Clean Breadth First Search

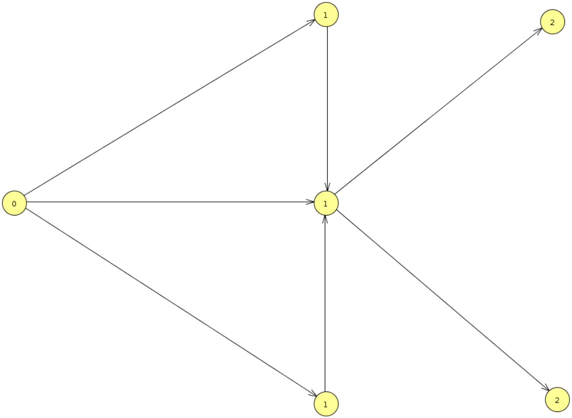
Yonatan R. Fogel

May 16, 2013

BFS Problem Description

- Given
  - Graph $G = (\mathbb{V}, \mathbb{E})$
  - Distinguished source vertex $s \in \mathbb{V}$
- Calculate
  - $Dist_{u \in \mathbb{V}} =$ length of the shortest path from s to u in G
  - *$Parent_{u \in \mathbb{V}} = v \in \mathbb{V}$ s.t. $(v, u) \in \mathbb{E}, Dist_u = Dist_v + 1$

BFS Example



Terms Used in this Presentation

- $n = |\mathbb{V}|$ the number of nodes in a graph
- $m = |\mathbb{E}|$ the number of edges in a graph
- $D = \mathcal{O}(n)$ the diameter of a graph
- Γ_u the set of vertexes adjacent to u
- T_s the running time for a serial algorithm
- T_P the running time for a parallel algorithm running on P cores
- T_∞ the running time for a parallel algorithm running on infinite cores
- $T_1 = \Omega(T_s)$ the running time for a parallel algorithm running on one core
- $W_P = \Omega(T_s)$ the total work done by a parallel algorithm running on P cores (excluding idle time)
  - Reducing W_P can reduce energy use [1]

Serial-BFS

for each vertex $u \in \mathbb{V}$
    $Dist_u \leftarrow \infty$
$Dist_s \leftarrow 0$
$Q \leftarrow \emptyset$
$\textsc{Enqueue}(\ Q,\ \ s\ )$
while $Q \neq \emptyset$ do
    $u \leftarrow \textsc{Dequeue}(\ Q\ )$
    for each vertex v in $\Gamma(u)$ do
        if $Dist_v = \infty$ then
            $Dist_v \leftarrow Dist_u + 1$
            $\textsc{Enqueue}(\ Q, v\ )$

Model of Computation

- Large shared memory
- Consistent caches between cores
- Synchronizing x tasks takes $T_\infty = \Theta(\log x)$ time
- Cilk has this model with randomized work stealing [3]

Motivation for (P)BFS

BFS is used for

- Path Finding
  - Video Games
  - Google Maps
- Analyzing social networks
- Designing and analyzing VLSI
- Task scheduling
- As a primitive in other algorithms

Existing Approaches for PBFS

- Assumes somewhat unrealistic PRAM model
- Specialized for specific hardware [7]
  - GPU
  - CRAY (hardware mutex every 64 bits, atomic add) [2]
- Uses atomic instructions[5]
- Specialized for sparse (dense) graphs only
- Specialized for bounded out-degree (Not scale-free) [4]
- T_1 or T_p is not asymptotically optimal
- Room for energy efficiency improvements (non-optimal W_P)
- Offloads some work to scheduler
  - Work-stealing (randomized) gives at best high probability bounds [4]
- Non level-synchronous[6]

Existing Algorithms

- MIT-bag [4] uses penants, bags, and reducer hyperobjects

Existing Algorithms

- MIT-bag [4] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [5] allocates space from FIFO in blocks

Existing Algorithms

- MIT-bag [4] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [5] allocates space from FIFO in blocks
- distinguished-bfs [6] contracts the graph to make it dense

Existing Algorithms

- MIT-bag [4] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [5] allocates space from FIFO in blocks
- distinguished-bfs [6] contracts the graph to make it dense
- cray-bfs [2] uses fast hardware mutexes and atomic increments

Level-Synchronous BFS

- All nodes at distance d from s are processed before any nodes at distance $d' > d$.
- $T_P = \Omega(n/p + m/p + D \log P)$
  - $T_p = \Omega(T_s/P) = \Omega(n/p + m/p)$
  - Let n_ℓ, m_ℓ be the number of nodes and edges visited at level ℓ.
  - Consider a graph where $\forall_{0<\ell\leq D} n_\ell + m_\ell = \Theta(P)$
  - Every level has $\Theta(P)$ work and uses $P_\ell \leq P$ cores
  - For each level, $T_p = \Omega(P/P_\ell + \log P_\ell) = \Omega(\log P)$ time

Bottlenecks for Parallelizing BFS

- FIFO
- *Dist* array

Clean BFS - Properties

- $T_s = \mathcal{O}\left(n + m\right)$
- $T_1 = \mathcal{O}\left(n + m\right)$
- $W_P = \mathcal{O}\left(n + m\right)$
- $T_P = \mathcal{O}\left(n/p + m/p + D \log P\right)$
- Scale-free
- Deterministic worst case bounds

