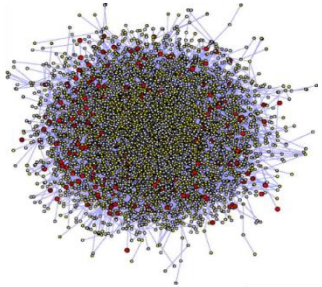
# Effective Evaluation of Betweenness Centrality on Multi-GPU systems

**Massimo Bernaschi**[1],

Giancarlo Carbone[2] Flavio Vella[2].

[1]IAC-National Research Council of Italy

[2]Sapienza University of Rome

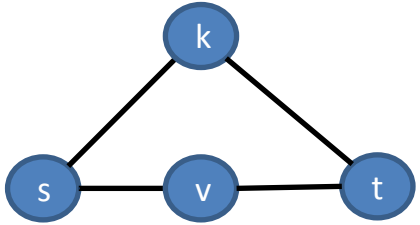


# Betweenness Centrality

A metrics to measure the influence or relevance of a node in a network

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- $\sigma_{st}$ is the number of shortest paths from **s** to **t**
- $\sigma_{st}(v)$ is the number of shortest paths from **s** to **t** passing through a vertex **v**




$\sigma_{st}$(v) = 1

$\sigma_{st}$ = 2

BC (v) = 0.5

# Betweenness Centrality

Measure of the influence or relevance of a node in a network

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- $\sigma_{st}$ is the number of shortest paths from **s** to **t**
- $\sigma_{st}(v)$ is the number of shortest paths from **s** to **t** passing through a vertex **v**

Time Complexity (Brandes' Algorithm)

- $O(nm)$ for unweighted graphs
- $O(nm + n^2 \log n)$ for weighted graphs

# Brandes' algorithm (2001)

1. Compute BFS starting from ***s***
    1. σ[v] for each ***v*** visited during the traversal of G
    2. Predecessors array
2. Sum all dependencies of ***s*** on predecessors array
3. Update BC score for each ***v*** ≠ ***s***

**Algorithm 1:** Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

# Brandes' algorithm (2001)

1. Compute BFS starting from ***s***
    1. σ[v] for each ***v*** visited during the traversal of G
    2. Predecessors array
2. Sum all dependencies of ***s*** on predecessors array
3. Update BC score for each $\boldsymbol{v} \neq \boldsymbol{s}$

Unfeasible for large-scale graphs!!!

**Algorithm 1:** Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

# GPU-based Brandes implementations

Exploiting GPU parallelism to improve the performance

Well-know problems due to Irregular Access Patterns
and unbalanced load distribution on traversal-based algorithms

Vertex vs Edge Parallelism

- Vertex-Parallelism: each thread is assigned to its own vertex
- Edge-Parallelism: each thread is in charge of a single edge
- Hybrid techniques (i.e., McLaughlin, A. and Bader, D. "Scalable and high performance betweenness centrality on the GPU [SC 2014])

# Multi-GPU-based Brandes implementations

"Scalable and high performance betweenness centrality on the GPU" [McLaughlin2014]

- Strategy
  - The graph is replicated among all computational nodes
  - Each root vertex can be processed independently
  - Use *MPI_Reduce* to update the bc score
- Advantages
  - Good scalability on graphs with one connected component

Main drawback

Data replication **limits the maximum size** of the graph!

# Algebraic Approach

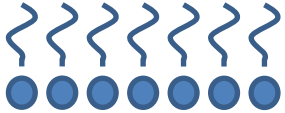
"The Combinatorial BLAS: Design, implementation, and applications" [Buluç2011].

- Strategy
  - Synchronous SpMM Multi-source Traversal based on Batch Algorithm [Robinson2008]
  - Graph partitioning based on a 2-D decomposition [Yoo2005]
- Drawback
  - No Heuristics
  - Different BFS-trees may have different depths

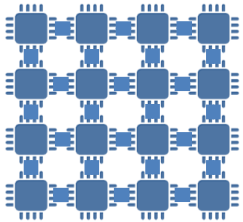
Load unbalancing intra- and inter-node on Real world graphs

# MGBC Parallel Distributed Strategy

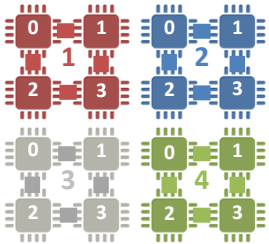
Multilevel parallelization of Brandes' algorithm + Heuristics



