Medusa

Simplified Graph Processing on GPUs

Motivation

- Graph processing algorithms are often inherently parallel
- GPUs consist of many processors running in parallel
- But… writing this code is *hard*

The Solution...

- Medusa is a C++ framework for graph processing on (multiple) GPUs
- Edge-Message-Vertex (EMV) programming model (BSP-like)
- Hides complexity of GPUs
- High programmability (expressive)

Related Work

- MTGL
    - Parallel graph library for multicore CPUs
- Pregel
    - Inspiration for the BSP model
- GraphLab2
    - Finer-grained like EMV model
- Green-Marl

Design Goals

- Programming interface:
    - High “programmability”
- System:
    - Fast

Programming Interface

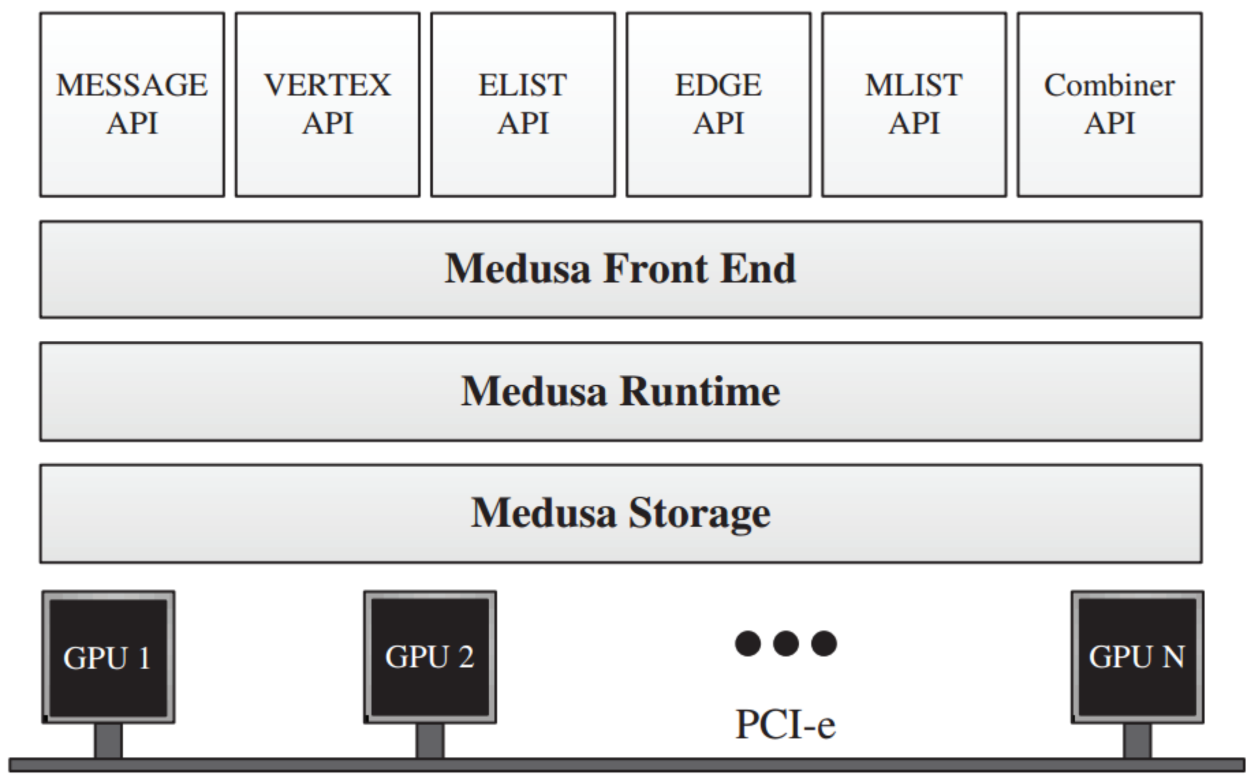
- User Defined APIs
    - Work on edges, messages, or vertices
    - The developer must provide implementations that conform to these interfaces
    - Where the algorithms themselves are specified
- System Provided APIs
    - Used to configure and run the algorithms