Clean BFS - Approach

- Evenly split edges among cores using prefix sum and binary search
- Combine output queues using prefix sum

CBFS - Prepare to Split Work

- One input queue $Q_{in} \subseteq \mathbb{V}$.
  - Each vertex in Q_{in} is unique*
  - $\forall_{u \in Q_{in}} Dist_u = \ell$
  - $\forall_{u \in Q_{in}} |\Gamma_u| > 0$
- Generate $OutDegrees[0 \leq i < |Q_{in}|] = |\Gamma(Q_{in}[i])|$ in parallel
- Perform a parallel prefix sum* on $OutDegrees$
$Q_{in} =$	1	3	2	4	...	...
$OutDegrees_{before} =$	1	3	2	4	...	...
$OutDegrees_{after} =$	1	4	6	10	...	m_ℓ

CBFS - Split Work and Process Edges

- Each core i processes m_ℓ / P edges
  - searches *OutDegrees* for $1 + \left\lfloor \frac{i \cdot m_\ell}{P} \right\rfloor$ to find starting edge
  - does $\mathcal{O}\left(\log \frac{n_\ell}{P} + \log P\right)$ work*
  - processes $\left\lfloor \frac{m_\ell}{P} \right\rfloor$ consecutive edges
    - $Q_i \leftarrow \emptyset$
    - for each edge (u, v)
    - if $Dist_v = \infty$ then
    - $Dist_v \leftarrow Dist_u + 1$
    - $Owner_v \leftarrow i$
    - $\textsc{Enqueue}(\ Q_i, v\ )$
  - Benign race conditions

CBFS - Dedup Vertexes and Combine Queues

- $Size_{-1} = 0$
- Each core i uses *Owner* to ensure each vertex lives in at most one output queue
  - $Q_i \leftarrow \{u \in Q_i : Owner_u = i\}$
  - $Size_i \leftarrow |Q_i|$
- Perform a parallel prefix sum* on *Size*
- Each core i copies its queue back into Q_{in} at offset $Size_{i-1}$

CBFS - Reducing W_P for binary searches

- $N \leftarrow |OutDegrees|$
- Each core i
  - $FirstDegree \leftarrow OutDegrees[\lfloor \frac{iN}{P} \rfloor - 1]$
  - $FirstDegreeNext \leftarrow OutDegrees[\left\lfloor \frac{(i+1)N}{P} \right\rfloor - 1]$
  - $FirstCore \leftarrow \left\lceil \frac{P \cdot FirstDegree}{m_\ell} \right\rceil$
  - $LastCore \leftarrow \left\lceil \frac{P \cdot FirstDegreeNext}{m_\ell} \right\rceil$
  - parallel for $j \leftarrow FirstCore$ to $LastCore$
  - $\quad SubList_j \leftarrow i$
- Using $SubList_i$, core i can search only n_ℓ / p indexes
- W_P goes from $\mathcal{O}(n + m + DP \log P)$ to $\mathcal{O}(n + m + DP)$

CBFS - Reducing W_P for parallel prefix sums

- Immediately after calculating m_ℓ
- $P_\ell \leftarrow \min(m_\ell, P)$
- Use at most P_ℓ cores until next time m_ℓ is calculated.
- This ensures the $\mathcal{O}(P_\ell)$ work every level is $\mathcal{O}(m_\ell)$ and can be absorbed into the constant.
- W_P goes from $\mathcal{O}(n + m + DP)$ to $\mathcal{O}(n + m + D) = \mathcal{O}(n + m)$

Future Work

- Optimize CBFS for the PRAM model
  - CBFS runs in same time for PRAM but is not asymptotically optimal
- Modify CBFS to remove false sharing

References I

Susanne Albers and Antonios Antoniadis.
Race to idle: New algorithms for speed scaling with a sleep state.

David A. Bader and Kamesh Madduri.
Designing multithreaded algorithms for breadth-first search and st-connectivity on the cray mta-2.
In *Proceedings of the 2006 International Conference on Parallel Processing*, ICPP '06, pages 523–530, Washington, DC, USA, 2006. IEEE Computer Society.

Matteo Frigo, Pablo Halpern, Charles E. Leiserson, and Stephen Lewin-Berlin.
Reducers and other cilk++ hyperobjects.
In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 79–90, New York, NY, USA, 2009. ACM.

References II

Charles E. Leiserson and Tao B. Schardl.
A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers).
In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, SPAA '10, pages 303–314, New York, NY, USA, 2010. ACM.

Erik Saule and Ümit V. Catalyurek.
An early evaluation of the scalability of graph algorithms on the intel mic architecture.
In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 1629–1639, Washington, DC, USA, 2012. IEEE Computer Society.

References III

J. Ullman and M. Yannakakis.
High-probability parallel transitive closure algorithms.
In *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, SPAA '90, pages 200–209, New York, NY, USA, 1990. ACM.

Andy Yoo, Edmond Chow, Keith Henderson, William McLendon, Bruce Hendrickson, and Umit Catalyurek.
A scalable distributed parallel breadth-first search algorithm on bluegene/l.
In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 25–, Washington, DC, USA, 2005. IEEE Computer Society.