Parallel Algorithms

Sathish Vadhiyar

PARALLEL SORTING

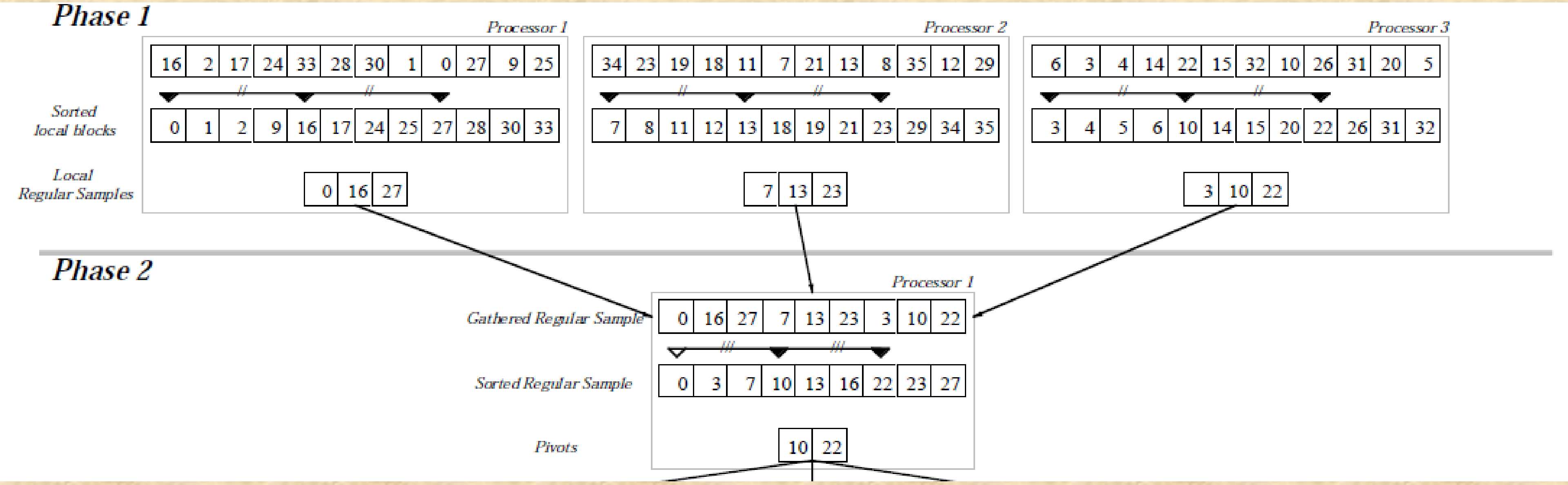
Introduction

- The input sequence of size N is distributed across P processors
- The output is such that elements in P_i is greater than elements in P_{i-1} and lesser than elements in P_{i+1}

Parallel Sorting by Regular Sampling (PSRS)

1. Each processor sorts its local data
2. Each processor selects a sample vector of size p-1; kth element is (n/p * (k+1)/p)
3. Samples are sent and merge-sorted on processor 0
4. Processor 0 defines a vector of p-1 *splitters* starting from p/2 element; i.e., kth element is p(k+1/2); broadcasts to the other processors

Example


Phase 1
Processor 1
16 2 17 24 33 28 30 1 0 27 9 25
Sorted local blocks
0 1 2 9 16 17 24 25 27 28 30 33
Local Regular Samples
0 16 27
Processor 2
34 23 19 18 11 7 21 13 8 35 12 29
7 8 11 12 13 18 19 21 23 29 34 35
7 13 23
Processor 3
6 3 4 14 22 15 32 10 26 31 20 5
3 4 5 6 10 14 15 20 22 26 31 32
3 10 22
Phase 2
Processor 1
Gathered Regular Sample
0 16 27 7 13 23 3 10 22
Sorted Regular Sample
0 3 7 10 13 16 22 23 27
Pivots
10 22

PSRS

5. Each processor sends local data to correct destination processors based on splitters; all-to-all exchange
6. Each processor merges the data chunk it receives

