Optimizing Breadth-First Search on Modern Multicore CPUs

Salvatore D. Andaloro

Department of Information Engineering and Computer Science, University of Trento

Breadth-First Search

- Breadth-First Search is a fundamental algorithm in graph analysis

Breadth-First Search

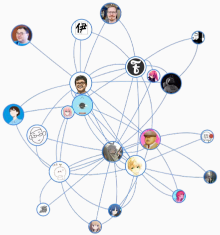
- Breadth-First Search is a fundamental algorithm in graph analysis
- Vertices are labeled based on the distance from a given *source* vertex

Breadth-First Search

- Breadth-First Search is a fundamental algorithm in graph analysis
- Vertices are labeled based on the distance from a given *source* vertex
- Used in many algorithms: Dijkstra, Maximum Flow, MSP...

Breadth-First Search

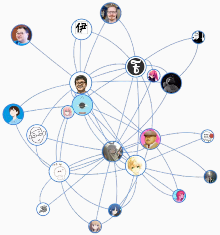
- Breadth-First Search is a fundamental algorithm in graph analysis
- Vertices are labeled based on the distance from a given *source* vertex
- Used in many algorithms: Dijkstra, Maximum Flow, MSP...



Social network

Breadth-First Search

- Breadth-First Search is a fundamental algorithm in graph analysis
- Vertices are labeled based on the distance from a given *source* vertex
- Used in many algorithms: Dijkstra, Maximum Flow, MSP...

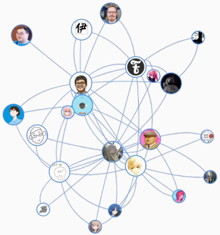


Social network

Road network

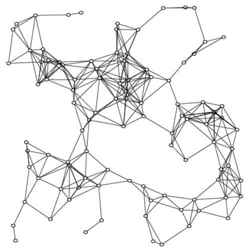
Breadth-First Search

- Breadth-First Search is a fundamental algorithm in graph analysis
- Vertices are labeled based on the distance from a given *source* vertex
- Used in many algorithms: Dijkstra, Maximum Flow, MSP...



