Direction-Optimized Parallel BFS in PIM

Hang Hu[1], Euna Kim[2], Hyesoon Kim[2]

Huazhong University of Science and Technology[1], Georgia Institute of Technology[2]

Georgia Tech | Center for Research into Novel Computing Hierarchies

What is the Direction-Optimized Breadth-First Search Algorithm in *PIM*?

I. Motivation

- During a top-down BFS of a social network, the number of successful checks (which turn into the next frontier) ramps up and down exponentially [Fig.1]. Our motivation comes from that when the frontier is large, there exists an opportunity to perform the BFS more efficiently by searching in the reverse direction.

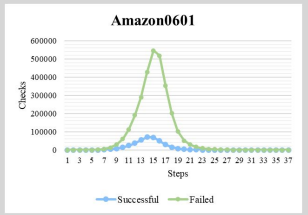

Fig.1 Number of checks on graph Amazon0601 (|V|=403394, |E|=3387388)

II. Direction-Optimized Parallel BFS

- Direction-optimized parallel BFS combines the conventional top-down and bottom-up algorithm to get a higher speedup.
- Implemented for distributed-memory systems using MPI.
- Partition methods: Chunk and Hash
- Bitmap vector is used to store MPI message

Sample BFS Code

```
/* Top-down step */
for(i = 0; i < Frontier.size(); i++) {
    u = Frontier[i];
    for(j = 0; j < LocalAdj[u].size(); j++) {
        v = LocalAdj[u][j];
        belong = v % size;
        v /= size;
        Next_send[belong][v / 8] |= 1 << (v % 8);
    }
}

/* Bottom-up step */
for(i = 0; i < local_v_num; i++) {
    if(LocalLevel[i] == -1) {
        for(j = 0; j < LocalAdj[i].size(); j++) {
            v = LocalAdj[i][j];
            belong1 = v % size;
            belong2 = v / size;
            if(frontier_status[belong1][belong2 / 8] >> (belong2 % 8) & 1) == 1) {
                LocalLevel[i] = cur_level + 1;
            }
        }
    }
}
```

III. Emerging Memory Technology

- PIM (Processing-in-Memory) is the integration of computing units and memory to reduce the unnecessary data movements which is the bottleneck of the performance in the traditional memory architecture

Top-down and Bottom-up BFS implementation & Evaluation

- **Benchmark:** Top-down and Bottom-up BFS algorithm
- **Dataset**: Selected graphs from Large Network Dataset Collection in SNAP: Stanford Network Analysis Platform.
- **Experiment setup**: Intel(R) Xeon(R) CPU E5-2696 v4 @ 2.20GHz, 88 CPUs, 252 GB shared memory
- **Evaluation**:
  1. Evaluated the top-down and bottom-up parallel BFS with partitions from 4 to 128 (each partition corresponds to an MPI node); Recorded the time cost and calculated the time ratio of bottom-up to top-down "B/T" [Fig.2].
  2. The majority of the computational work in BFS is checking edges of the frontier to see if the endpoint has been visited. Thus, the total number of checks and the number of successful checks are counted to calculate failed checks [Fig.3]. '*' represents the undirected version of a graph. The total checks are also counted [Fig.4]. "Minimum" indicates the ideal result the hybrid BFS will achieve.
  3. Evaluated the BFS time cost while utilizing different graph partitioning methods (chunk partition and hash partition) [Fig.5].
- **Analysis**:
  1. Despite the time difference between top-down and bottom-up BFS, two conclusions can be drawn:
     a) The time cost always decreases and then increases. The lowest point occurs around when there are 32 nodes.
     b) As the number of nodes increases, the time cost of top-down BFS gradually approaches that of bottom-up.
  2. On almost every tested graph, bottom-up BFS has steps with fewer failed checks compared to top-down BFS (7^{th} – 9^{th} step in Fig.3). This indicates the ideal speedup can be up to 1.514 on Amazon0601* [Fig.4].
  3. Hash partition makes the calculation more evenly distributed to each MPI node, resulting in faster BFS traversal compared to chunk partition [Fig.5].

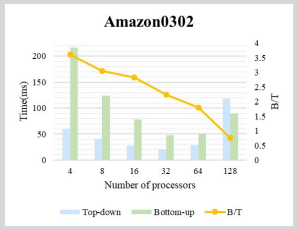

Fig.2 Time cost on graph Amazon0302 (|V|=262k,|E|=1.23M)

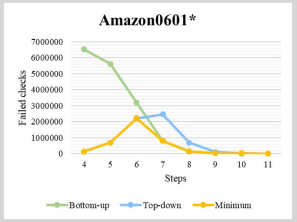

Fig.3 Failed checks on graph Amazon0601* (4^{th} – 11^{th} step)

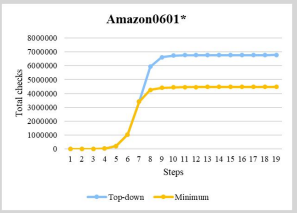

Fig.4 Total checks on graph Amazon0601*

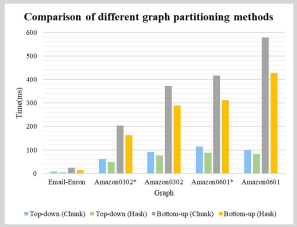

Fig.5 Comparison of different graph partitioning

Related Work

- Previous work has shown that given the message size, the more processors, the more communication overhead [Fig.6]. This explains the B/T curve as the number of nodes increases.
- Scott Beamer implemented a similar hybrid parallel BFS with a 2-D partition [Fig.7].
- Koji Ueno further introduced Bitmap-Based Sparse Matrix Representation to improve the space overhead. He also reordered the vertex IDs to improve load balance.

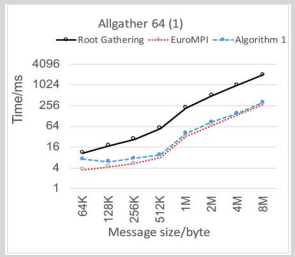

Fig.6 MPI_Allgather results

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,C} \\ \vdots & \ddots & \vdots \\ A_{R,1} & \cdots & A_{R,C} \end{pmatrix}$$

Fig.7 2-D partition

Future Work

- We aim to further optimize our parallel BFS by replacing MPI_Alltoall and MPI_Allgather with MPI_Send.
- Dynamic graphs have been an important topic in graph processing considering the real world graph dataset which constantly changing.. We are working on extending our distributed direction-optimized parallel BFS in order to process dynamic graphs.

Georgia Tech