Connected components

Смирнов Кирилл
371

Определение:

Две вершины u и v называются **связаными** *(англ. adjacent)*, если в графе G существует путь из u в v (обозначение: $u \rightsquigarrow v$).

Определение:

Компонентой связности *(англ. connected component)* называется класс эквивалентности относительно связности.

Пример использования

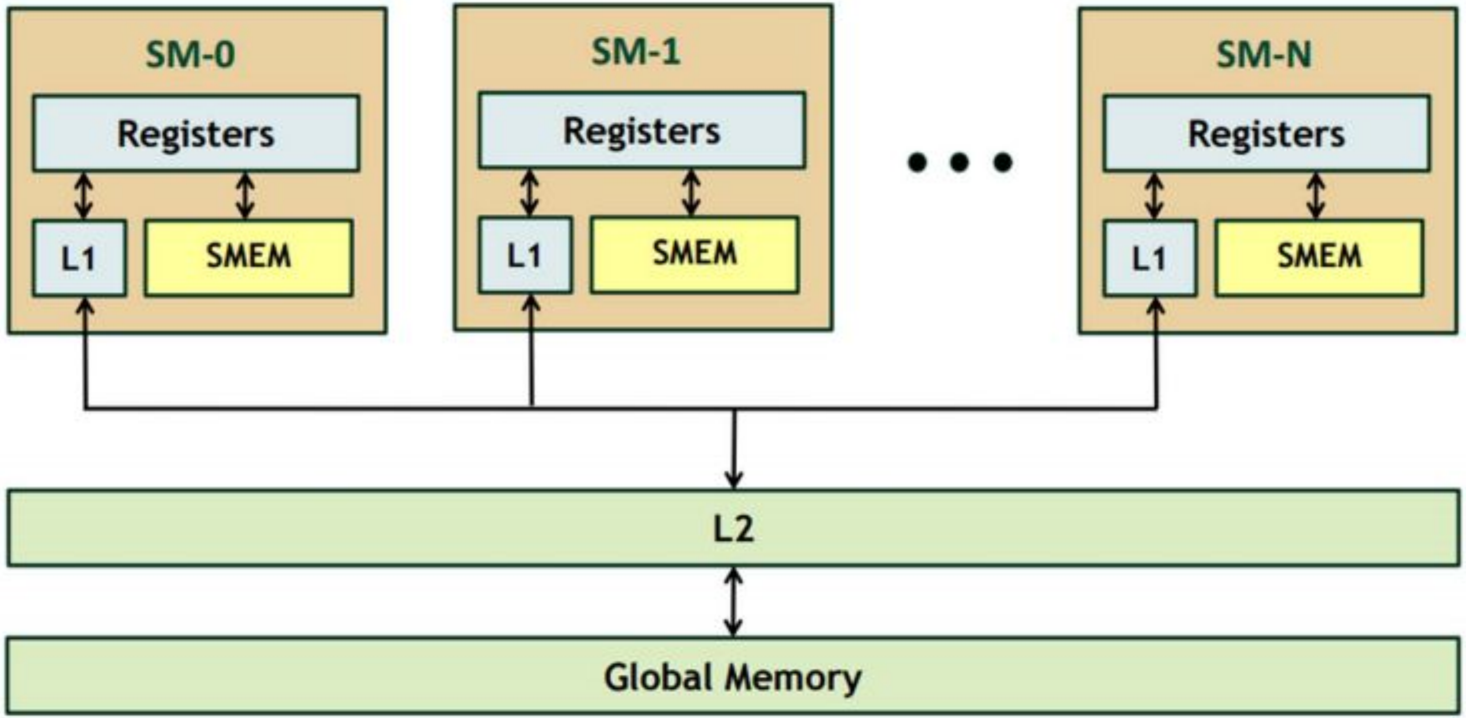
1. Кластеризация
    a. группировка пикселей изображения в группы (aka Pixel connectivity)
2. Social networks
    a. Изучения свойств общества посредством представления его в виде графа (aka Social network analysis)
3. Pipeline
    a. Определение компонент связности позволяет выстроить конвейеры, в который каждая компонента обрабатывается независимо