- Node-level parallelism
  - CUDA threads work on the same graph within one computing node



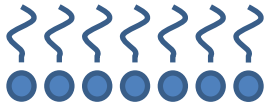
- Cluster-level parallelism
  - The graph is distributed among multiple computing nodes (each node owns a subset)

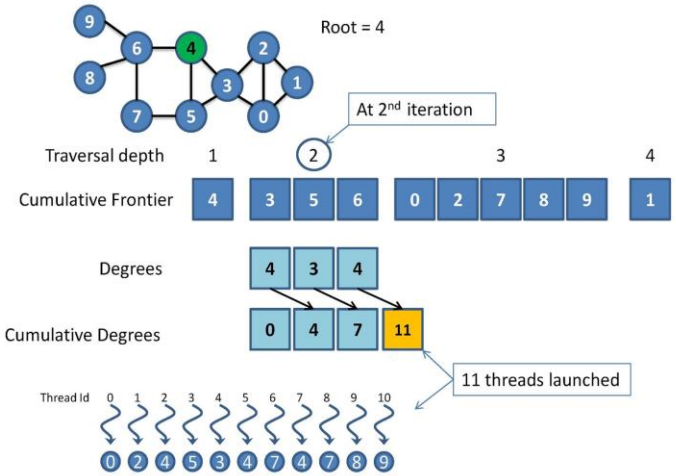



- Subcluster-parallelism
  - Computing nodes are grouped in subsets each working independently on its own replica of the same graph

# Node-level parallelism

- Distance BFS
  - Exploiting atomic-operations on Nvidia *Kepler* architecture
  - Data-thread mapping based on prefix sum and binary search.





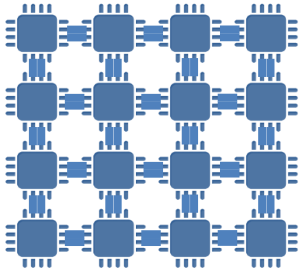

First Optimization!

Save extra-computation paid to have the regular access pattern

Avoiding prefix scan in dependency accumulation

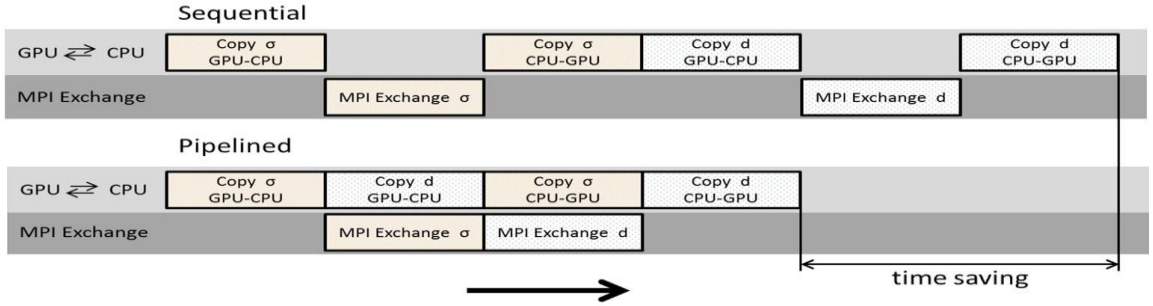
# Cluster-level parallelism

- 2-Dimensional partitioning
  - The graph is distributed across a 2-D mesh
  - Only $\sqrt{p}$ processors involved in the communication at time during traversal steps
- No Predecessors (contrary to Brandes) → No predecessor exchanging in distributed-system



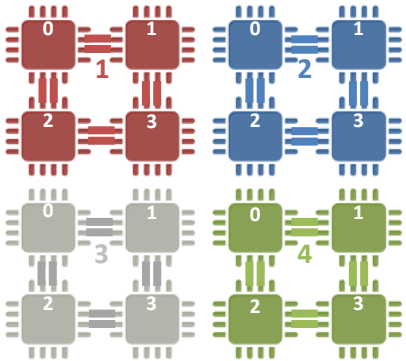
Second Optimization!

