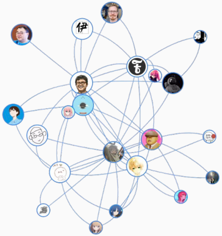
Optimizing Breadth-First Search on Modern Multicore CPUs

Salvatore D. Andaloro

Department of Information Engineering and Computer Science, University of Trento

Breadth-First Search

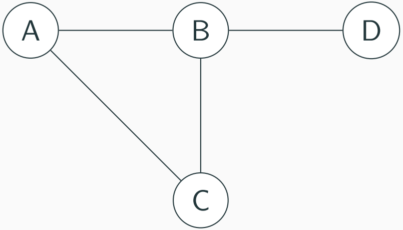
- Breadth-First Search is a fundamental algorithm in graph analysis
- Used in many algorithms: Dijkstra, Maximum Flow, MSP...
- Vertices are labeled based on the distance from a given *source* vertex



Social network

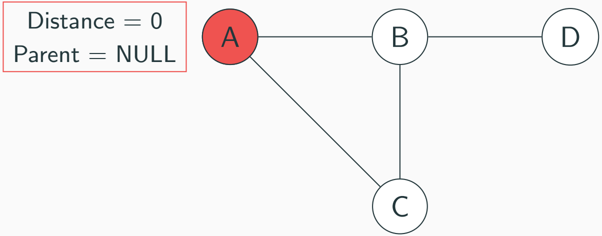
Road network

Breadth-First Search Example



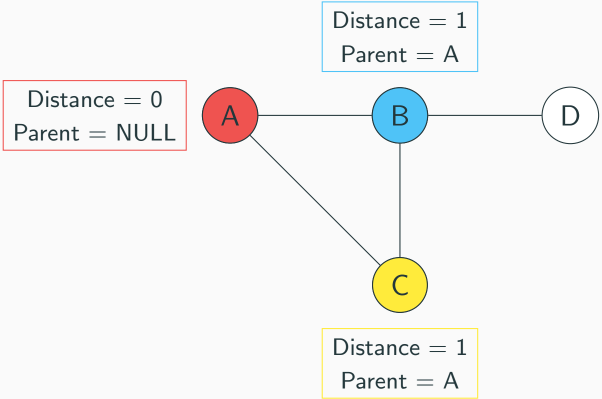
Source vertex: A

Breadth-First Search Example



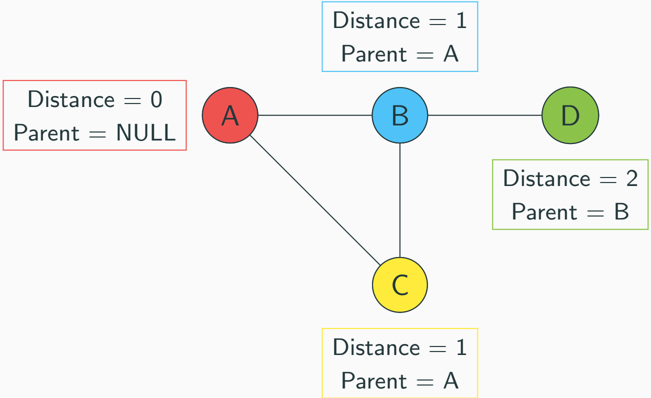
Frontier: A

Breadth-First Search Example



Frontier: B, C

Breadth-First Search Example



Distance = 0
Parent = NULL

Distance = 1
Parent = A

Distance = 2
Parent = B

Distance = 1
Parent = A

Frontier: D

Modern Computer Architectures

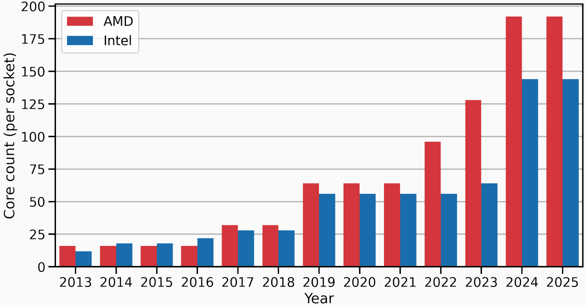
- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)