Serial algorithm

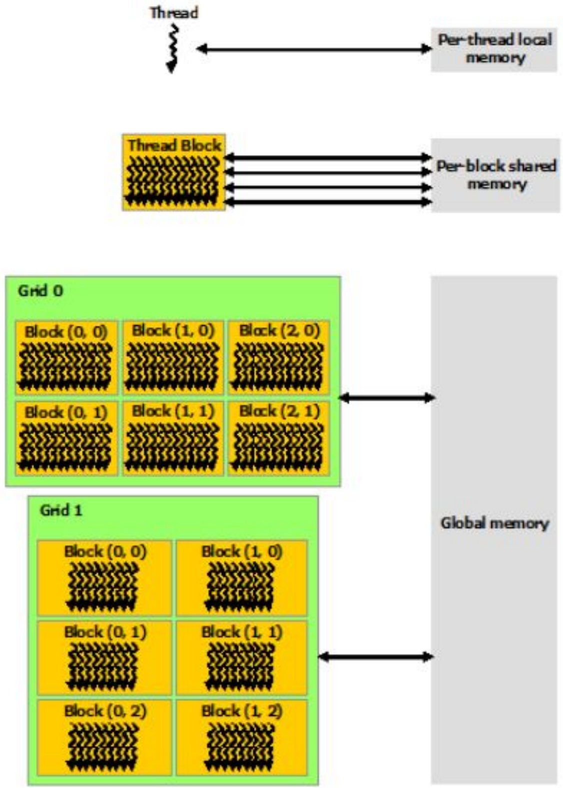
```
int BFS(G: (V, E), source: int, destination: int):
    d = int[|V|]
    fill(d, ∞)
    d[source] = 0
    Q = ∅
    Q.push(source)
    while Q ≠ ∅
        u = Q.pop()
        for vu in E
            if d[v] == ∞
                d[v] = d[u] + 1
                Q.push(v)
    return d[destination]
```

```
function doDfs(G[n]: Graph):
   visited = array[n, false]

   function dfs(u: int):
      visited[u] = true
      for v: (u, v) in G
         if not visited[v]
            dfs(v)

   for i = 1 to n
      if not visited[i]
         dfs(i)
```

Память GPGPU



Потоки GPGPU



A Fast GPU Algorithm for Graph Connectivity

Jyothish Soman, Kothapalli Kishore, and P J Narayanan
IIIT-Hyderabad

Main idea

- DSU + Kruskal

Details

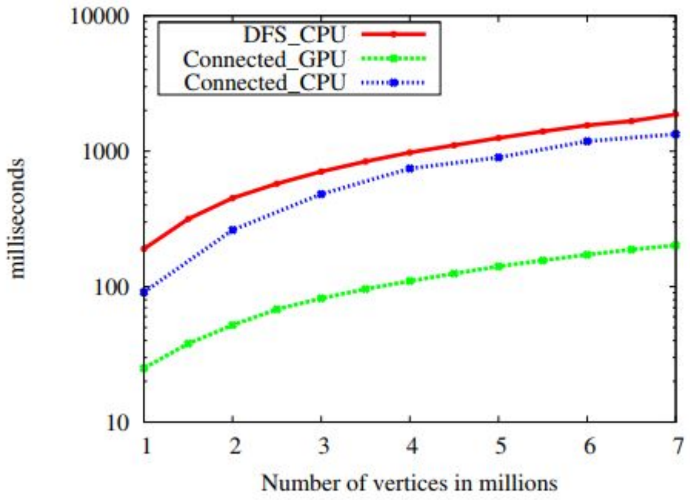
- Hooking
- Pointer jumping

DSU(disjoint-set union) - the data structure that maintains a collections S = {S1, S2, … , Sn} of disjoint dynamic sets © Introduction to algorithms 3rd edition Tomash. Cormen

Results:

Data set	# vertices, #edges (in M)	Run Time (in ms)
Live journal	4.8, 69	207
Wiki Talk	2.4, 5	12
Citation n/w	3.7, 16.5	127
Road Networks		
California	2, 5.5	27
Pennsylvania	1.0, 3.0	15
Texas	1.4, 3.8	17