Pipelining CPU-GPU and MPI Communications



# Subcluster-level parallelism

- Multiple independent searches
  - a batch of vertices is assigned to each SC
  - a vertex at time inside a SC
- Configurable graph replication (fr) and graph distribution (fd) factors
  - the fr-replicas are assigned one for each SC
  - the graph is mapped onto each SC according to fd.
- MPI Communicators Hierarchy




p=16, fd=4

fr = p/d = 4

Advantage!

Each subcluster updates BC score only at the end of its own searches.
No synchronization among subclusters

# Experimental Setup

Piz Daint@CSCS

**#6 TOP500.org**

(http://www.top500.org/system/177824)

- Cray XC30 system with 5272 computing nodes
- Each node:
  - CPU Intel Xeon E5-2670 with 32GB of DDR3
  - GPU Nvidia Tesla K20x with 6 GB of DDR5
- SW Environment:
  - GCC 4.8.2
  - CUDA 6.5
  - Cray MPICH 6.2.2

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

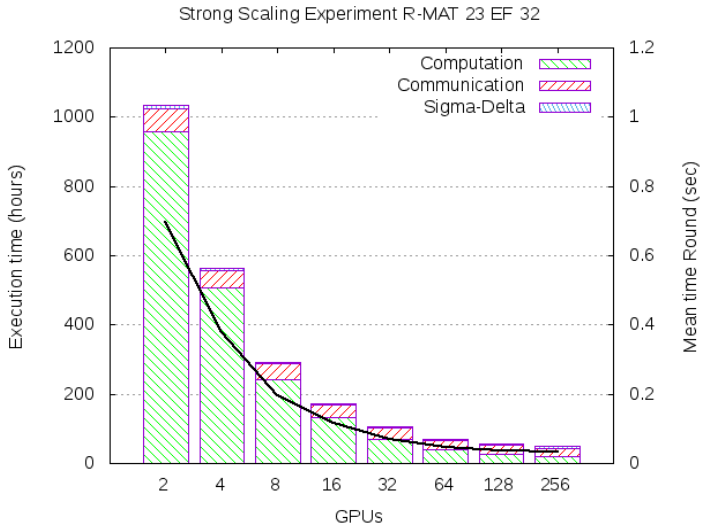
# Comparison Single-GPU

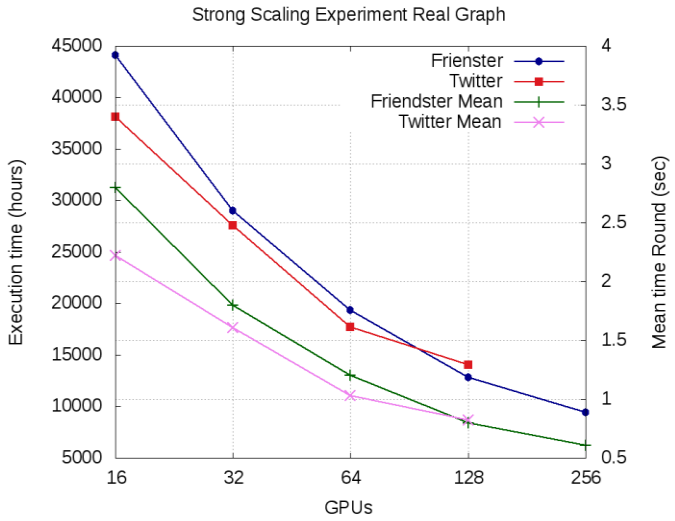
| SNAP Graph | SCALE | EF | MC | S1 | S2 | G | MGBC |
|---|---|---|---|---|---|---|---|
| RoadNet-CA | 20.91 | 1.41 | **0.067** | 0.371 | 0.184 | 0.298 | 0.085 |
| RoadNet-PA | 20.05 | 1.40 | **0.035** | 0.210 | 0.114 | 0.212 | 0.071 |
| com-Amazon | 18.35 | 2.76 | 0.008 | 0.009 | 0.006 | - | **0.005** |
| com-LJ | 21.93 | 8.67 | 0.210 | 0.143 | **0.084** | - | 0.100 |
| com-Orkut | 21.55 | 38.14 | 0.552 | 0.358 | **0.256** | - | 0.314 |

S = scale EF = Edge Factor $|V| = 2^{SCALE}$ and $|M| = EF \times 2^{SCALE}$ Avg. Time (sec)

Mc = McLaughlin et al "Scalable and high performance betweenness centrality on the GPU" [SC 2014].
S1 and S2 = Saryuce et al. "Betweenness centrality on GPUs and heterogeneous architectures " [GPGPU 2013].
G =Wang et al. " Gunrock: a high-performance graph processing library on the GPU " [PPoPP 2015].