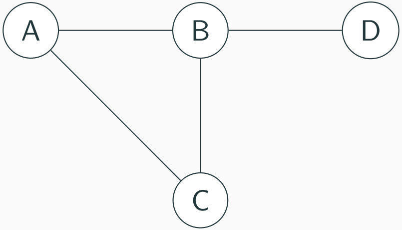
Social network

Road network



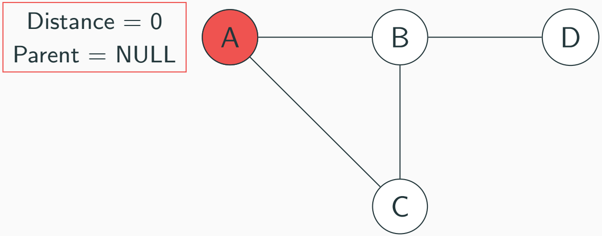
Synthetic graph

Breadth-First Search Example



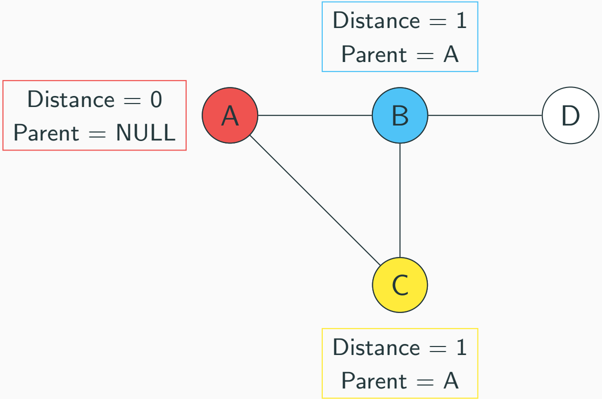
Source vertex: A

Breadth-First Search Example



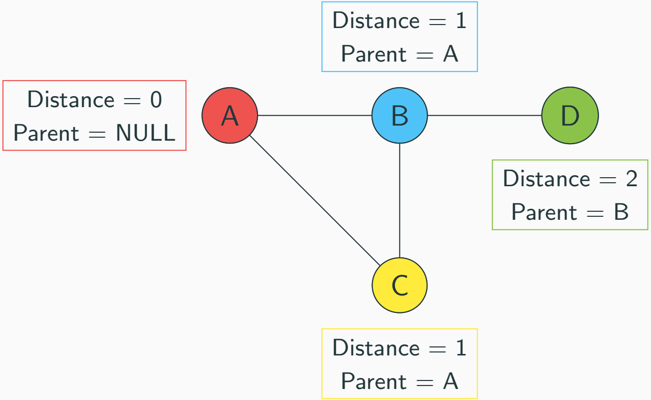
Frontier: A

Breadth-First Search Example



Frontier: B, C

Breadth-First Search Example



Frontier: D

Modern Computer Architectures

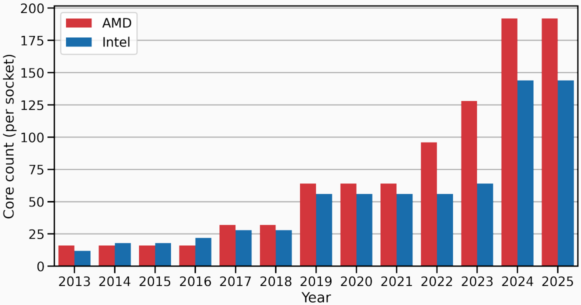
- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)

Modern Computer Architectures

- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)
- In practice, it is a **memory-bound algorithm**
    - Cache effects must be considered

Modern Computer Architectures

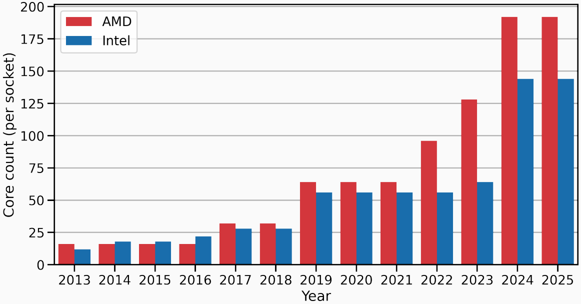
- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)
- In practice, it is a **memory-bound algorithm**
    - Cache effects must be considered
- CPUs exhibit growing amount of **parallelism**...



Evolution of core counts per socket for AMD and Intel processors

Modern Computer Architectures

- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)
- In practice, it is a **memory-bound algorithm**
    - Cache effects must be considered
- CPUs exhibit growing amount of **parallelism**...
- ...and new architectures are coming to the market (ARM, RISC-V)



Evolution of core counts per socket for AMD and Intel processors

Contents

- Two implementations with different parallel programming paradigms
    1. OpenMP implementation using the *MergedCSR* data structure
    2. Pthreads implementation using *MergedCSR* + custom synchronization routines

Contents

- Two implementations with different parallel programming paradigms
  1. OpenMP implementation using the *MergedCSR* data structure
  2. Pthreads implementation using *MergedCSR* + custom synchronization routines
- Evaluated against GAP Benchmark suite
- Speedups compared on three different architectures (AMD x86, RISC-V, ARM)

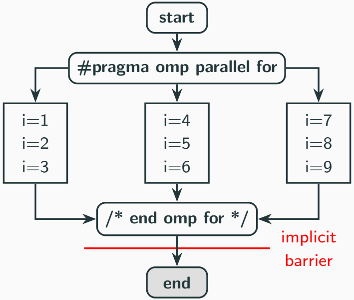
GAP suite logo

Compared architectures

OpenMP implementation

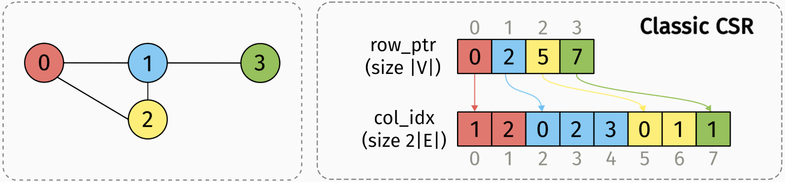
- **OpenMP** is a widely used framework for **parallel programming** in C and C++
- Uses simple compiler directives called pragmas

```
#pragma omp parallel for
for (int i = 1; i <= 9; i++) {
  A[i] = i
}
```