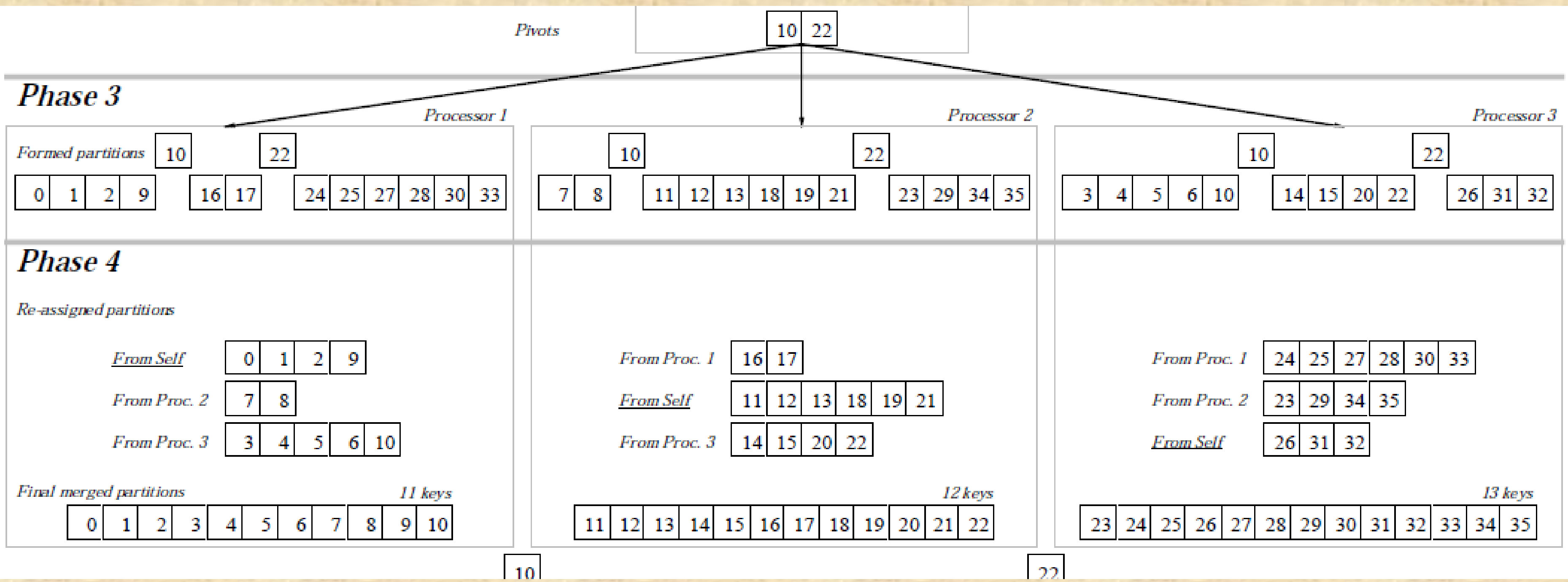
Step 5

- Each processor finds where each of the p-1 pivots divides its list, using a binary search
- i.e., finds the index of the largest element number larger than the jth pivot
- At this point, each processor has p sorted sublists with the property that each element in sublist i is greater than each element in sublist i-1 in any processor

Step 6

- Each processor i performs a p-way merge-sort to merge the ith sublists of p processors

Example Continued



Processor 1 re-assigned partitions:

From Self	0 1 2 9
From Proc. 2	7 8
From Proc. 3	3 4 5 6 10

Final merged partitions (11 keys): 0 1 2 3 4 5 6 7 8 9 10

Processor 2 re-assigned partitions:

From Proc. 1	16 17
From Self	11 12 13 18 19 21
From Proc. 3	14 15 20 22

Final merged partitions (12 keys): 11 12 13 14 15 16 17 18 19 20 21 22

Processor 3 re-assigned partitions:

