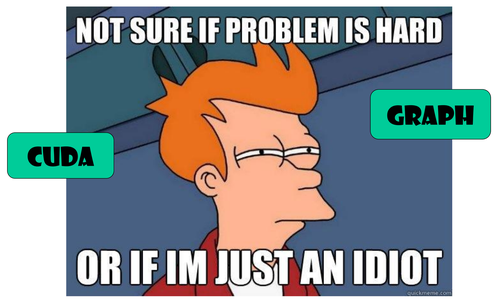
MPGraph: A High Level API for Fast Development of High Performance Graphic Analytics on GPUs

Zhisong Fu
SYSTAP, LLC

Motivation

- Scalable graph analytics are critical for a large range of application domains.
- CPU graph algorithms are known to scale poorly due to non-locality and limited memory bandwidth. Scalable, data-parallel graph analytics on many-core hardware is a fundamentally hard problem that goes beyond the current state of the art.
- The SIMT architecture used in GPUs constrains both the design and implementation of graph analytics algorithms and data structures, making **the development of such programs difficult and time-consuming.**



Introduction

- **MPGraph:** an open source library for high-performance graph processing.

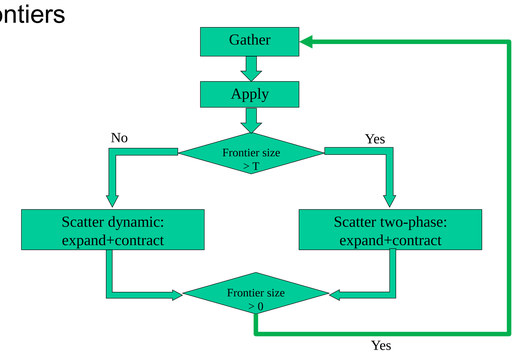
- **Program overview:**
  - **GAS model:** three conceptual phases of a vertex-program: Gather, Apply, and Scatter.

$$\Sigma \leftarrow \bigoplus_{v \in \mathbf{Nbr}[u]} g\left(D_u, D_{(u,v)}, D_v\right)$$

$$D_u^{\text{new}} \leftarrow a\left(D_u, \Sigma\right)$$

$$\forall v \in \mathbf{Nbr}[u]: \quad \left(D_{(u,v)}\right) \leftarrow s\left(D_u^{\text{new}}, D_{(u,v)}, D_v\right)$$

  - **Dynamic scheduling + two-phase decomposition:**
    - Dynamic scheduling scales poorly for large frontiers
    - Two-phase decomposition leads to much overhead for small frontiers



Highlights

- **High performance:** ultra-fast graph processing at up to 3 billion edges/second over 100s of millions of edges on a single GPU and is up to 294x faster than parallel CPU implementations.

- **Easy development:**
  - **Simple development process:**
    - Fill the Algorithm struct definition (shown below) that consists of 12 member structs and functions according to the three phases of the GAS model. Not all member structs and functions are necessary for an algorithm, and the definitions of the unnecessary ones are left blank.
    - Write the main function that invokes the graph processing processor of MPGraph.

  - **Simple API:** the Algorithm struct according to the GAS model

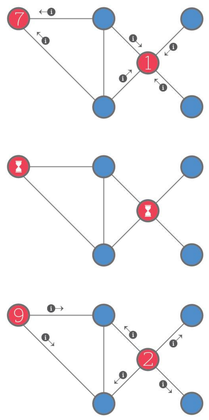
```
struct Algorithm
{
   struct VertexType {};     //States on vertices
   struct EdgeType{};        //States on edges

struct gather_edge{};
struct gather_sum{};
struct gather_vertex{};

struct apply{};
struct post_apply{};

struct expand_vertex{};
struct expand_edge{};
struct contract{};

   static void Initialize(){}   //Initialization of struct of array
   Static void memfree(){}  //Free all memory
};
```



- **AOS to SOA:** coalesced memory access

```
struct foo
{
    Type1 a;
    Type2 b;
    Type3 c;
} A[N];
```

```
struct foo
{
    Type1* a;
    Type2* b;
    Type3* c;
};
Struct foo A;
myMalloc(A.a, A.b. A.c, N);
```

Result

- CPU: Intel X5680, 12M Cache, 3.33 GHz, 6.40 GT/s Intel QPI GPU: Nvidia Tesla K20, peak performance 3.52 Tflops for single precision, 208 GB/sec memory bandwidth.
- The GraphLab results reported below are using a single CPU core. GraphLab actually slows down with more cores (negative scaling).
- The running times shown below are in milliseconds (ms).

Graphs:

Graph	#Vertices	#Edges
webbase	1,000,005	3,105,536
delaunay_n21	2,097,152	6,291,408
Bitcoin	6,297,539	28,143,065
Wiki	3,566,907	45,030,389
Kron	1,048,576	89,239,674

BFS:

Graph	GraphLab	MPGraph	Speedup
webbase	80.8	1.2	67.3
delaunay_n21	4715.8	24.5	193.0
Bitcoin	104201.6	354.3	294.1
Wiki	4880.7	51.0	95.7
Kron	686.9	53.3	12.9

SSSP:

Graph	GraphLab	MPGraph	Speedup
webbase	85.2	5.7	14.9
delaunay_n21	4692.3	98.3	47.7
Bitcoin	108288.5	502.5	215.5
Wiki	4871.1	118.1	41.2
Kron	626.7	104.2	6.0

Connected Components:

Graph	GraphLab	MPGraph	Speedup
webbase	863.5	41.6	20.8
delaunay_n21	7988.2	302.2	26.4
Bitcoin	18280.9	512.4	35.7
Wiki	4836.5	686.3	7.0
Kron	3310.0	309.2	10.7

Page Rank:

Graph	GraphLab	MPGraph	Speedup
webbase	731.3	97.0	7.5
delaunay_n21	1555.9	146.6	10.6
Bitcoin	57615.4	796.8	72.3
Wiki	10706.8	2610.3	4.1
Kron	4931.3	1671.3	3.0