Clean Breadth First Search

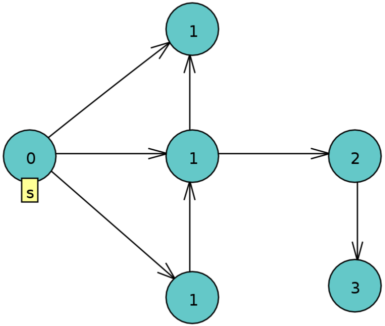
Yonatan R. Fogel

May 16, 2013

BFS Problem Description

- Given
  - Graph $G = (\mathbb{V}, \mathbb{E})$
  - 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



Terminology

- $n = |\mathbb{V}|$ 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
- $\Gamma_u =$ 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
- $T_\infty =$ 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 [Albers and Antoniadis,]

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

            $Dist_v \leftarrow Dist_u + 1$

            $\textsc{Enqueue}(\ Q, v\ )$

Computation Model

- Large shared memory
- Consistent caches between cores
- Synchronizing y tasks takes $T_\infty = \Theta(\log y)$ time
- Cilk has this model with randomized work stealing [Frigo et al., 2009]

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 Algorithms

- MIT-bag [Leiserson and Schardl, 2010] uses penants, bags, and reducer hyperobjects

Existing Algorithms

- MIT-bag [Leiserson and Schardl, 2010] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [Saule and Catalyurek, 2012] allocates space from FIFO in blocks

Existing Algorithms

- MIT-bag [Leiserson and Schardl, 2010] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [Saule and Catalyurek, 2012] allocates space from FIFO in blocks
- distinguished-bfs [Ullman and Yannakakis, 1990] contracts the graph to make it dense

Existing Algorithms

- MIT-bag [Leiserson and Schardl, 2010] uses penants, bags, and reducer hyperobjects
- block-queue-bfs [Saule and Catalyurek, 2012] allocates space from FIFO in blocks
- distinguished-bfs [Ullman and Yannakakis, 1990] contracts the graph to make it dense
- cray-bfs [Bader and Madduri, 2006] uses fast hardware mutexes and atomic increments

General Approaches

- Assumes PRAM model
- Specialized for specific hardware [Yoo et al., 2005]
  - GPU
  - CRAY (hardware mutex every 64 bits, atomic add) [Bader and Madduri, 2006]
- Uses atomic instructions [Saule and Catalyurek, 2012]
- Specialized for sparse (or dense) graphs only
- Specialized for bounded out-degree (not scale-free) [Leiserson and Schardl, 2010]
- 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 high probability bounds [Leiserson and Schardl, 2010]
- Non level-synchronous [Ullman and Yannakakis, 1990]

Level-Synchronous BFS

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

Clean BFS - Properties

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

Clean BFS - High Level

For each level $\ell \in [0 \ldots D)$

1. To get optimal T_1, T_P
   1.1 Prepare to Split Work
   1.2 Split Work and Process Edges
   1.3 Dedup Vertexes and Combine Queues
2. To get optimal W_P
   2.1 Reduce Search Space
   2.2 Dynamically Choose Number of Cores

Clean BFS - Prepare to Split Work

1. One input queue $Q_{in} \subseteq \mathbb{V}$
   1.1 Each vertex in Q_{in} is unique
   1.2 $\forall u(u \in Q_{in} \Rightarrow Dist_u = \ell)$
   1.3 $\forall u(u \in Q_{in} \Rightarrow |\Gamma_u| > 0$
2. Generate $OutDegrees[0 \leq i < |Q_{in}|] = |\Gamma(Q_{in}[i])|$ in parallel
3. 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

Clean BFS - Split Work and Process Edges

1. Each core i processes m_ℓ / P edges
   1.1 searches *OutDegrees* for $1 + \left\lfloor \frac{i\ m_\ell}{P} \right\rfloor$ to find starting edge
   1.2 does $\mathcal{O}\left(\log \frac{n_\ell}{P} + \log P\right)$ work
   1.3 processes $\left\lfloor \frac{m_\ell}{P} \right\rfloor$ consecutive edges
      1.3.1 $Q_i \leftarrow \emptyset$
      1.3.2 for each edge (u, v)
      1.3.3   if $Dist_v = \infty$ then
      1.3.4 $Dist_v \leftarrow Dist_u + 1$
      1.3.5 $Owner_v \leftarrow i$
      1.3.6 $\textsc{Enqueue}(\ Q_i, v\ )$
   1.4 Benign race conditions