Modern Computer Architectures

- BFS has $\mathcal{O}(V + E)$ time and space complexity (under RAM model)
- In practice, it is a **memory-bound algorithm**
    - Algorithm exhibits poor cache locality

Modern Computer Architectures

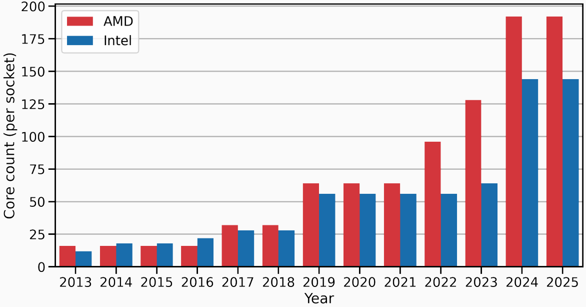
- BFS has $\mathcal{O}(V+E)$ time and space complexity (under RAM model)
- In practice, it is a **memory-bound algorithm**
  - Algorithm exhibits poor cache locality
- CPUs exhibit growing amount of **parallelism**...



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**
    - Algorithm exhibits poor cache locality
- 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

- New *MergedCSR* data structure
- Two optimized parallel implementations (OpenMP and pthreads)

Contents

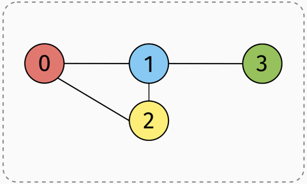
- New *MergedCSR* data structure
- Two optimized parallel implementations (OpenMP and pthreads)
- Evaluated against GAP Benchmark suite
- Speedups compared on three different architectures (AMD x86, RISC-V, ARM)

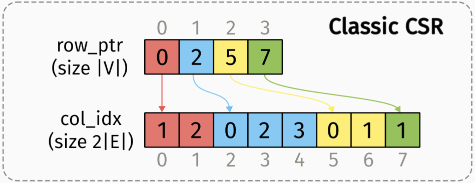
GAP suite logo

Compared architectures

From CSR to MergedCSR

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

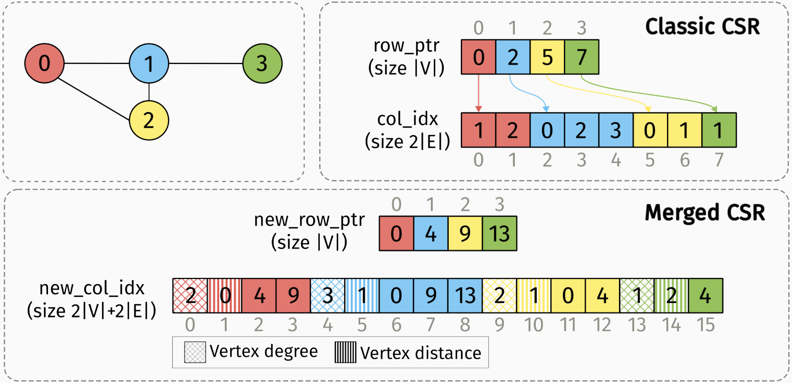




	0	1	2	3				
row_ptr (size |V|)	0	2	5	7				
col_idx (size 2|E|)	1	2	0	2	3	0	1	1
	0	1	2	3	4	5	6	7

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)