Example

One user defined function:

```
/* ELIST API */
struct SendRank {
__device__ void operator() (EdgeList el, Vertex v) {
    int edge_count = v.edge_count;
    float msg = v.rank/edge_count;
    for (int i = 0; i < edge_count; i ++)
        el[i].sendMsg(msg);
}
/* VERTEX API */
struct UpdateVertex {
__device__ void operator() (Vertex v, int super_step) {
   float msg_sum = v.combined_msg();
   vertex.rank = 0.15 + msg_sum*0.85;
}
...
```

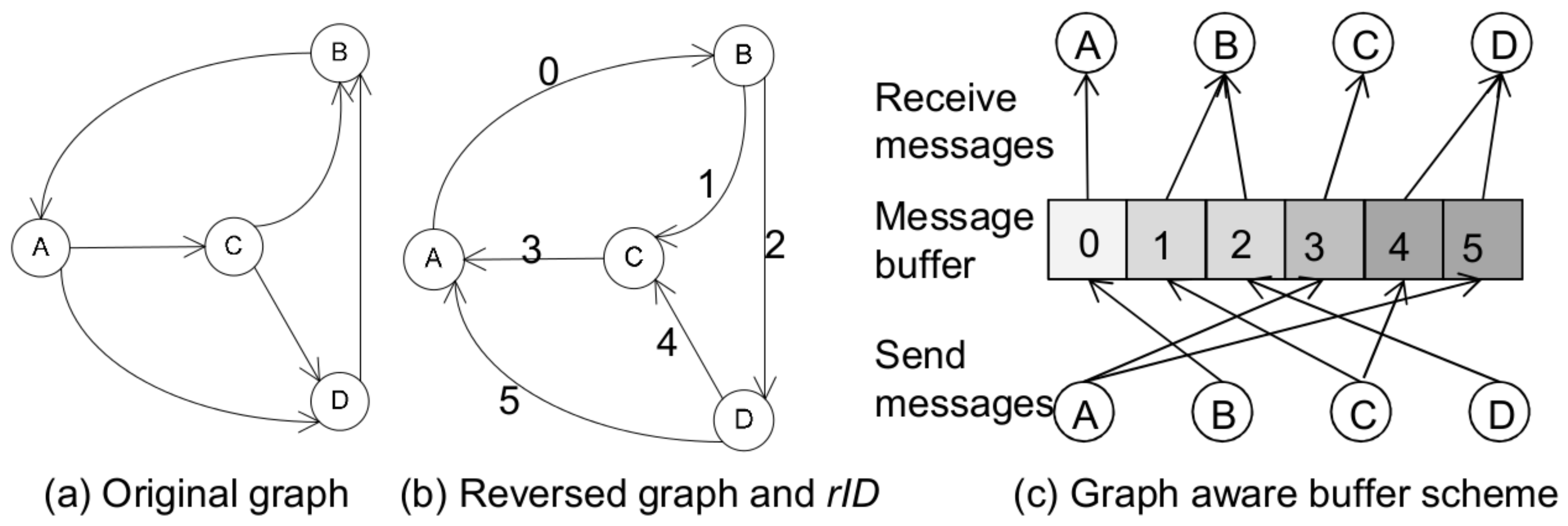
System Overview



Graph-Aware Buffer Scheme

- Messages temporarily build up in buffers
- Problem: statically or dynamically allocate buffer memory?
- Best of both worlds: size based on max messages that can be sent along an edge. Reverse graph array avoids need to group messages for processing

Graph-Aware Buffer Scheme



(a) Original graph (b) Reversed graph and *rID* (c) Graph aware buffer scheme

Support for Multiple GPUs

- Graph partitioned for each GPU with METIS
- Vertices with out-edges crossing partitions must be replicated
- Dominates processing time
- Optimisation: replicate vertices *n* hops from replicated head vertices.
    - Replication only after *n* iterations, but now more vertices to process