# Strong Scaling

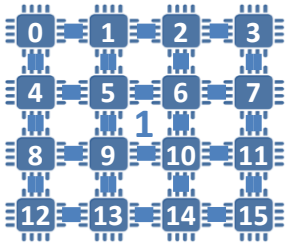



| G | SCALE | EF |
|---|---|---|
| R-MAT | 23 | 32 |
| Twitter | ~25 | ~35 |
| com-Friendster | ~26 | ~27 |

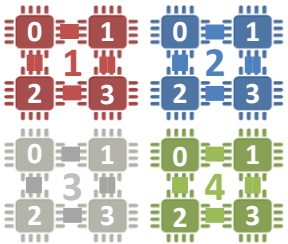
# Subcluster

**16 Processors**
**1 Cluster in a 4x4 Mesh**




| GPUs (p) | fd | fr | Time (hours) |
|---|---|---|---|
| 2 | 2x1 | 1 | ≈ 211 |
| 128 | 2x1 | 64 | ≈ 3.5 |
| 256 | 2x1 | 128 | **≈ 1.7** |
| 256 | 2x2 | 64 | ≈ 2.3 |

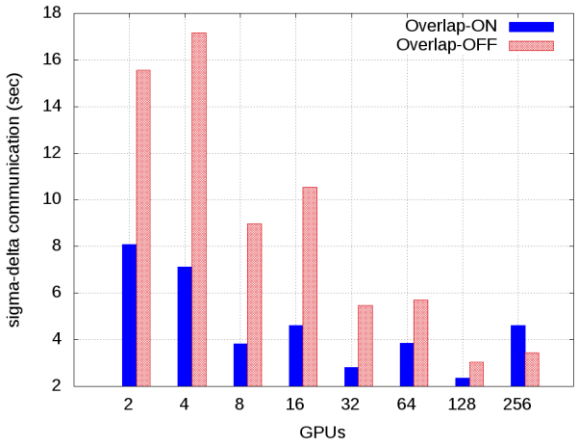
**16 Processors**
**4 sub-clusters in a 2x2 Mesh**