Classic CSR

row_ptr (size |V|)
0 1 2 3
0 2 5 7

col_idx (size 2|E|)
1 2 0 2 3 0 1 1
0 1 2 3 4 5 6 7

Merged CSR

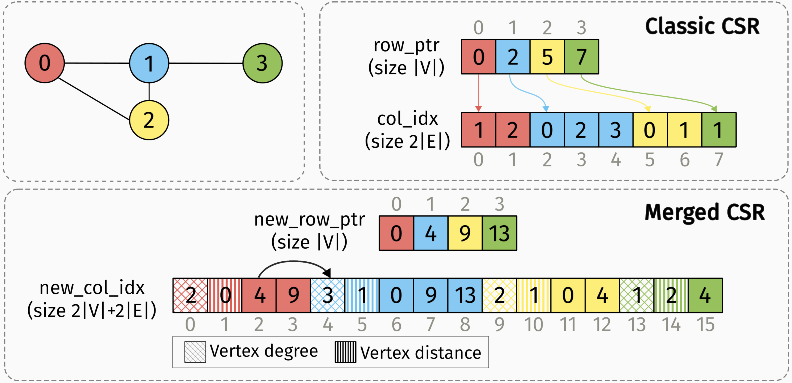
new_row_ptr (size |V|)
0 1 2 3
0 4 9 13

new_col_idx (size 2|V|+2|E|)
2 0 4 9 3 1 0 9 13 2 1 0 4 1 2 4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

▨ Vertex degree ▥ Vertex 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)



Classic CSR

	0	1	2	3
row_ptr (size |V|)	0	2	5	7

	0	1	2	3	4	5	6	7
col_idx (size 2|E|)	1	2	0	2	3	0	1	1

Merged CSR

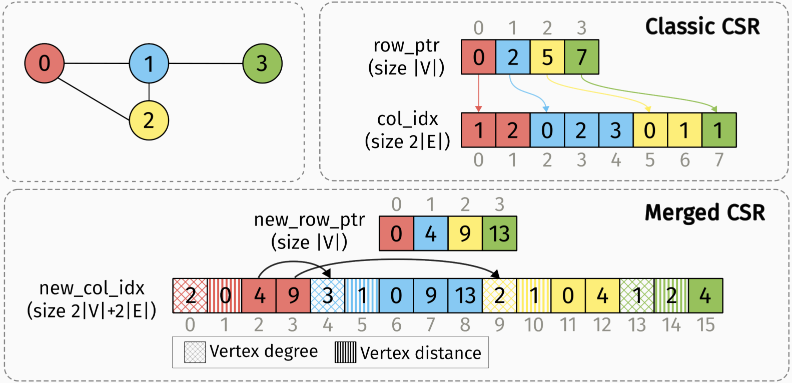
	0	1	2	3
new_row_ptr (size |V|)	0	4	9	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
new_col_idx (size 2|V|+2|E|)	2	0	4	9	3	1	0	9	13	2	1	0	4	1	2	4

▨ Vertex degree ▥ Vertex 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)



Classic CSR

	0	1	2	3
row_ptr (size |V|)	0	2	5	7

	0	1	2	3	4	5	6	7
col_idx (size 2|E|)	1	2	0	2	3	0	1	1

Merged CSR

	0	1	2	3
new_row_ptr (size |V|)	0	4	9	13

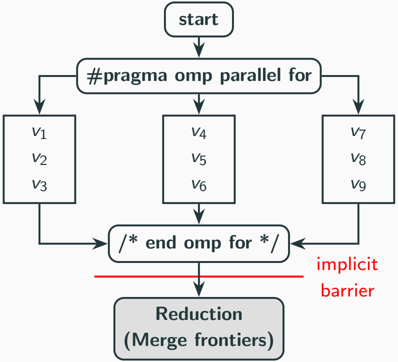
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
new_col_idx (size 2|V|+2|E|)	2	0	4	9	3	1	0	9	13	2	1	0	4	1	2	4