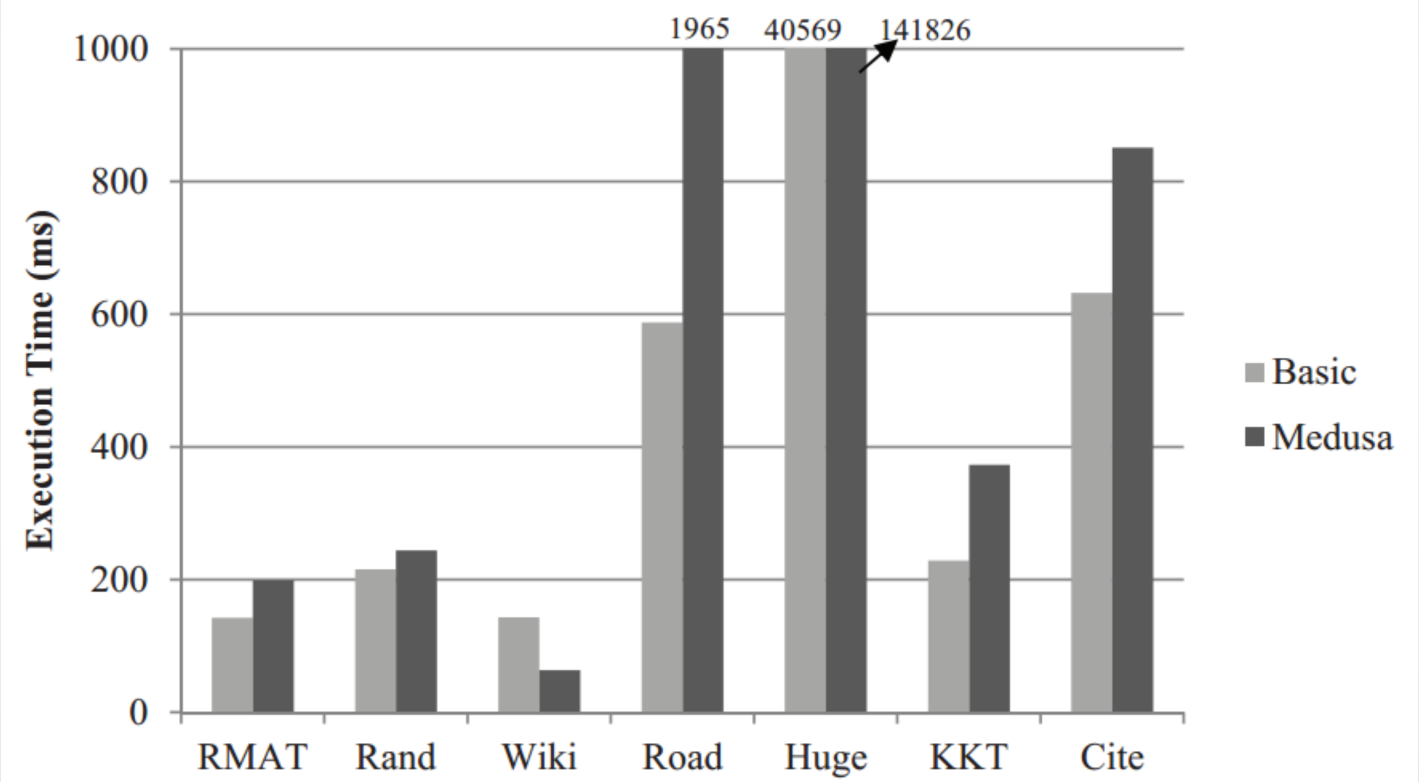
Evaluation

- Single workstation with 4 NVIDIA GPUs
- 8 different sparse graphs
  - real-world and synthetic
- Tested against 3 types of state-of-the-art manual GPU implementations
- Tested against MTGL framework running on a 12-core CPU

vs Tuned Manual Implementation

- Tested against two different state of the art manual implementations
- Tested using BFS
- Medusa performance better on all but one graph
- Manual implementation techniques may not be applicable to Medusa if they hurt programmability

