Community Detection on a GPU

Fredrik Manne, Md. Naim
University of Bergen, Norway

Mahantesh Halappanavar, Antonio Tumeo
Pacific Northwest National Laboratories, USA
Community Detection

Objective: Cluster vertices that “belong together”.

Modularity indicates the density of intra-communities links as compared to inter-communities.
The Louvain Algorithm [Blondel et al. 2008]

- Greedy, bottom up hierarchical method.
- Nodes move to neighboring communities to optimize local modularity.
One pass of the Louvain algorithm

Repeat {
  For each vertex $v$
    For each neighboring community $C_j$
      Compute modularity gain if $v$ moves to $C_j$

      If $v$ can achieve a positive gain by moving
        Move $v$ to $C_j$ giving highest gain

  } Until no further modularity gain

Contract each community to a single node and repeat
Determining which community to move to

For each vertex $v$
   For each vertex $w$ in $N(v)$
      Hash edge weight of $(v, w)$ with $C_w$
Previous Parallel Implementations

• [Wickramaarachchi et al. 2014] MPI implementation, speedup up to 5 using 128 processors.

• [Ovelgönne 2013] Hadoop implementation

• [Cheong et al. 2013] GPU. Speedup up to 5 on single GPU and 17 on multi-GPU.

• [Halappanavar et al. 2015] OpenMP implementation, Speedup up to 16 using 32 threads.
GPU Threads

Thread Block

Divided into groups of **32** threads

= warp

Operates in SIMD

Warp Scheduler

**Multiprocessor** (192 cores)
(15 such Multiprocessors)
GPU Memory

Kepler Memory Hierarchy

- SM-0: Registers, L1, SMEM, Read only
- SM-1: Registers, L1, SMEM, Read only
- SM-N: Registers, L1, SMEM, Read only

L2

Global Memory (DRAM)
The Parallel Algorithm [Hallapanavar et al.]

Modularity optimization

Repeat {
    For each vertex $v$ in parallel {
        Determine possible gain by moving $v$ to each neighboring community
        Store community giving highest gain
    }
    For each vertex $v$ in parallel
        Move $v$ to its destination community,
}

} Until no further modularity gain

Community aggregation

For each community $C_j$ in parallel:

    Contract $C_j$ to a single node
Design Issues

What is the unit of computation on the GPU?

Different possibilities:
• Each thread handles one vertex
• Each warp handles one vertex
• Each thread block handles one vertex
One Vertex Per Thread

- Neighbor lists
- Threads in a warp
- Individual hash tables

+ Writes can be carried out simultaneously.
+ Little coordination between threads
+ Can use shared memory

- Cannot read coalesced
- Uneven load balance
- All threads must finish before thread block is done
One Warp Per Vertex

Neighbors of vertex v

Threads in a warp

Hash table for vertex v

+ Coalesced reading
+ Can use shared memory

-Might have unused threads if few neighbors
-Thread block must wait for all warps to finish
-Must use atomic updates
Several Warps Per Vertex

Neighbors of vertex $v$

Warp $i$ - Most vertices have small neighborhoods
- Poor utilization of resources
- Must hash in global memory
- Needs synchronization

Hash table for vertex $v$

+ Good for high degree vertices
Our Solution

Group vertices based on the size of their neighborhoods

\[
|N(v)| := 1 - 7 \quad \Rightarrow \quad \text{Bin1; Use quarter-warp and shared memory}
\]
\[
|N(v)| := 8 - 15 \quad \Rightarrow \quad \text{Bin2; Use half-warp and shared memory}
\]
\[
|N(v)| := 16 - 31 \quad \Rightarrow \quad \text{Bin3; Use full-warp and shared memory}
\]
\[
|N(v)| := 32 - L \quad \Rightarrow \quad \text{Bin4; Use full-warp and shared memory}
\]

Remaining vertices can be processed with thread blocks of different size using global memory

Use similar binning techniques to compute aggregated communities
Experiments

Modularity Optimization Phase:

[Graph showing speedup of GPU compared to OMP (32 threads) with y-axis labeled 'Speedup' and x-axis labeled 'Increasing #E (0.5M to 0.75B)']
Conclusion

- Still ongoing work
- Seems promising compared to OpenMP
- Binning based on vertex degree is key to success

To do
- Preprocessing to reduce size of input (degree 1)
- Process several bins simultaneously
- Better understanding of where the GPU algorithm works