From Proc. 1	24 25 27 28 30 33
From Proc. 2	23 29 34 35
From Self	26 31 32

Final merged partitions (13 keys): 23 24 25 26 27 28 29 30 31 32 33 34 35

Analysis

- The first phase of local sorting takes $O((n/p)\log(n/p))$
- 2nd phase:
  - Sorting $p(p-1)$ elements in processor 0 – $O(p^2 \log p^2)$
  - Each processor performs $p-1$ binary searches of n/p elements – $p\log(n/p)$
- 3rd phase: Each processor merges $(p-1)$ sublists
  - Size of data merged by any processor is no more than $2n/p$ (proof)
  - Complexity of this merge sort $2(n/p)\log p$
- Summing up: $O((n/p)\log n)$

Analysis

- 1st phase – no communication
- 2nd phase – p(p-1) data collected; p-1 data broadcast
- 3rd phase: Each processor sends (p-1) sublists to other p-1 processors; processors work on the sublists independently

- Graph Algorithms

Graph Traversal

- Graph search plays an important role in analyzing large data sets
- Relationship between data objects represented in the form of graphs
- Breadth first search used in finding shortest path or sets of paths

Parallel BFS
Level-synchronized algorithm

- Proceeds level-by-level starting with the source vertex
- Level of a vertex – its graph distance from the source
- Also, called **frontier-based** algorithm
- The parallel processes process a level, synchronize at the end of the level, before moving to the next level – Bulk Synchronous Parallelism (**BSP**) model
- How to decompose the graph (vertices, edges and adjacency matrix) among processors?

Distributed BFS with 1D Partitioning

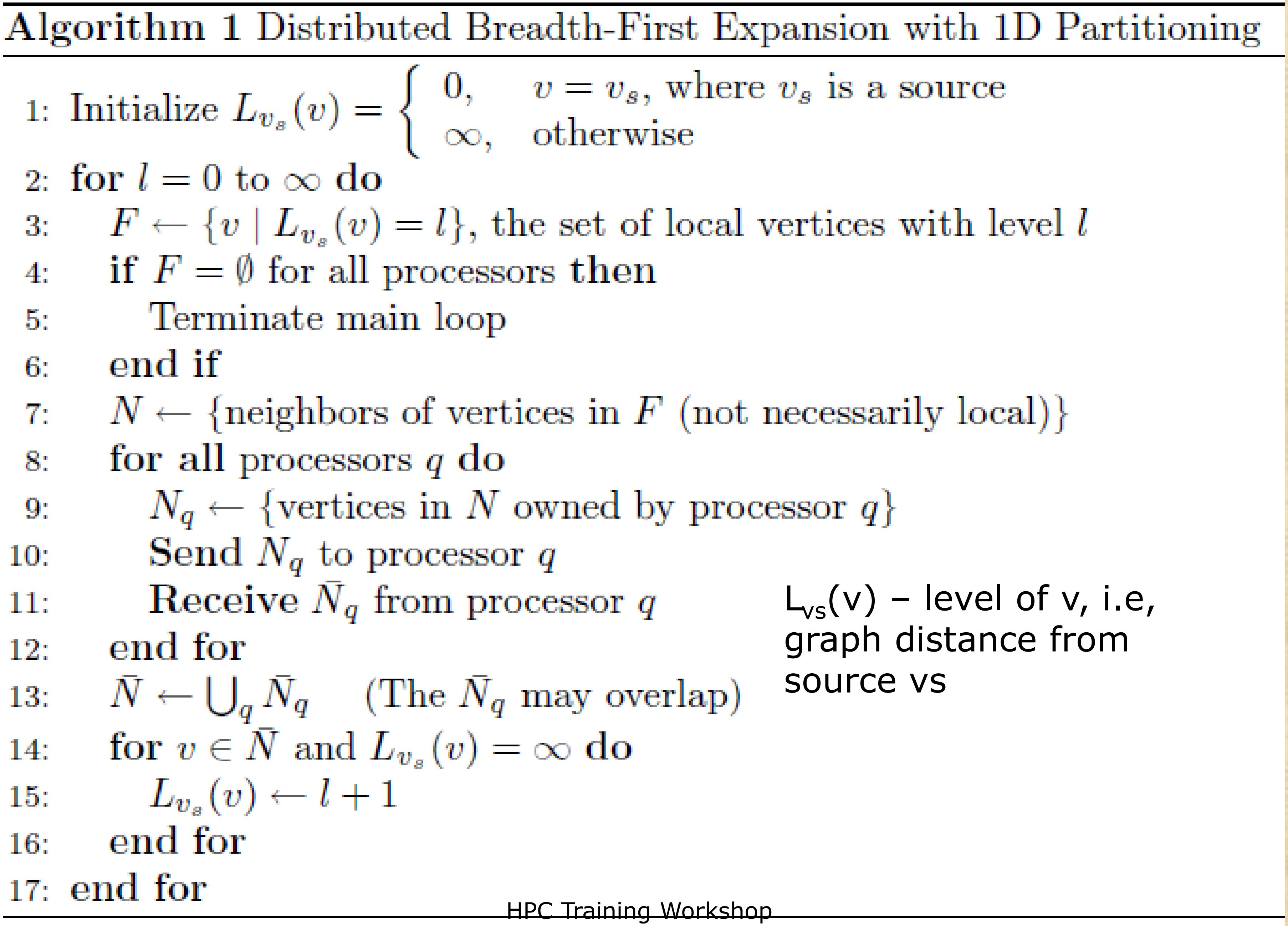
- Each vertex and edges emanating from it are owned by one processor
- 1-D partitioning of the adjacency matrix