▨ Vertex degree ▥ Vertex distance

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
↪  reduction(vec_add : next_frontier)
for (const auto &v : this_frontier) {
  // Process vertex v
}
```



Inefficiencies of the OpenMP implementation

- Merging step is not parallel

Inefficiencies of the OpenMP implementation

- Merging step is not parallel
- In graphs with large diameter, OpenMP enters the `parallel for` 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

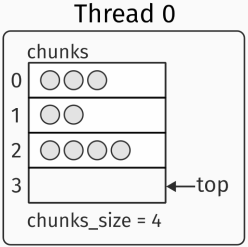
Pthreads (unofficial) logo

Pthreads implementation summary

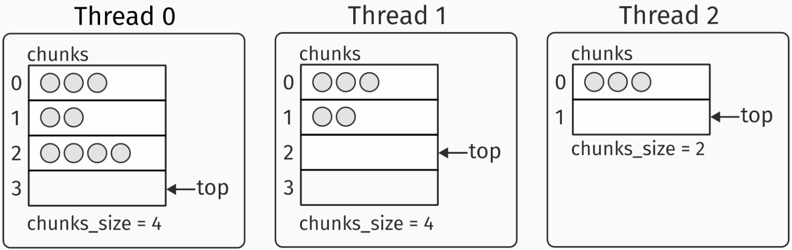
- Pthreads: low-level threading library to create and manage threads in C
- Implementation components:
  1. Custom data structure to handle the vertices in the frontier
  2. Work-stealing mechanism for load balancing
  3. Thread pool to manage thread creation and destruction
  4. Custom barrier for thread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

Pthreads (unofficial) logo

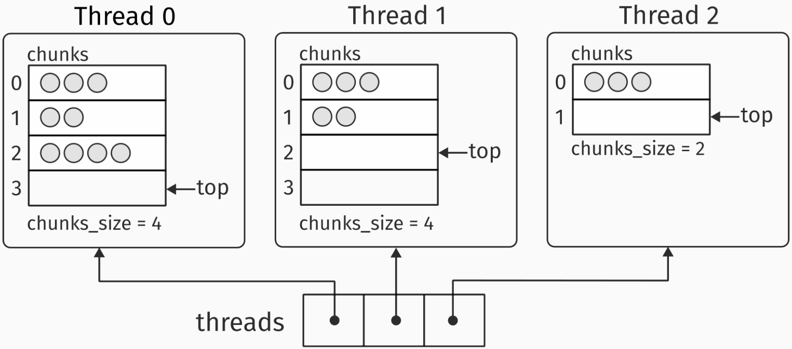
Frontier implementation



