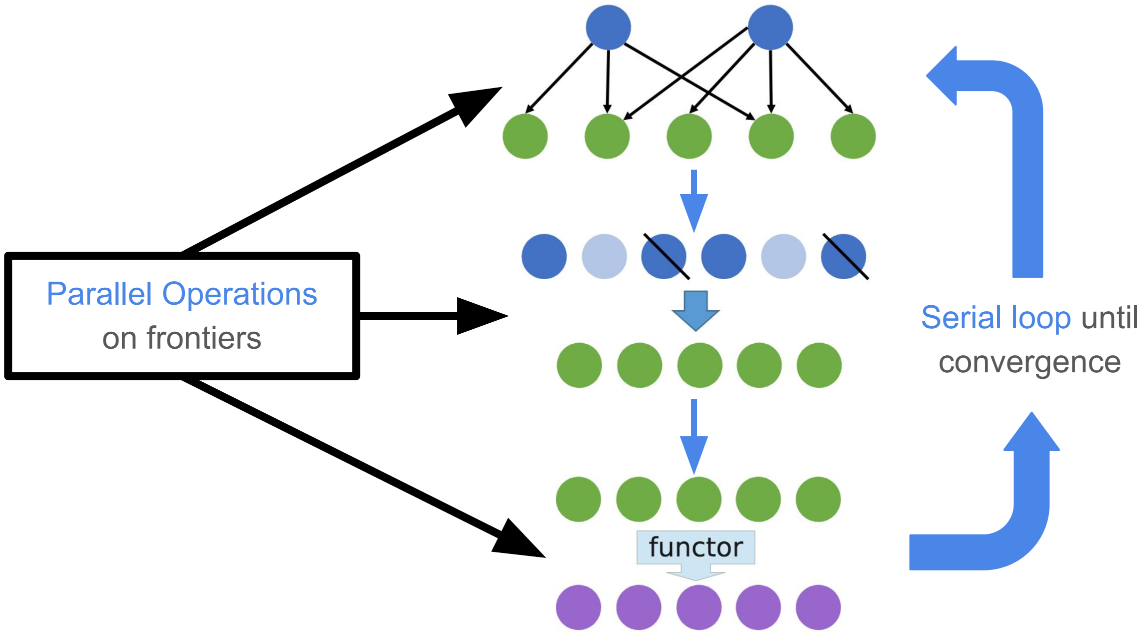
Chooses a **subset** of current frontier as the new front.



Compute

Applies a **compute** operation on all elements in its input front.

Gunrock: Programming Model (cont.)



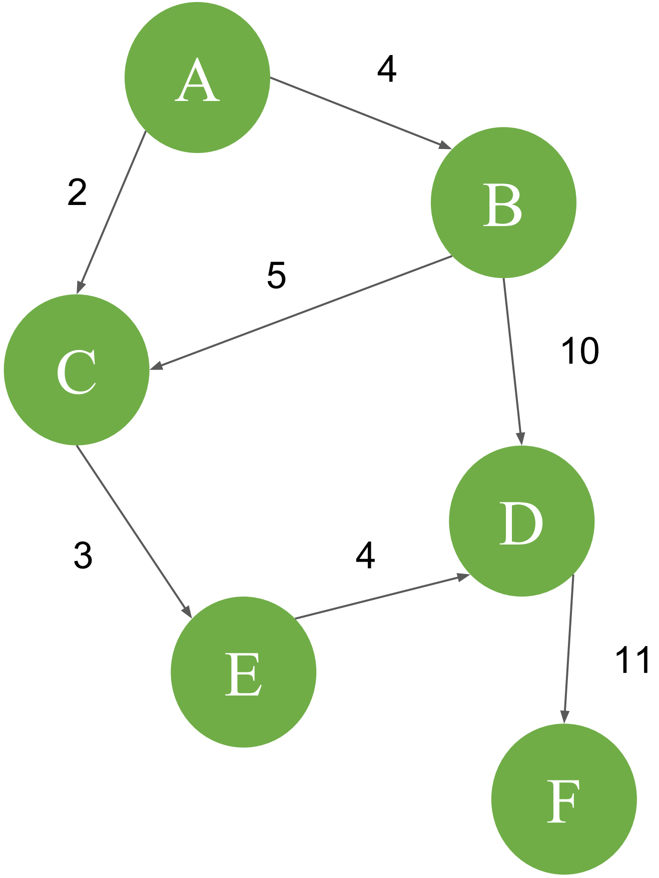
Parallel Operations on frontiers

Serial loop until convergence

How do you write a graph primitive using Gunrock?