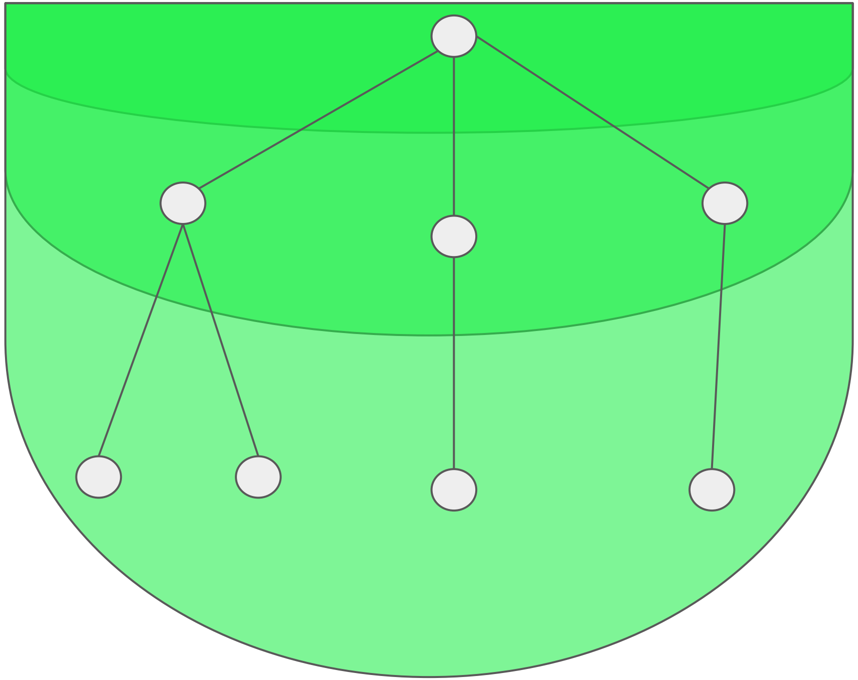
Table III
RUN TIME OF OUR ALGORITHM ON VARIOUS REAL-WORLD INSTANCES.