$$\left[\begin{array}{c} A_1 \\ \hline A_2 \\ \hline \vdots \\ \hline A_P \end{array}\right]$$

- Edges emanating from vertex v is its edge list = list of vertex indices in row v of adjacency matrix A

1-D Partitioning

- At each level, each processor owns a set F - set of frontier vertices owned by the processor
- Edge lists of vertices in F are merged to form a set of neighboring vertices, N
- Some vertices of N owned by the same processor, while others owned by other processors
- Messages are sent to those processors to add these vertices to their frontier set for the next level

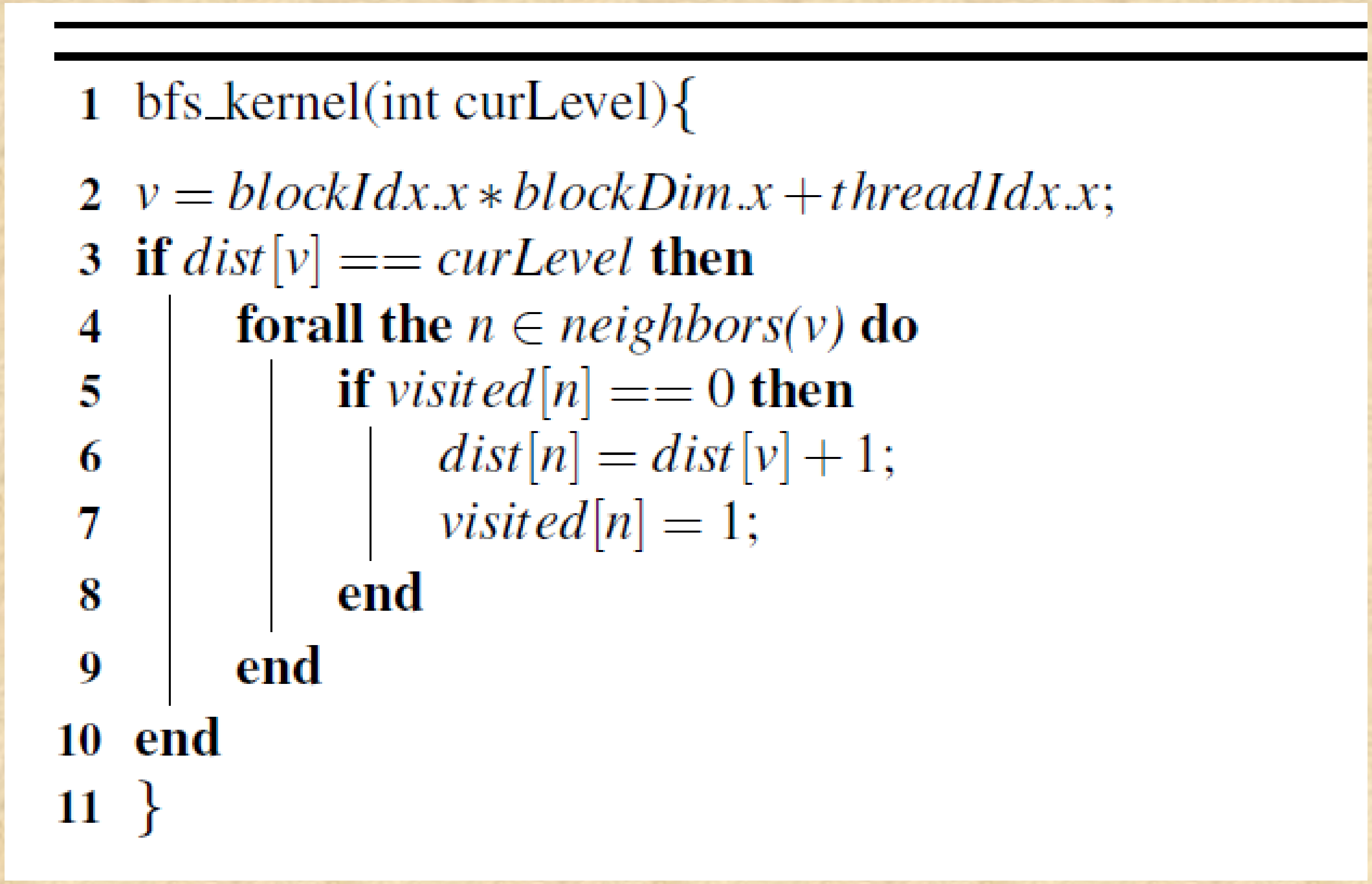


Algorithm 1 Distributed Breadth-First Expansion with 1D Partitioning

```
1: Initialize $L_{v_s}(v) = \begin{cases} 0, & v = v_s, \text{ where } v_s \text{ is a source} \\ \infty, & \text{otherwise} \end{cases}$
2: for $l = 0$ to $\infty$ do
3:     $F \leftarrow \{v \mid L_{v_s}(v) = l\}$, the set of local vertices with level $l$
4:     if $F = \emptyset$ for all processors then
5:         Terminate main loop
6:     end if
7:     $N \leftarrow$ {neighbors of vertices in $F$ (not necessarily local)}