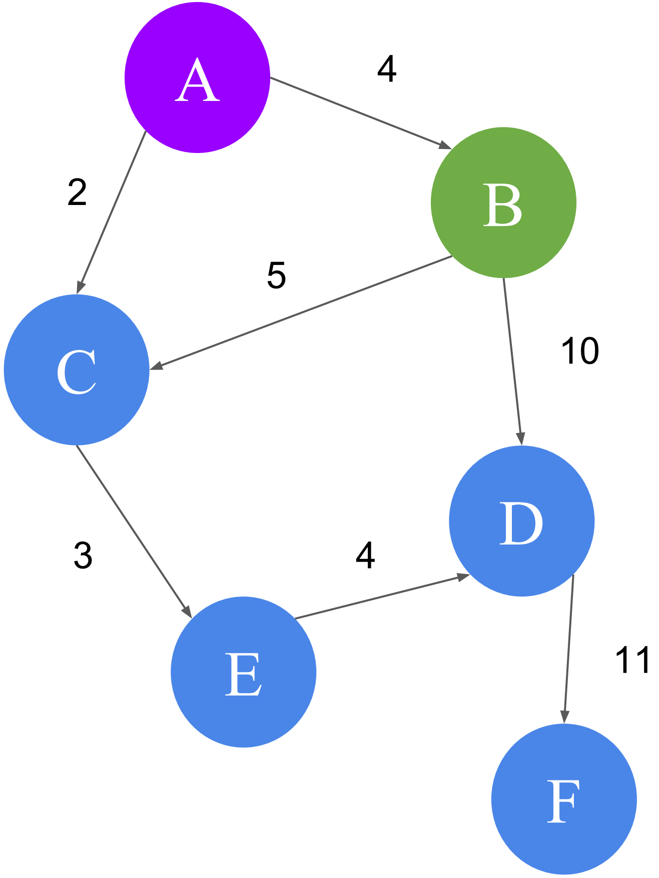
Example:
The Shortest Path Problem

Find a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized.



Example:
The Shortest Path Problem

- Let **A** be the source node.
- Shortest Path: (A, C, E, D, F)



Example:
Single Source Shortest Path

In SSSP we have to find **shortest paths** from a **source vertex v** to **all other vertices** in the graph.

Source: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html

Example: Gunrock’s SSSP Algorithm Pseudocode

```
function SSSP(Graph, source):
```

Reset

```
  for each vertex v in Graph:          // Initialization
    distances[v] := infinity; predecessors[v] := undefined
  distances[source] := 0               // Distance of source
  $Q_0$:= {source}                     // Set of vertices (frontier)
```

```
  while $Q_0$ is not empty:            // Iteration loop
    $Q_1$ := {}
```

Advance

```
    for each v in $Q_0$, do in parallel:
      for each edge e<v,u> of v, do in parallel:
        new_distance := distances[v] + weights[e]
        temp_distance := min(distances[u], new_distance) // atomic
        if (temp_distance > new_distance)
          predecessors[u] := v
          put u in $Q_1$
        else
          put invalid in $Q_1$
```

```
    $Q_0$ := {}
```

Filter

```
    for each u in $Q_1$, do in parallel:
      if (u is valid):
        put u in $Q_0$;
```

Directory Structure
gunrock/...