Simple Manual Implementation SSSP



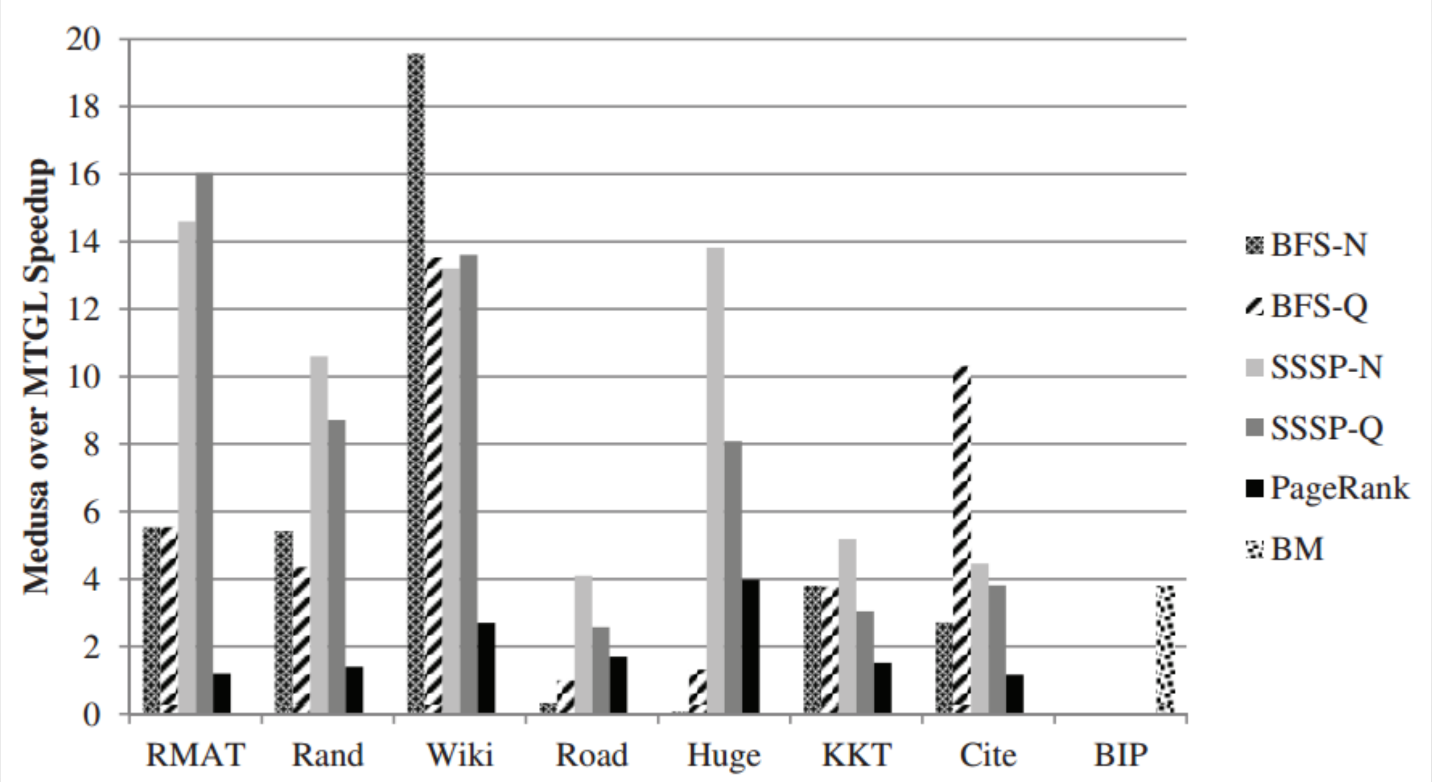
vs Contract-Expand BFS

Performance is variable depending on the graph when compare to Merril et al.’s recent work.

	Medusa	**Contract-Expand**	**Hybrid**
Huge	0.1	0.4	0.4
KKT	0.4	0.7	1.1
Cite	2.7	1.3	3.0

Traversed edges, higher is better

Comparison with CPU Framework



Limitations/Criticisms

- No sophisticated support for distributed systems, e.g. failure handling (unlike Pregel)
- Limited justification for maximising “programmability” (many popular systems are simpler)
- No evaluation with different numbers of GPUs and numbers of hops to replicate

Conclusion

- Time will tell with the programming model
- Performance really depends on the graph/algorithm
    - Great vs CPUs!
- Interesting to combine the concept with other systems