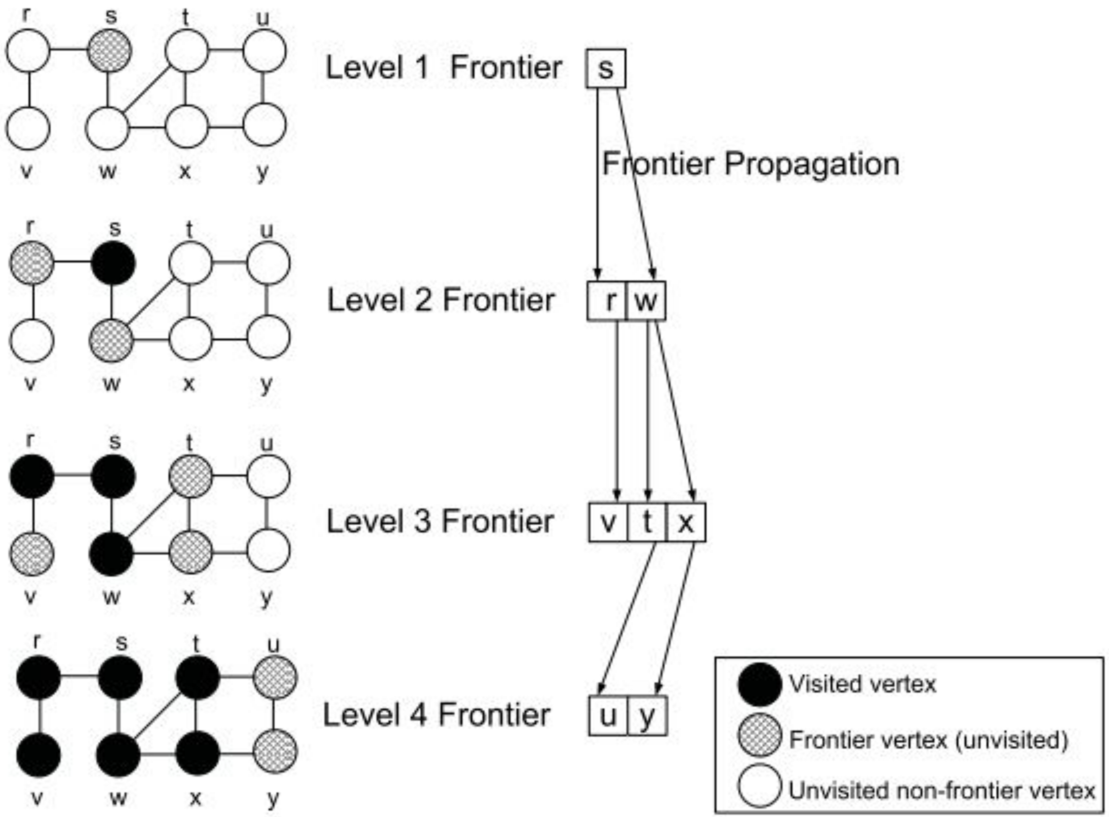


An Effective GPU Implementation of Breadth-First Search

Lijuan Luo Martin Wong Wen-mei Hwu
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign
{lluo3, mdfwong, w-hwu}@illinois.edu

```
int BFS(G: (V, E), source: int, destination: int):
    d = int[|V|]
    fill(d, ∞)
    d[source] = 0
    Q = ∅
    Q.push(source)
    while Q ≠ ∅
        u = Q.pop()
        for vu in E
            if d[v] == ∞
                d[v] = d[u] + 1
                Q.push(v)
    return d[destination]
```





r
s
t
u
v
w
x
y
Level 1 Frontier
s
Frontier Propagation
Level 2 Frontier
r
w
Level 3 Frontier
v
t
x
Level 4 Frontier
u
y
Visited vertex
Frontier vertex (unvisited)
Unvisited non-frontier vertex

Synchronization

- Atomic operations
- Host - Device communication

Results:

Table 1: BFS results on regular graphs

#Verte	IIIT-BFS	CPU-BFS	UIUC-BFS	Sp.
1M	462.8ms	146.7ms	67.8ms	2.2
2M	1129.2ms	311.8ms	121.0ms	2.6
5M	4092.2ms	1402.2ms	266.0ms	5.3
7M	6597.5ms	2831.4ms	509.5ms	5.6
9M	9170.1ms	4388.3ms	449.3ms	9.8
10M	11019.8ms	5023.0ms	488.0ms	10.3

Table 2: BFS results on real world graphs

	#Vertex	IIIT-BFS	CPU-BFS	UIUC-BFS	Sp.
New York	264,346	79.9ms	41.6ms	19.4ms	2.1
Florida	1,070,376	372.0ms	120.7ms	61.7ms	2.0
USA-East	3,598,623	1471.1ms	581.4ms	158.5ms	3.7
USA-West	6,262,104	2579.4ms	1323.0ms	236.6ms	5.6

Table 3: BFS results on scale-free graphs

#Vertex	IIIT-BFS	CPU-BFS	UIUC-BFS
1M	161.5ms	52.8ms	100.7ms
5M	1015.4ms	284.0ms	302.0ms
10M	2252.8ms	506.9ms	483.6ms

My results

Languages & tools:

- C++11
- CUDA 7.5

Hardware:

- i5-5200U CPU @ 2.20GHz × 4
- GeForce 920M

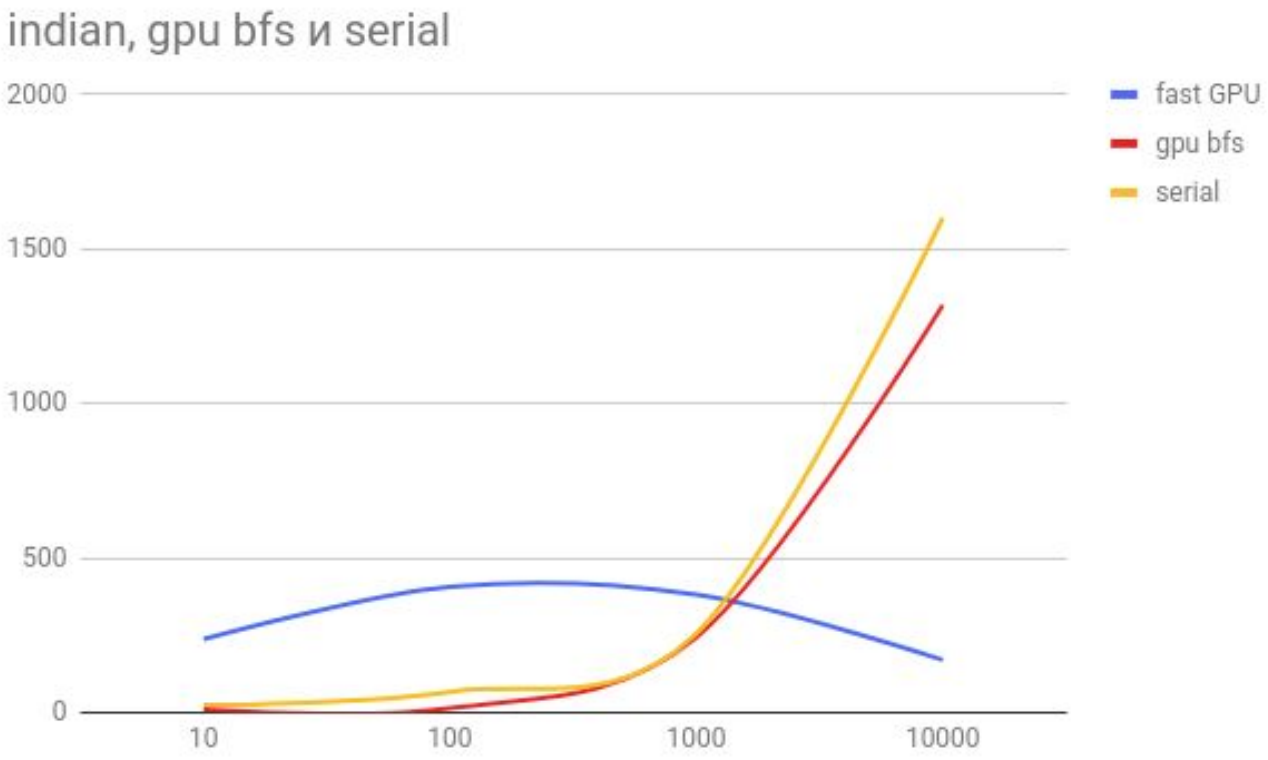
Source: https://github.com/cmirnov/Connected-components

Benchmarking

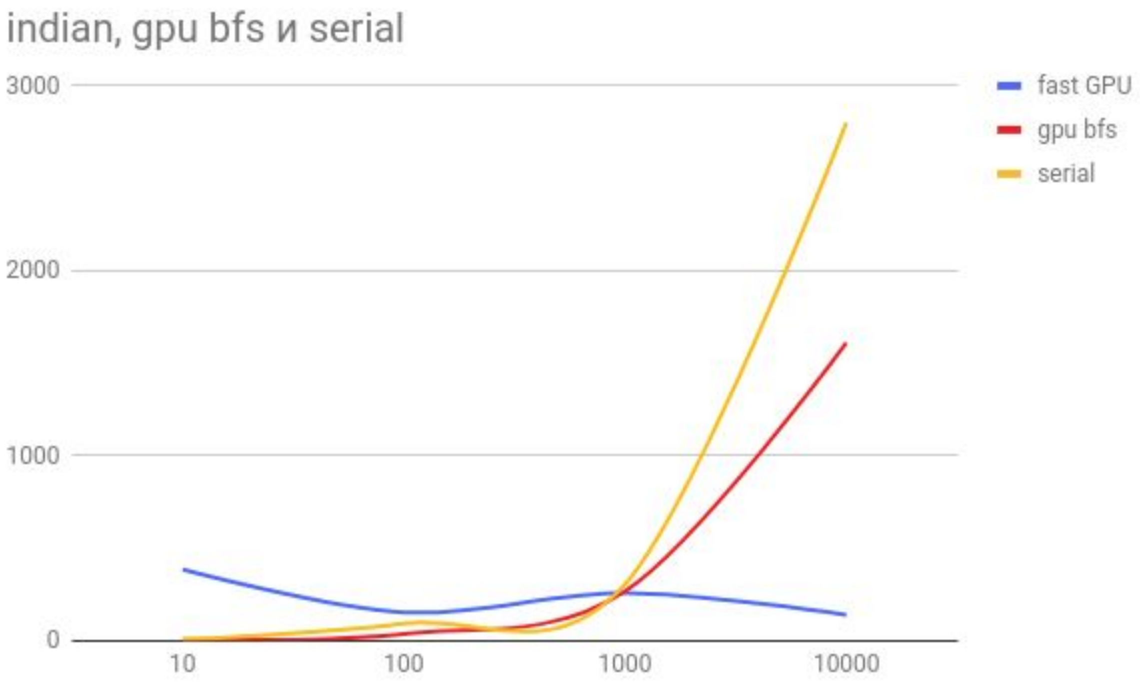
- In additional to these two algorithms, serial BFS algorithms was implemented to compare GPU and CPU algorithms efficiencies.
- All measurements were done on syntactic date. The generator may be found in gen.cpp
- There are 3 different graph topologies:
  - each node has at least 1 edge (fig. 1)
  - each node has at least 4 edges (fig. 2)
  - each node has at least 16 edges (fig. 3)
- The graph size range is from 10 nodes to 10000 nodes