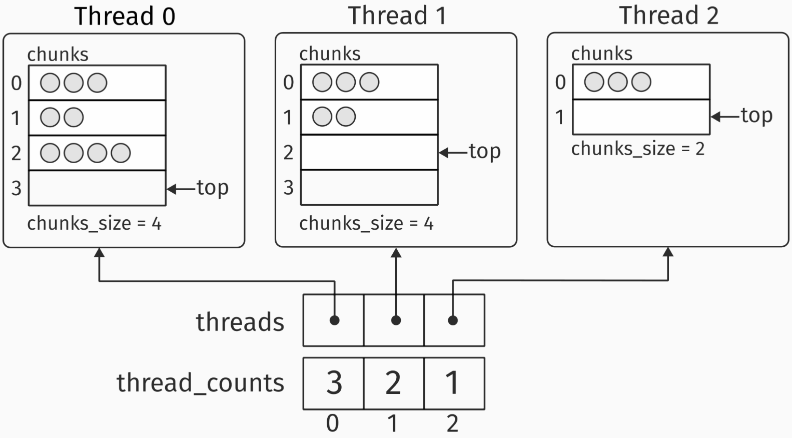
Frontier implementation



Frontier implementation

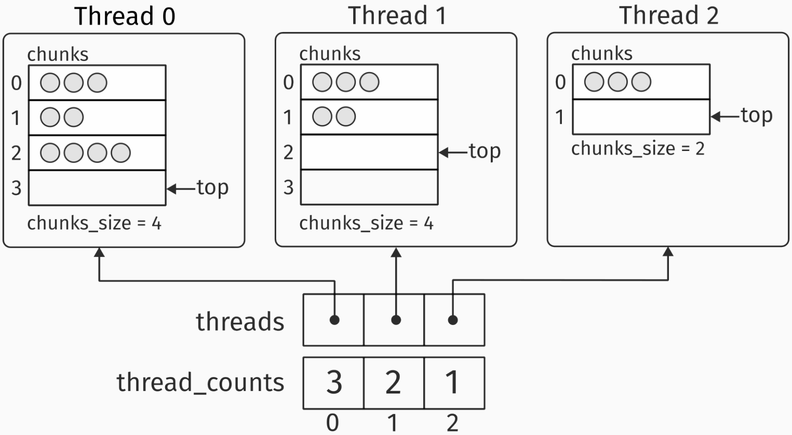


Frontier implementation



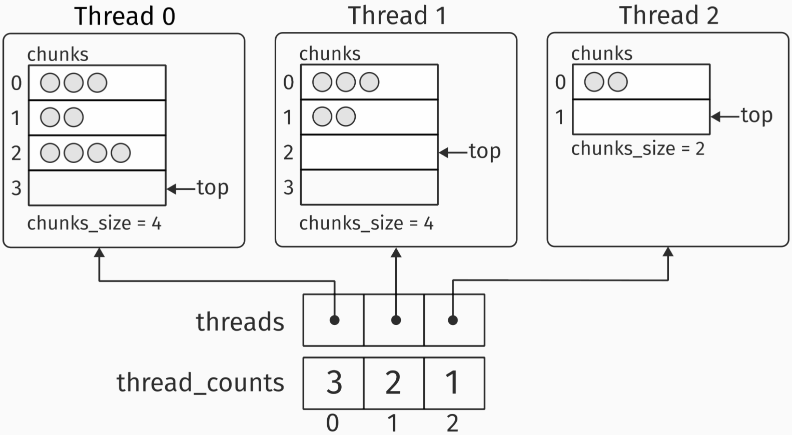
Work-stealing mechanism

Thread 2 processes its vertices...



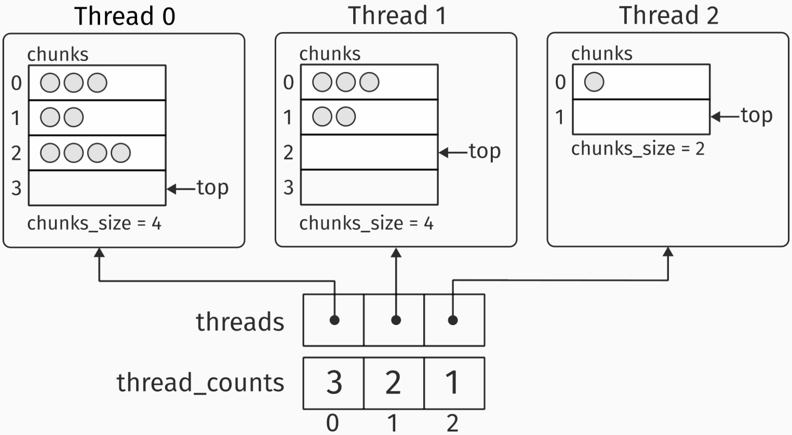
Work-stealing mechanism

Thread 2 processes its vertices...



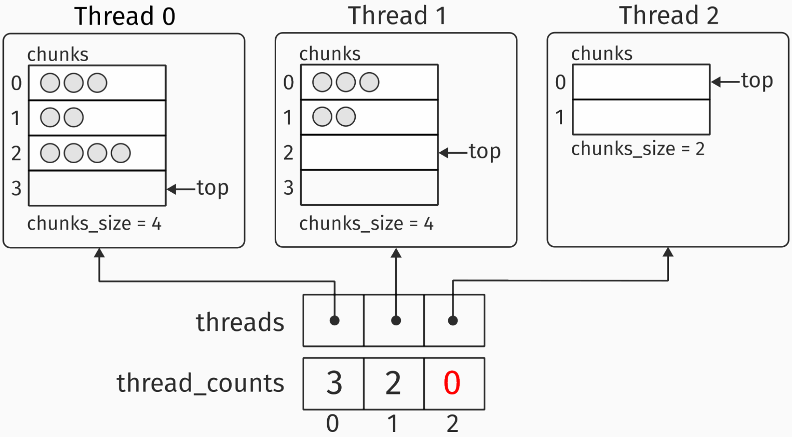
Work-stealing mechanism

Thread 2 processes its vertices...



Work-stealing mechanism

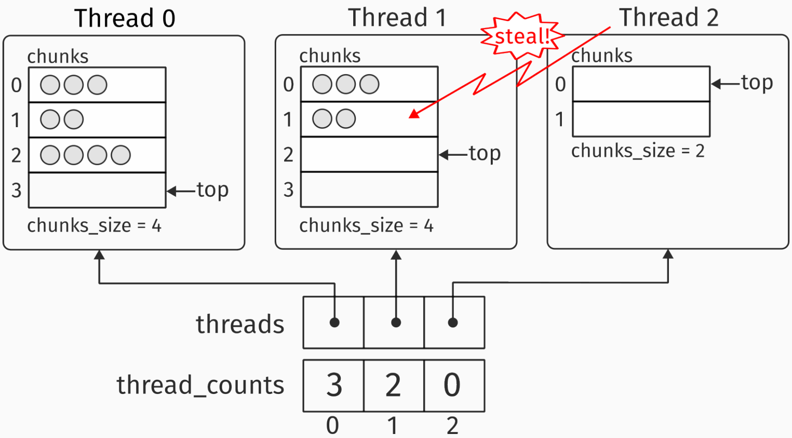
Thread 2 is out of work, will attempt a steal soon...



	0	1	2
thread_counts	3	2	0

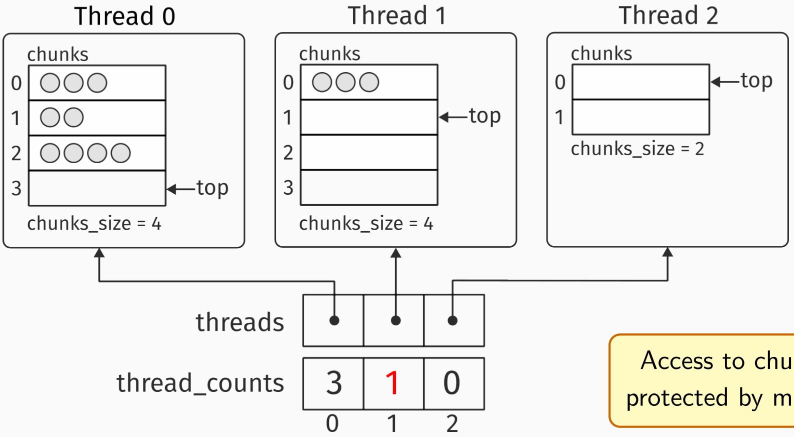
Work-stealing mechanism

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



Work-stealing mechanism

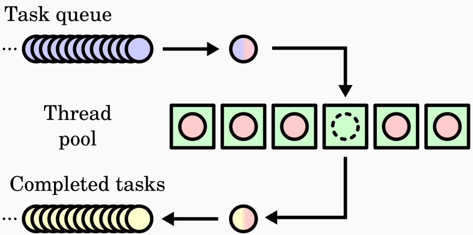
Thread 2 processes the stolen vertices and updates the global count.



Access to chunks is protected by mutexes.

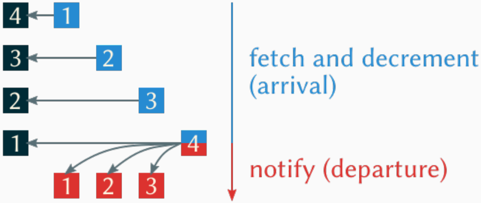
Thread pool

- When the program is run, a group of threads is spawned
- At the beginning of each BFS run, the threads are awaken