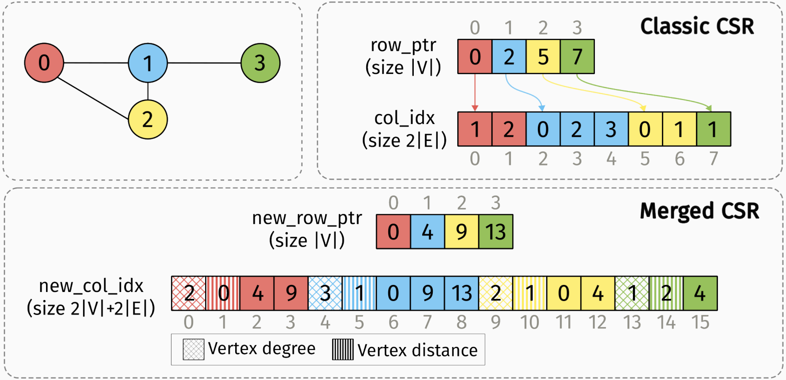
From CSR to MergedCSR

- Graphs are usually stored in the Compressed Sparse Row format (CSR)



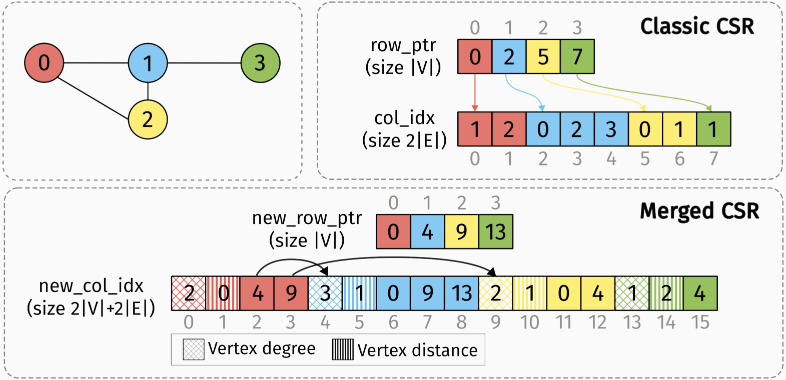
From CSR to MergedCSR

- Graphs are usually stored in the Compressed Sparse Row format (CSR)
- MergedCSR core idea: access only `row_ptr` array during BFS traversal
  - `row_ptr` array contains also algorithm-specific metadata (ex. distance)



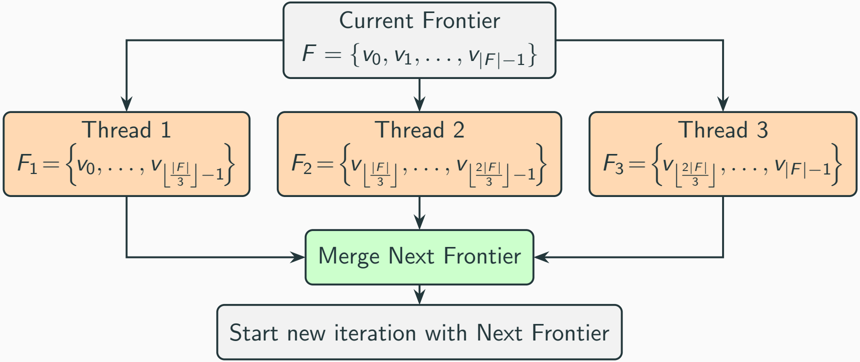
From CSR to MergedCSR

- Graphs are usually stored in the Compressed Sparse Row format (CSR)
- MergedCSR core idea: access only `row_ptr` array during BFS traversal
  - `row_ptr` array contains also algorithm-specific metadata (ex. distance)



Parallelization strategies

- Different parallelization strategies, depending on the graph type
- Strategy used: Frontier partitioning + Merge step



Implementation

```
#pragma omp declare reduction(vec_add : \
  omp_out.insert(omp_out.end(), omp_in.begin(), omp_in.end()))

#pragma omp parallel for reduction(vec_add : next_frontier)
↪  if(this_frontier.size() > 50)
for (const auto &v : this_frontier) {
  for (vertex i = v + 2; i < end; i++) { // Iterate over neighbors
    vertex neighbor = new_col_idx[i];
    // If neighbor is not visited, add to frontier
    if (DISTANCE(neighbor) == max()) {
    next_frontier.push_back(neighbor);
    DISTANCE(neighbor) = distance; // Set the distance
    }
  }
}
```