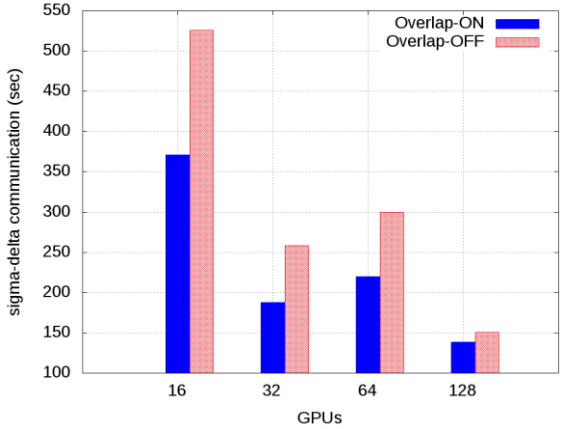

SNAP com-Orkut graph: Vertices ≈ 3E+06 – Edges ≈ 2E+08

# Optimizations Impact

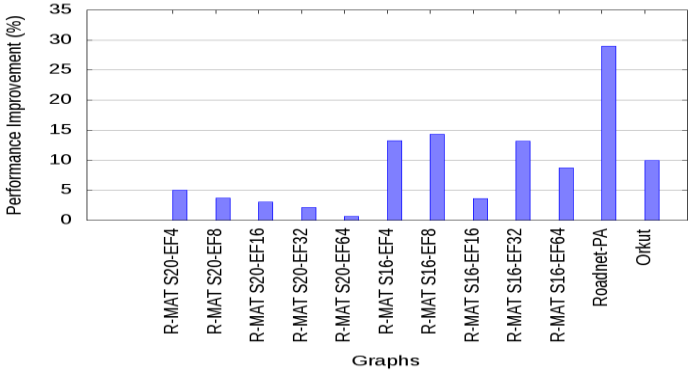
a) R-MAT S23 EF32



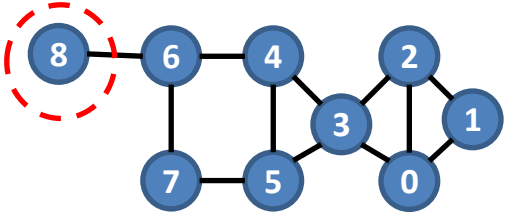

b) Twitter




c) Prefix-sum optimization

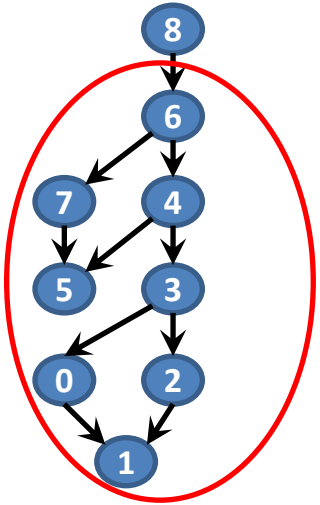


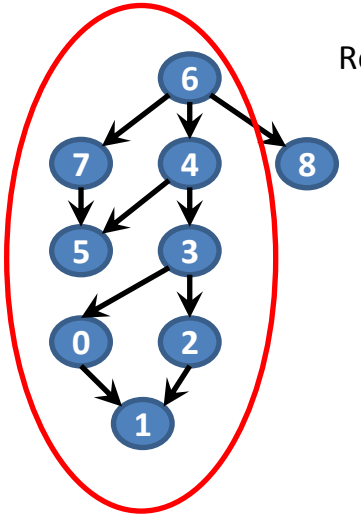
# 1-Degree Reduction




- Vertices with only one neighbour
- Removing 1-degree nodes from the graph (preprocessing)
- Reformulating the evaluation of the dependency
- First distributed implementation

Root = 8




Root = 6

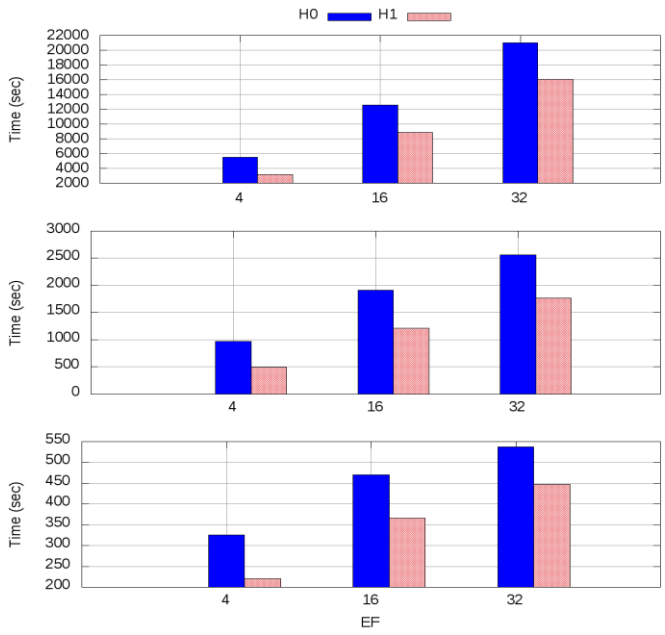
# 1-Degree Results

Benefits of 1-degree reduction

1. Avoid execution of BC calculation for 1-degree vertices
2. Reduce number of vertices to traverse

| Graph | 1-degree | Preprocessing (sec) | Speed-up |
|---|---|---|---|
| com-Youtube | 53% | 0.62 | 2.8x |
| roadNet-CA | 16% | 0.55 | 1.2x |
| com-DBLP | 14% | 0.19 | 1.2x |
| com-Amazon | 8% | 0.16 | 1.1x |
| R-MAT 20 E-16 | 13% | 1.2 | 1.3x |