  1. Process own chunks
  2. Steal work from other threads



Sense-Reversal Centralized Barrier

- Barrier: point that threads must reach before any can proceed
- Procedure:
    1. Central counter tracks arriving threads
    2. Last thread resets counter + increment distance
    3. Others threads spin wait until distance changes
    4. All threads are released together



Experimental setup

- Experiments run on 3 platforms:
  - AMD EPYC 7543 CPU @ 2.8 GHz (32 cores)
  - Sophon SG2042 RISC-V CPU @ 2.0 GHz (64 cores)
  - NVIDIA Grace CPU Superchip @ up to 3.0 GHz (144 cores)

Experimental setup

- Experiments run on 3 platforms:
  - AMD EPYC 7543 CPU @ 2.8 GHz (32 cores)
  - Sophon SG2042 RISC-V CPU @ 2.0 GHz (64 cores)
  - NVIDIA Grace CPU Superchip @ up to 3.0 GHz (144 cores)
- Datasets: 3 road networks (USA, Europe, Asia), 3 FEM meshes (Earth's crust, steel hook, porous material), 1 random geometric graph (RGG)

Experimental setup

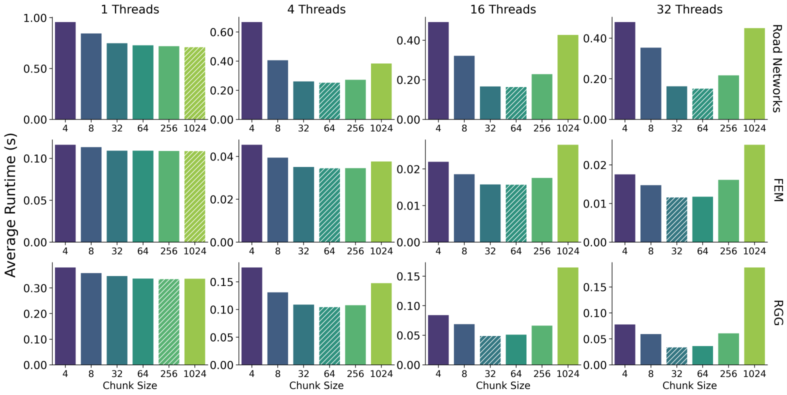
- Experiments run on 3 platforms:
    - AMD EPYC 7543 CPU @ 2.8 GHz (32 cores)
