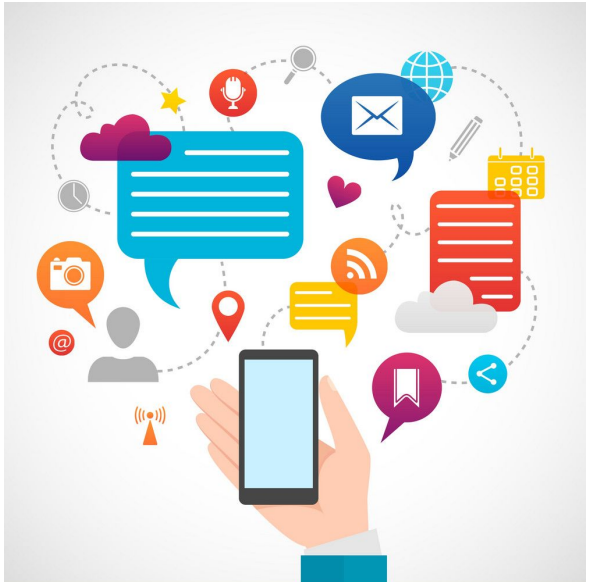
Parallel Betweenness Centrality Algorithm (ParBC)

Rui Qiu (rq2170)
Hao Zhou (hz2754)

Introduction

With the popularization of Smart Devices and social applications, a huge quantity of vast social network are available



Problem Definition and Dataset

To measure how central is a node…

Betweenness Centrality is formulated:

$$Betweenness(k) = \sum_{i \neq k \neq j} \left(\frac{\sigma_{i,j}(k)}{\sigma_{i,j}}\right)$$

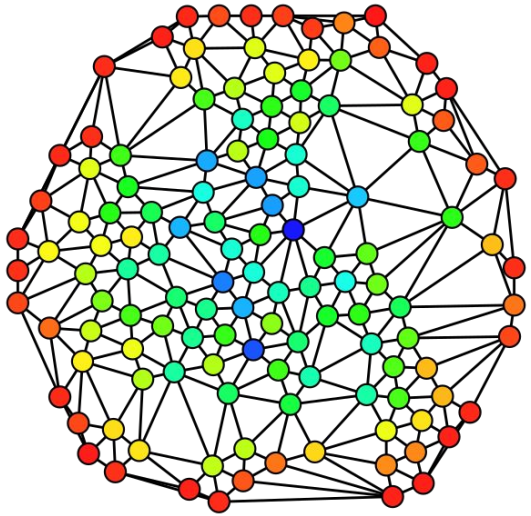
Algorithm Design

Originally proposed:

1. Calculate all pairwise shortest path length for all node pairs in G
2. Calculate betweenness Centrality. Inspired by Floyd–Warshall algorithm.

New proposed algorithm:

Direct calculations, easily implemented in pure parallel manner



Enhanced algorithm and parallel implementation

$$\sigma_{i,j} = \sum_{w \in pred_i(j)} \sigma_{i,w}$$

$$\delta_s(v) = \sum_{w \in Pred_s(v)} \sum_{t \in V} \delta_{s,t}(v, \{v, w\}) = \sum_{w \in Pred_s(v)} \left(\frac{\sigma_{s,v}}{\sigma_{s,w}} + \sum_{t \in V, t \neq w} \frac{\sigma_{s,v}}{\sigma_{s,w}} \frac{\sigma_{s,t}(w)}{\sigma_{s,t}}\right) = \sum_{w \in Pred_s(v)} \frac{\sigma_{s,v}}{\sigma_{s,w}} (1 + \delta_s(w))$$

$$Betweenness(v) = \sum_{s \in V} \delta_s(v)$$

Enhanced algorithm and parallel implementation

```
G ← the graph
Betweenness ← {n : 0|∀n ∈ G}
for s in G do
    pred ← {n : [ ]|∀n ∈ G}; dist ← {n : −1|∀n ∈ G}; sigma ← {n : 0|∀n ∈ G}; S ← [ ]
    dist[s] = 0; sigma[s] = 1
    queue ← [s]
    while queue is not empty do
        v = queue.get()
        S.append(v)
        for w in G.neighbours(v) do
            if dist[w] == −1 then
                dist[w] = dist[v] + 1
                queue.put(w)
            end if
            if dist[w] == dist[v] + 1 then
                sigma[w]+ = sigma[v]
                pred[w].append(v)
            end if
        end for
        delta ← {n : 0|∀n ∈ G}
        for w in reverse(S) do
            for v in pred[w] do
                delta[v]+ = sigma[v]/sigma[w] ∗ (1 + delta[w])
                if w ≠ s then
                    Betweenness[w]+ = delta[w]/2
```

Evaluation (correctness)

$$< 10^{-13}$$

Evaluation (efficiency)