*Source: Stanford Large Network Dataset Collection



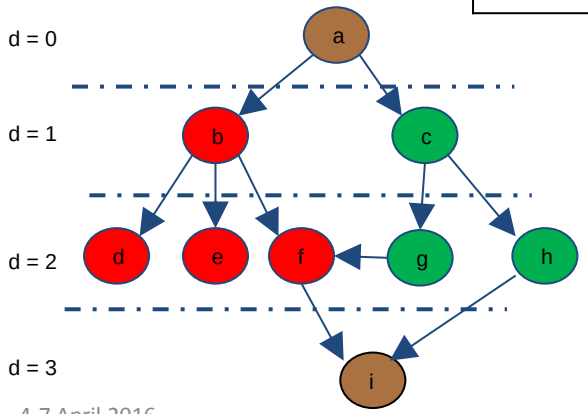

Impact of 1-degree: computation (top), communication (middle) and sigma-overlap (bottom) on R-MAT 20 and EF 4, 16 and 32.

# 2-Degree Heuristics

## Key Idea

Deriving BFS-tree of a 2-degree vertex from BFS-trees of its own neighbours

| Vertices | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| a | - | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| b | 1 | 0 | 2 | 1 | 1 | 1 | 2 | 3 | 2 |
| c | 1 | 2 | 0 | 3 | 3 | 2 | 1 | 1 | 2 |




let ***a*** be a 2-degree vertex and ***b***, ***c*** be its own neighbours. ***d*** is the distance from the source vertex

$$d_a(v) = min\{d_b(v), d_c(v)\} + 1$$

# DMF Algorithm

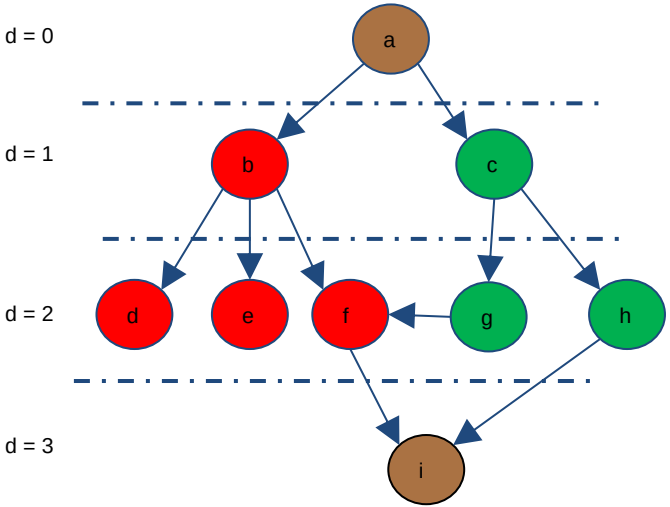
we can derive SSSP from ***a***

$$\sigma_a(v) = \begin{cases} \sigma_b(v) & \text{if } d_b(v) < d_c(v) \\ \sigma_c(v) & \text{if } d_b(v) > d_c(v) \\ \sigma_b(v) + \sigma_c(v) & \text{if } d_b(v) == d_c(v) \end{cases}$$

Dynamic Merging of Frontiers Algorithm

1. Compute the SSSP from ***b*** and ***c*** storing the number of shortest path and distance vectors of both
2. Compute level-by-level the Dependency Accumulation of ***b*** and **c** concurrently.
   The contributions of ***a*** for each visited vertex ***v*** is computed on-the-fly

# DMF Algorithm Example




Dependency of ***b*** {h} at d=3
$d_b(h) = 3$  $d_c(h) = 2$

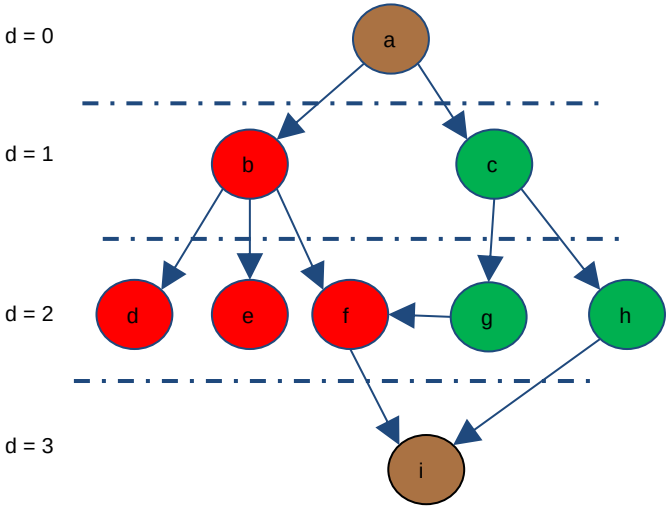
Vertex ***b*** does not compute the dependency of ***a***