    - Sophon SG2042 RISC-V CPU @ 2.0 GHz (64 cores)
    - NVIDIA Grace CPU Superchip @ up to 3.0 GHz (144 cores)
- Datasets: 3 road networks (USA, Europe, Asia), 3 FEM meshes (Earth's crust, steel hook, porous material), 1 random geometric graph (RGG)
- Tools: GCC compiler, Likwid, SBatchMan

Experimental setup

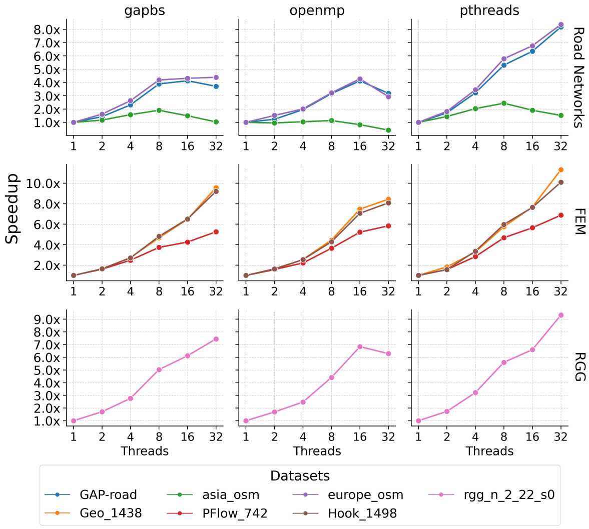
- Experiments run on 3 platforms:
    - AMD EPYC 7543 CPU @ 2.8 GHz (32 cores)
    - Sophon SG2042 RISC-V CPU @ 2.0 GHz (64 cores)
    - NVIDIA Grace CPU Superchip @ up to 3.0 GHz (144 cores)
- Datasets: 3 road networks (USA, Europe, Asia), 3 FEM meshes (Earth's crust, steel hook, porous material), 1 random geometric graph (RGG)
- Tools: GCC compiler, Likwid, SBatchMan
- Compared against the GAP benchmark suite

Chunk size impact on performance

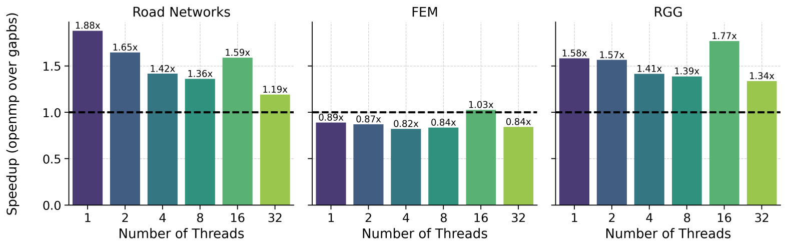
- Chunk size determines the number of vertices in a chunk
- Chunk sizes of 32 and 64 are optimal for most datasets in multithreaded environments



Scalability



Speedup - OpenMP



Speedup of the OpenMP implementation compared to the GAPBS implementation