Gunrock (Repository) at https://gunrock.github.io

- **gunrock**
  - app
    - <primitive>
      - <primitive>_problem.cuh
      - <primitive>_enactor.cuh
      - <primitive>_test.cuh
      - <primitive>_app.cu
  - …
- **tests**
  - <primitive>
    - test_<primitive>.cu
  - …
- ...

gunrock/app/<primitive> directory

Our new gunrock primitive will be defined in this directory, along with the tests associated with it.

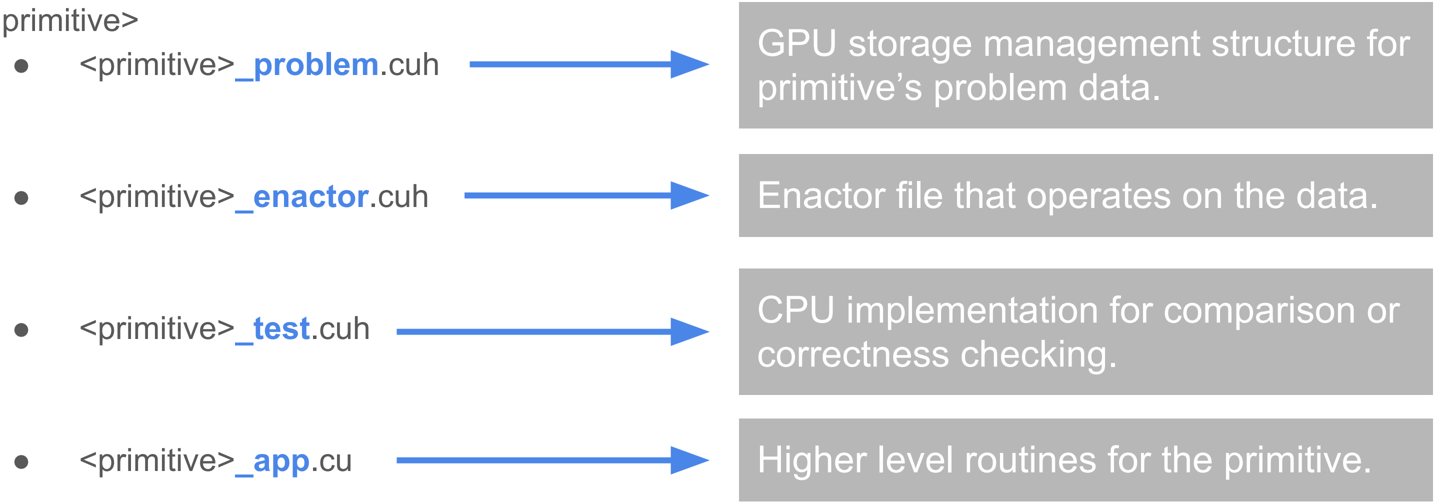
gunrock/tests/<primitive> directory

A test driver file to create an executable for your primitive.