Dependency of ***c*** {d,e} at d=3
$d_b(d) = 2$ $d_c(e) = 3$
$d_b(e) = 2$ $d_c(e) = 3$

Nothing to do for ***a***!!!!

| Vertices | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| a | - | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| b | 1 | 0 | 2 | 1 | 1 | 1 | 2 | 3 | 2 |
| c | 1 | 2 | 0 | 3 | 3 | 2 | 1 | 1 | 2 |

# DMF Algorithm Example




Dependency of ***b*** {3,7,9} at d=2
da(c) = 2 db(c) = 2
da(g) = 2 db(g) = 2
da(i) = 2 db(i) = 2

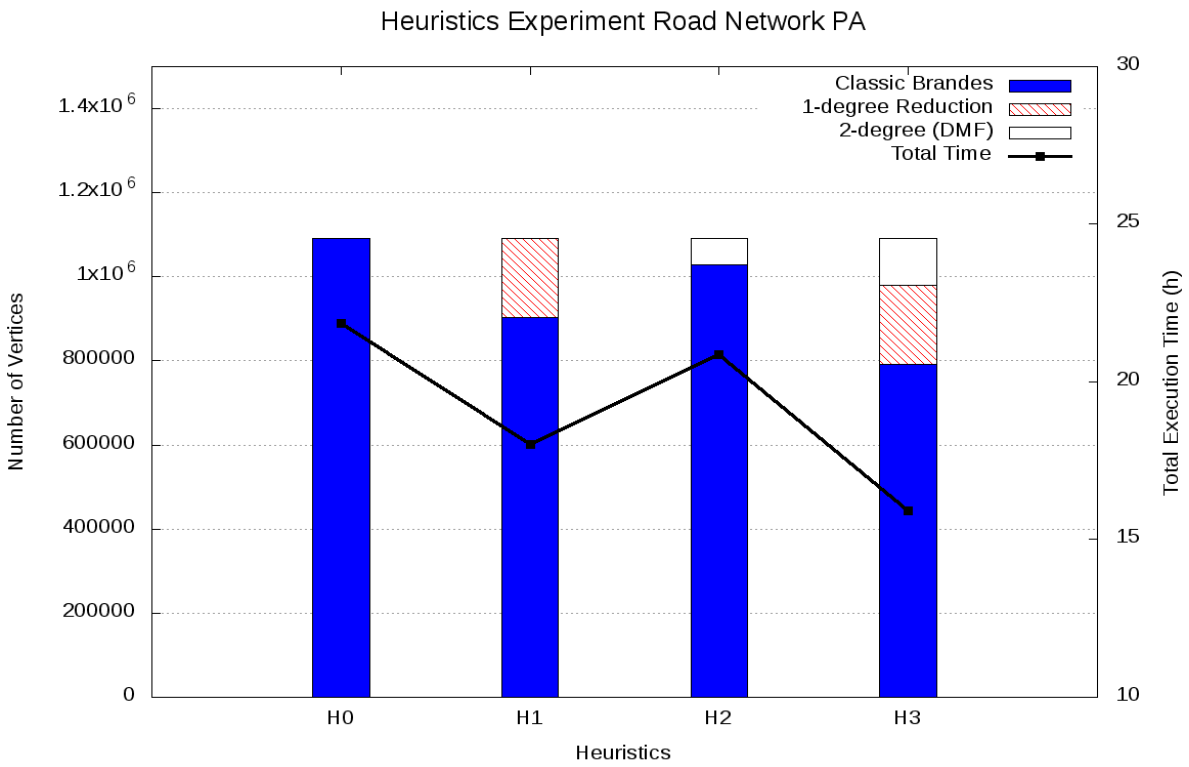
Vertex ***b*** computes the dependency of ***a*** on ***i*** (partially)

Dependency of ***c*** {b,f,h} at d=2
db(b)= 1 dc(b) = 2
db(f) = 1  dc(f) = 2
db(i) = 2  dc(i) = 2