Inefficiencies of the OpenMP implementation

- Merging step is not parallel

Inefficiencies of the OpenMP implementation

- Merging step is not parallel
- Poor cache locality, as vertices are collected and repartitioned among the cores

Inefficiencies of the OpenMP implementation

- Merging step is not parallel
- Poor cache locality, as vertices are collected and repartitioned among the cores
- For large-diameter graphs, OpenMP enters the `parallel` region more than 10k times for a single BFS runs

Pthreads implementation summary

- Pthreads: low-level threading library to create and manage threads in C

Pthreads (unofficial) logo

Pthreads implementation summary

- Pthreads: low-level threading library to create and manage threads in C
- Implementation components:
  1. Custom data structure to handle the vertices in the frontier

Pthreads (unofficial) logo

Pthreads implementation summary

- Pthreads: low-level threading library to create and manage threads in C
- Implementation components:
  1. Custom data structure to handle the vertices in the frontier
  2. Work-stealing mechanism for load balancing

Pthreads (unofficial) logo

Pthreads implementation summary

- Pthreads: low-level threading library to create and manage threads in C
- Implementation components:
  1. Custom data structure to handle the vertices in the frontier
  2. Work-stealing mechanism for load balancing
  3. Thread pool to manage thread creation and destruction

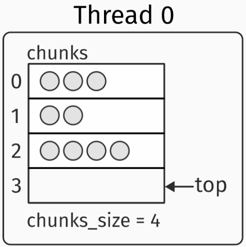
Pthreads (unofficial) logo

Pthreads implementation summary

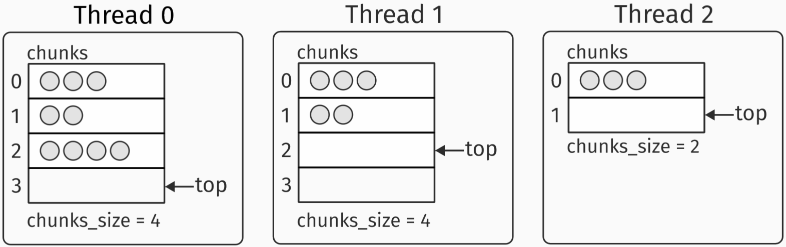
- Pthreads: low-level threading library to create and manage threads in C
- Implementation components:
  1. Custom data structure to handle the vertices in the frontier
  2. Work-stealing mechanism for load balancing
  3. Thread pool to manage thread creation and destruction
  4. Custom barrier for thread synchronization