Directory Structure
gunrock/app/<primitive>

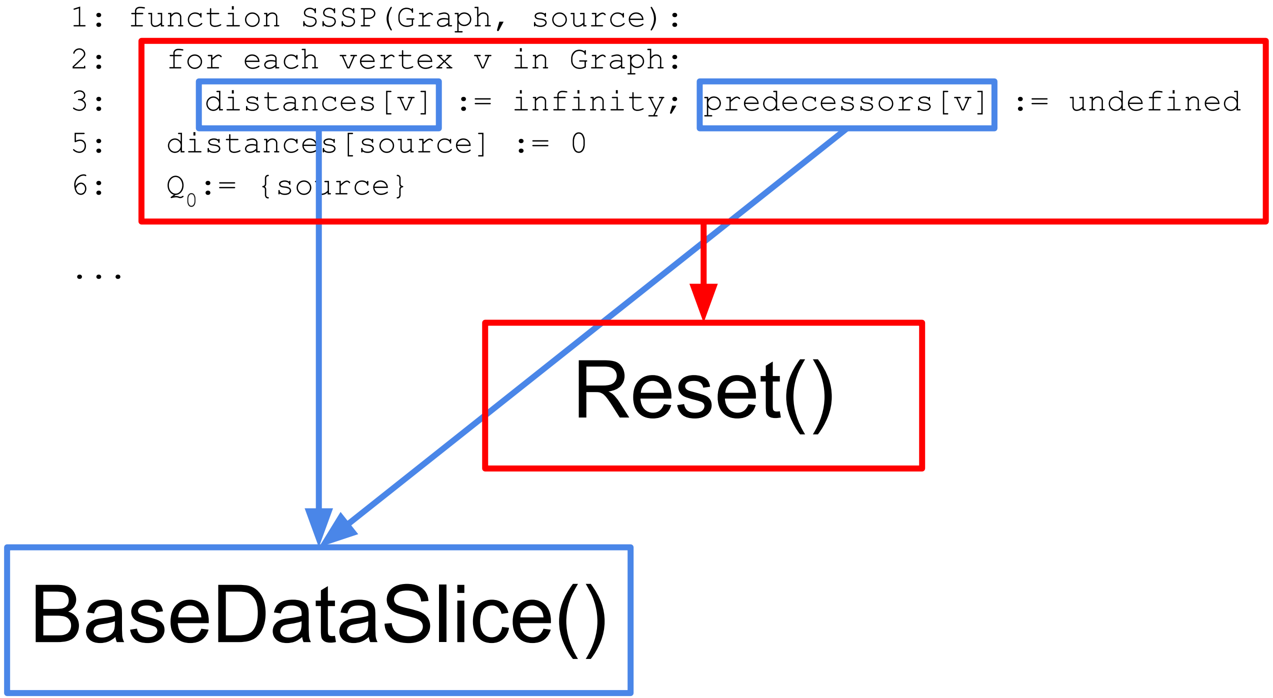
Gunrock (Repository) at https://gunrock.github.io

- **gunrock**
  - app
    - <primitive>



- <primitive>_**problem**.cuh → GPU storage management structure for primitive’s problem data.
- <primitive>_**enactor**.cuh → Enactor file that operates on the data.
- <primitive>_**test**.cuh → CPU implementation for comparison or correctness checking.
- <primitive>_**app**.cu → Higher level routines for the primitive.

Example: SSSP Problem



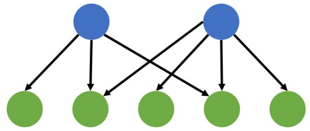
```
function SSSP(Graph, source):
    for each vertex v in Graph:
      distances[v] := infinity; predecessors[v] := undefined
    distances[source] := 0
    $Q_0$:= {source}

...
```

Reset()

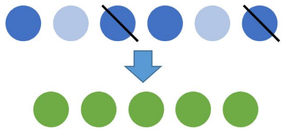
BaseDataSlice()

Example: SSSP Enactor

```
function SSSP(Graph, source):
...
  while $Q_0$ is not empty:
    $Q_1$ := {}
    for each v in $Q_0$, do in parallel:
      for each edge e<v,u> of v, do in parallel:
        new_distance := distances[v] + weights[e]
        temp_distance := min(distances[u], new_distance)
        if (temp_distance > new_distance)
          predecessors[u] := v
          put u in $Q_1$
        else
          put invalid in $Q_1$
    $Q_0$ := {}
    for each u in $Q_1$, do in parallel:
      if (u is valid):
        put u in $Q_0$;
```



Advance



Filter

Example:
SSSP Test

- Display solution
- Result validation
    - CPU Reference
    - Correctness checking

Example:
SSSP App

- Call to **Problem** and **Enactor**
- Recording Statistics
- External APIs

    Interface to C

    Interface to Python

Coding Tutorial

available @ https://tinyurl.com/gunrockgtc

Section S8594, Wednesday, Mar 28, 3:00–3:25 PM, Hilton Santa Clara

Latest Development of Gunrock: a Graph Processing Library on GPUs

- New APIs
  - Operators: Advance, filter, compute
  - External interfaces
  - Graph representations
- New Graph Applications
  - Subgraph matching
  - Graph coloring
  - Random walks, etc.

Acknowledgements

The Gunrock team

Gunrock code contributors

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

Questions?

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