Implemented algorithms

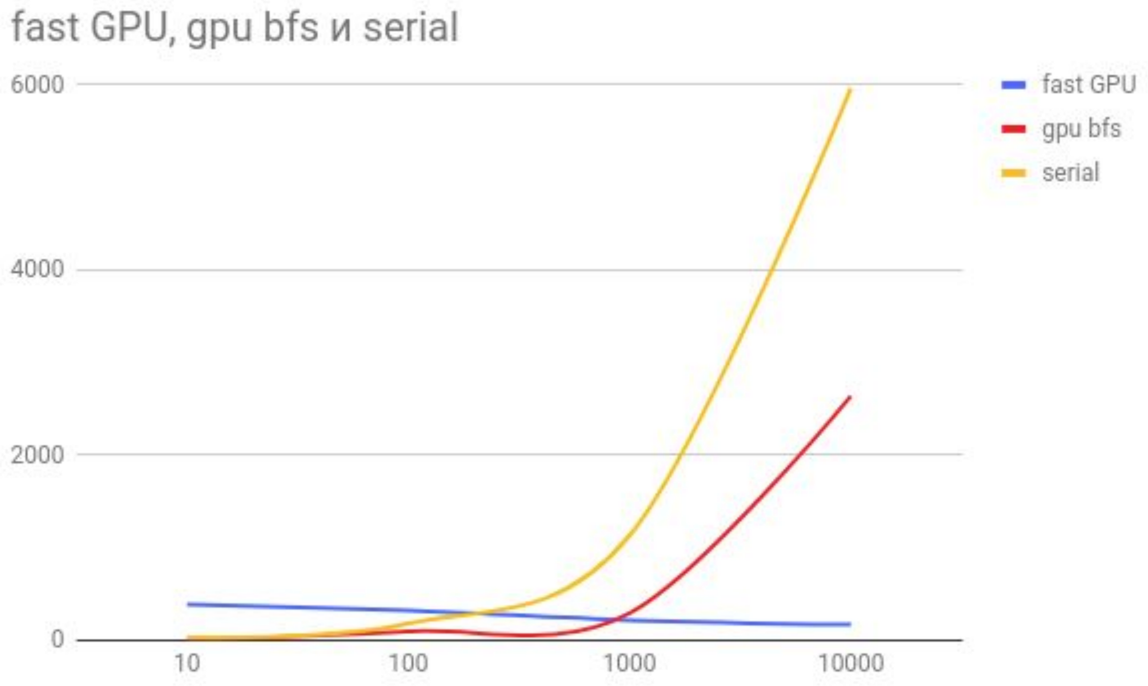
- Fast (see A Fast GPU Algorithm For Graph Connectivity)
- GPU BFS (see An Effective GPU Implementation Of Breadth-First Search)
- Serial (see https://en.wikipedia.org/wiki/Breadth-first_search)



each node has only one edge



each node has 4 edges



each node has 16 edges

Conclusion

- The Fast GPU algorithm is preferable for big graphs
- The Fast GPU algorithm is preferable for graphs with big node degrees
- GPU BFS is more efficient on small graphs

Graphs explanation

all GPU algorithms have interesting fluctuating artifacts. It’s especially noticeable at the second picture (4-edges). One possible explanation may be specific qualities of GPU scheduler which prefers overfitting rather then starving. As a result, execution time drops a little bit.