Pthreads (unofficial) logo

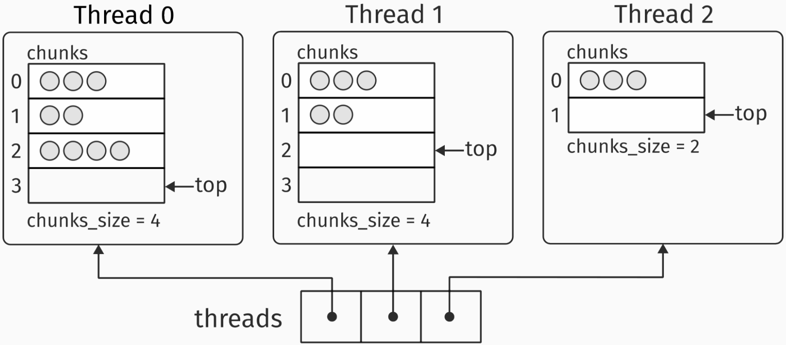
Frontier implementation



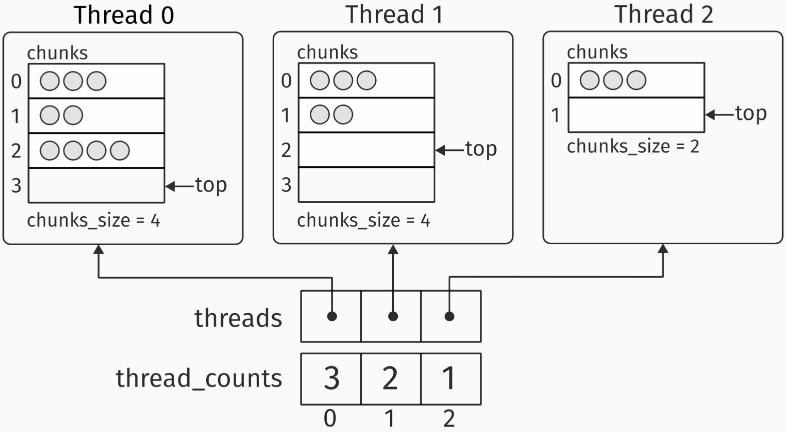
Frontier implementation



Frontier implementation

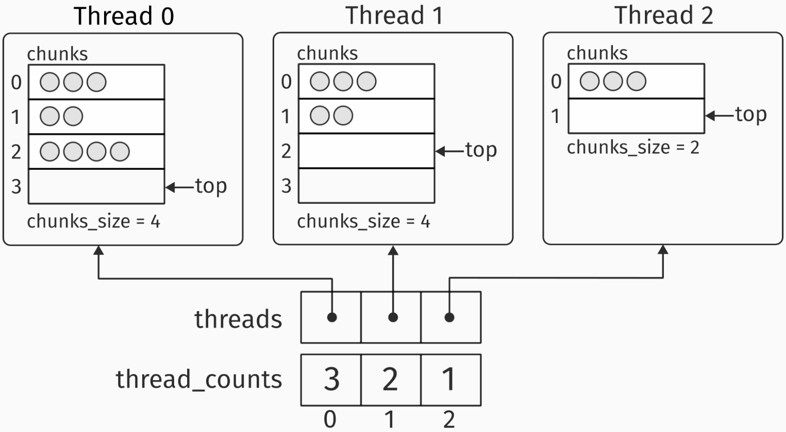


Frontier implementation



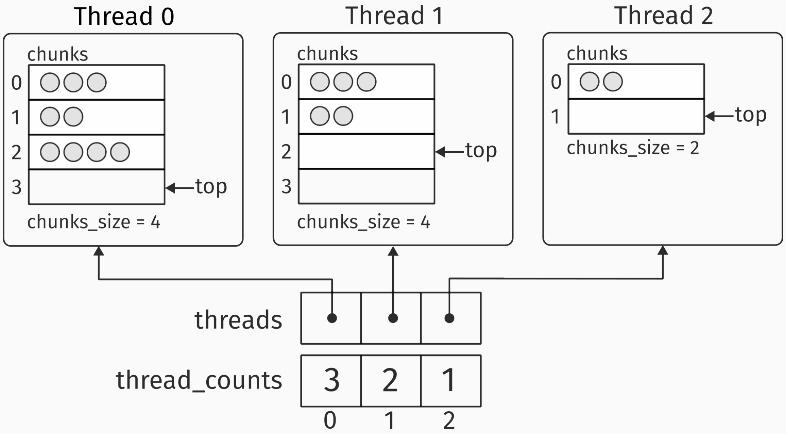
Work-stealing mechanism

Thread 2 processes its vertices...



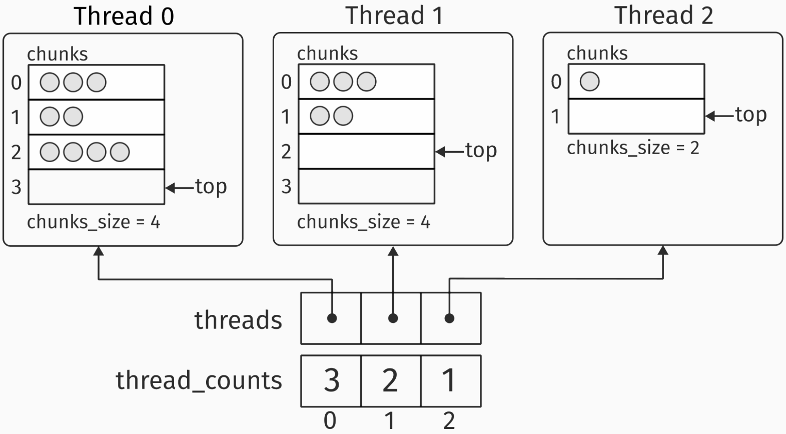
Work-stealing mechanism

Thread 2 processes its vertices...