Clean BFS - Dedup Vertexes and Combine Queues

1. $Size_{-1} = 0$
2. Each core i uses *Owner* to ensure each vertex lives in at most one output queue
   2.1 $Q_i \leftarrow \{u \in Q_i : Owner_u = i\}$
   2.2 $Size_i \leftarrow |Q_i|$
3. Perform a parallel prefix sum on *Size*
4. Each core i copies its queue back into Q_{in} at offset $Size_{i-1}$

Clean BFS - Reduce Search Space

1. $N \leftarrow |OutDegrees|$
2. Each core i
   2.1 $FirstDegree \leftarrow OutDegrees[\lfloor \frac{iN}{P} \rfloor - 1]$
   2.2 $FirstDegreeNext \leftarrow OutDegrees[\left\lfloor \frac{(i+1)N}{P} \right\rfloor - 1]$
   2.3 $FirstCore \leftarrow \left\lceil \frac{P \;\; FirstDegree}{m_\ell} \right\rceil$
   2.4 $LastCore \leftarrow \left\lceil \frac{P \;\; FirstDegreeNext}{m_\ell} \right\rceil$
   2.5 parallel for $j \leftarrow FirstCore$ to $LastCore$
   2.6 $\quad SubList_j \leftarrow i$
3. Using $SubList_i$, core i can search only n_ℓ / p indexes
4. W_P reduces from $\mathcal{O}\left(n + m + DP \log P\right)$ to $\mathcal{O}\left(n + m + DP\right)$

Clean BFS - Dynamically Choose Number of Cores

- In "Prepare to Split Work", right after the parallel prefix sum, $P_\ell \leftarrow \textsc{min}(\ m_\ell, P\ )$
- Use at most P_ℓ cores until next time m_ℓ is calculated
- This ensures the $\mathcal{O}(P_\ell)$ work every level is $\mathcal{O}(m_\ell)$
- W_P reduces from $\mathcal{O}(n + m + DP)$ to $\mathcal{O}(n + m + D) = \mathcal{O}(n + m)$ (optimal)

Conclusions

- Clean BFS has optimal T_1, T_P, W_P in our computation model
- TODO MORE?

Future Work

- Optimize Clean BFS for the PRAM model
  - Clean BFS runs in same time for PRAM but is not asymptotically optimal
  - Try using approximate parallel prefix sum [Goldberg and Zwick, 1995]
- Modify Clean BFS to remove false sharing
  - Examine using an $\mathcal{O}(n)$ sort algorithm for levels where $m_\ell = \Omega n^{\frac{1}{c}}$
  - Try to distribute work by cacheline
- Examine non level-synchronous approaches

References I

Albers, S. and Antoniadis, A.
Race to idle: New algorithms for speed scaling with a sleep state.

Bader, D. A. and Madduri, K. (2006).
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. IEEE Computer Society.

Frigo, M., Halpern, P., Leiserson, C. E., and Lewin-Berlin, S. (2009).
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. ACM.

References II

Goldberg, T. and Zwick, U. (1995).
Optimal deterministic approximate parallel prefix sums and their applications.
In *In Proc. Israel Symp. on Theory and Computing Systems (ISTCS'95*, pages 220–228.

Leiserson, C. E. and Schardl, T. B. (2010).
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. ACM.

References III

Saule, E. and Catalyurek, U. V. (2012).
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. IEEE Computer Society.

Ullman, J. and Yannakakis, M. (1990).
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. ACM.

References IV

Yoo, A., Chow, E., Henderson, K., McLendon, W., Hendrickson, B., and Catalyurek, U. (2005).
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. IEEE Computer Society.