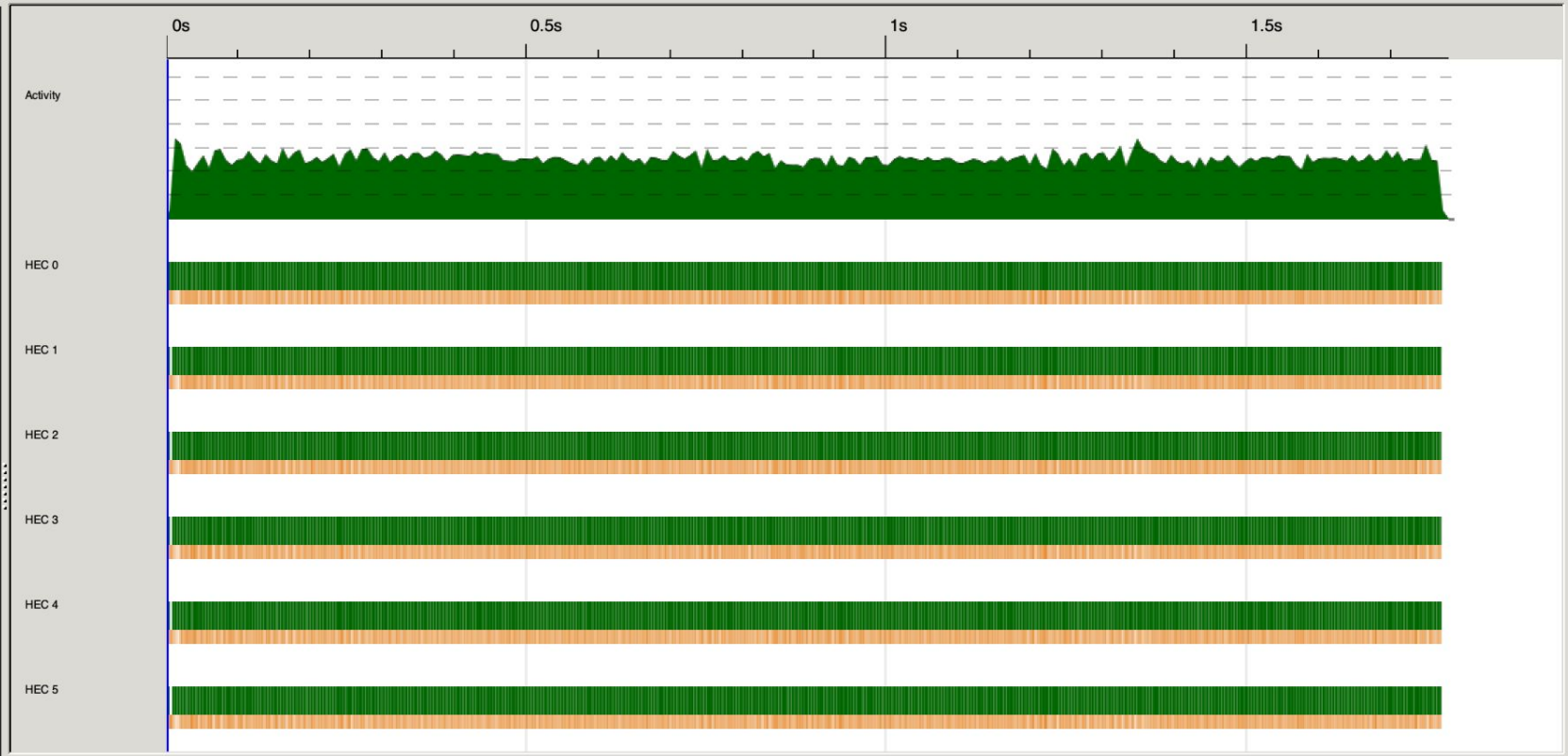
Table 3: Experiment Results for Simulate1000						
N	time(s)	converted	gc'd	fizzled	total	Speedup
seq	6.012	N/A	N/A	N/A	N/A	N/A
2	3.620	1998	1	1	2000	X1.66
3	2.590	1998	1	1	2000	X2.32
4	2.110	1998	1	1	2000	X2.85
5	1.920	1998	1	1	2000	X3.13
6	1.780	1998	1	1	2000	X3.38

File View Move Help

Key | Traces | Bookmarks

- running
- GC
- GC waiting
- create thread
- seq GC req
- par GC req
- migrate thread
- thread wakeup
- shutdown
- user message
- perf counter
- perf tracepoint
- create spark
- dud spark
- overflowed spark
- run spark
- fizzled spark
- GCed spark

Timeline



Time | Heap | GC | Spark stats | Spark sizes | Process info | Raw events

Total time: 1.780s
Mutator time: 895.437ms
GC time: 884.992ms

Evaluation

Sounds inspiring!

Move to the social network of GitHub

Evaluation and Benchmark

Table 5: Experiment Results for SNAP		
Method	time	Speedup
seq	44h 43 mins	N/A
ParBC-N6	18h 38 mins	X2.40
NetworkX	16h 23 mins	X2.73

Thank you