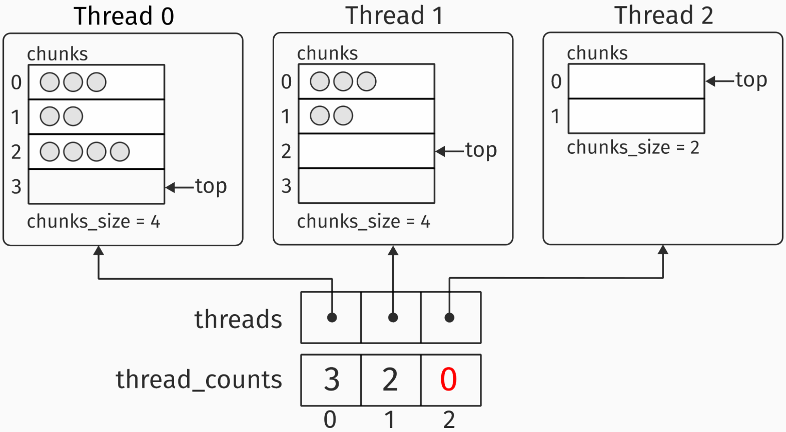
Work-stealing mechanism

Thread 2 processes its vertices...



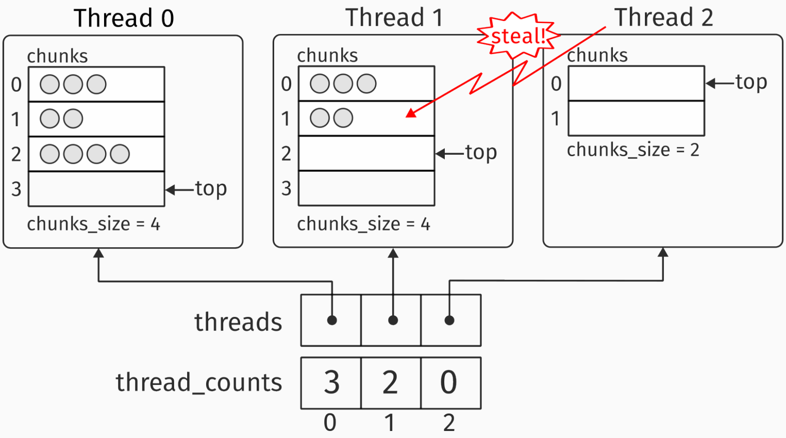
Work-stealing mechanism

Thread 2 is out of work, will attempt a steal soon...



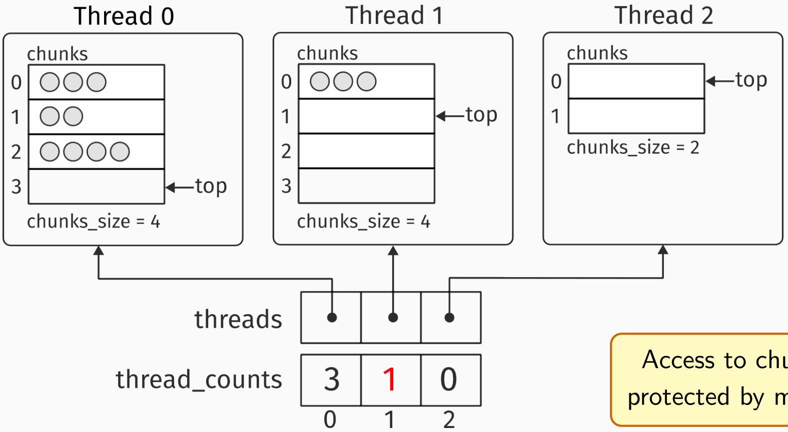
Work-stealing mechanism

Thread 2 steals a chunk of work from Thread 1...



Work-stealing mechanism

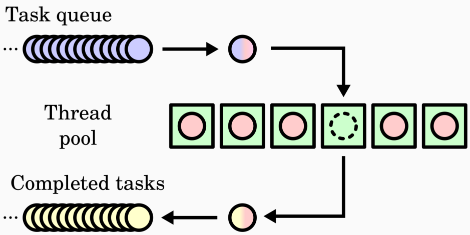
Thread 2 processes the stolen vertices and updates the global count.



Access to chunks is protected by mutexes.

Thread pool

- When the program is run, a group of threads is spawned
- At the beginning of each BFS run, the threads are awaken and the starting vertex is assigned to the 0^{th} thread



Sense-Reversal Centralized Barrier

- Central counter tracks arriving threads
- Last thread resets counter + toggles global sense
- Others wait until global = local sense
- All released together, barrier reusable