8:     for all processors $q$ do
9:         $N_q \leftarrow$ {vertices in $N$ owned by processor $q$}
10:        Send $N_q$ to processor $q$
11:        Receive $\bar{N}_q$ from processor $q$
12:    end for
13:    $\bar{N} \leftarrow \bigcup_q \bar{N}_q$    (The $\bar{N}_q$ may overlap)
14:    for $v \in \bar{N}$ and $L_{v_s}(v) = \infty$ do
15:        $L_{v_s}(v) \leftarrow l + 1$
16:    end for
17: end for
```

$L_{vs}(v)$ – level of v, i.e, graph distance from source vs

BFS on GPUs



```
bfs_kernel(int curLevel){
$v = blockIdx.x * blockDim.x + threadIdx.x;$
if $dist[v] == curLevel$ then
    forall the $n \in neighbors(v)$ do
        if $visited[n] == 0$ then
            $dist[n] = dist[v] + 1;$
            $visited[n] = 1;$
        end
    end
end
}
```

BFS on GPUs

- One GPU thread for a vertex
- For each level, a GPU kernel is launched with the number of threads equal to the number of vertices in the graph
- Only those vertices whose assigned vertices are frontiers will become active
- Do we need atomics?
- Severe load imbalance among the treads
- Scope for improvement