***b*** and ***c*** contributes to Dependency of **a** !!!!

| Vertices | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| a | - | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| b | 1 | 0 | 2 | 1 | 1 | 1 | 2 | 3 | 2 |
| c | 1 | 2 | 0 | 3 | 3 | 2 | 1 | 1 | 2 |

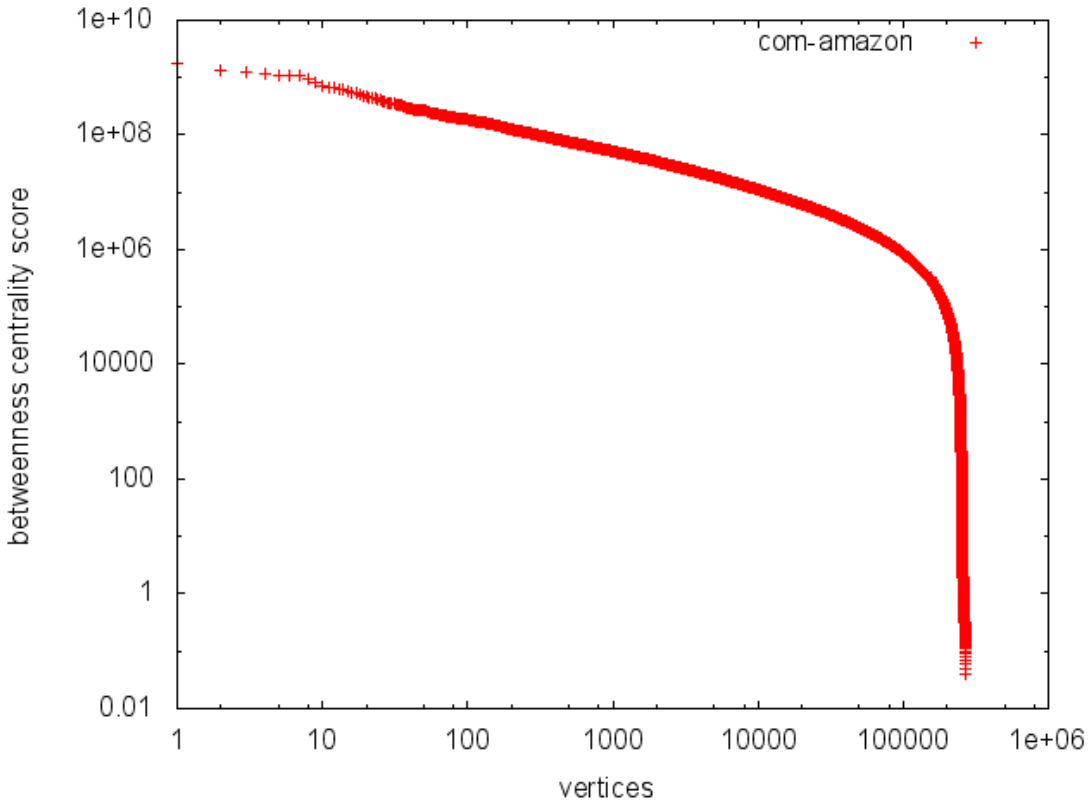
# Heuristics Results



# BC Analysis of a real-world graph

- Amazon product co-purchasing network

| Dataset statistics | |
|---|---|
| Nodes | 334863 |
| Edges | 925872 |
| Nodes in largest WCC | 334863 (1.000) |
| Edges in largest WCC | 925872 (1.000) |
| Nodes in largest SCC | 334863 (1.000) |
| Edges in largest SCC | 925872 (1.000) |
| Average clustering coefficient | 0.3967 |
| Number of triangles | 667129 |
| Fraction of closed triangles | 0.07925 |
| Diameter (longest shortest path) | 44 |
| 90-percentile effective diameter | 15 |



# Conclusion and future works

- Data-thread mapping approach is effective on graphs with different characteristics
- First 2-D fully distributed BC
  - Good scaling **up to 128 GPUs**
- Sub-clustering easily scale even with many GPUs
  - Linear scaling **up to 256 GPUs**
- Distributed 1-degree reduction heuristics
- New heuristics to solve 2-degree vertex on-the-fly
- MGBC computes real-world graph 234M < 2 hours
- Future works
  - Generalization of DFM
  - Heuristics on Algebraic Approach


# ???

massimo.bernaschi@cnr.it