Optimizing Breadth-First Search on Modern Multicore CPUs

Salvatore D. Andaloro

Introduction

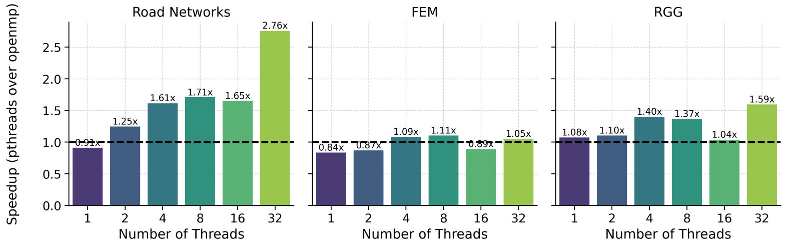
OpenMP

Pthreads

Results

Conclusions

Speedup - Pthreads



Speedup of the pthreads implementation compared to the OpenMP implementation

Optimizing Breadth-First Search on Modern Multicore CPUs

Salvatore D. Andaloro

Introduction

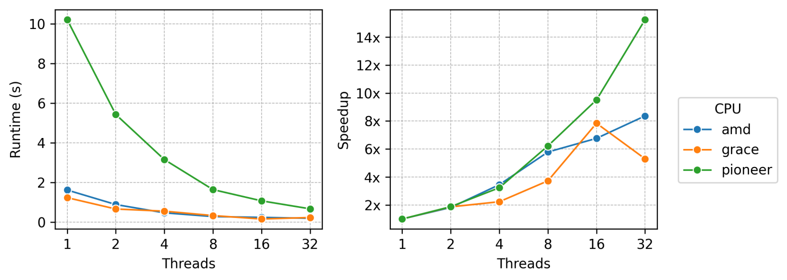
OpenMP

Pthreads

Results

Conclusions

Comparison on different architectures



Execution time and speedup on different architectures for the Europe road network dataset

Conclusions

- Presented a multithreaded implementation of the BFS algorithm using OpenMP and Pthreads
- Compared it on different architectures (x86, RISC-V, ARM) and different datasets
- Achieved $\approx 1.5x$ geomean speedup for OpenMP and $\approx 2x$ speedup for Pthreads compared to the GAP benchmark suite
- Future work: explore other graph algorithms, optimize for more graph types, use different barrier or synchronization primitives

Thank You!

Questions?