

# A new optimal algorithm for Level-Synchronous Parallel BFS in the Cilk model

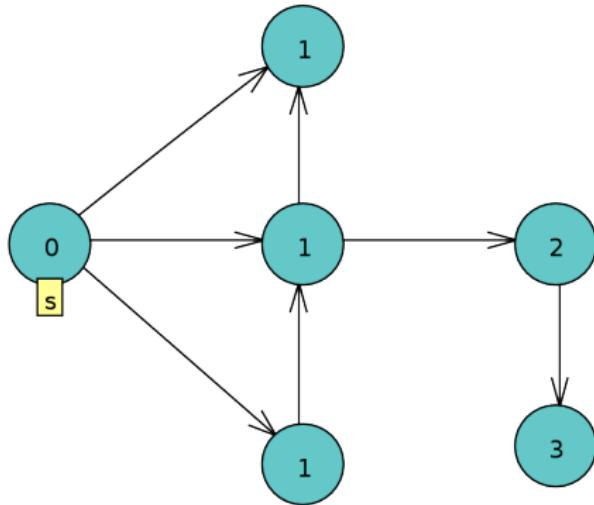
Yonatan R. Fogel

Stony Brook University

May 17, 2013

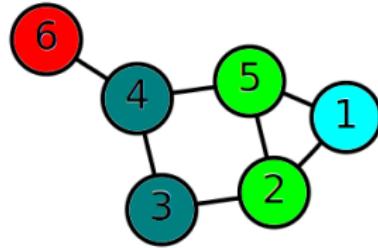
# BFS

- ▶ Given
  - ▶ Graph  $G = (\mathbb{V}, \mathbb{E})$  with diameter  $D$  and max out-degree  $\Delta$
  - ▶ Source vertex  $s \in \mathbb{V}$
- ▶ Calculate
  - ▶  $Dist_{u \in \mathbb{V}} = \text{length of the shortest path from } s \text{ to } u \text{ in } G$



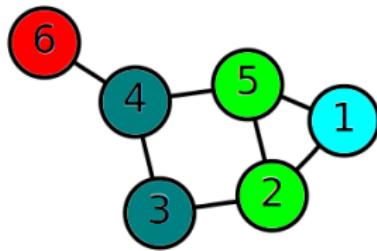
## Parallel BFS

- ▶ The runtime for serial BFS is  $T_s = \Theta(|V| + |E|)$
- ▶ For large graphs, calculating  $Dist$  can be slow
- ▶ We can speed up BFS by processing edges and/or nodes in parallel
- ▶ Our new TBN-BFS achieves near-perfect linear speedup when  $p \ll \frac{|V|+|E|}{D \log(|V|+|E|)}$



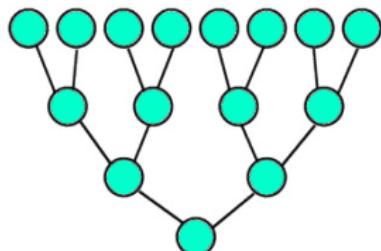
## Level-Synchronous BFS

- ▶ All nodes at distance  $d$  from  $s$  are processed before any nodes at distance  $d' > d$
  - ▶ Level-Synchronous BFS has lower bounds of
    - ▶  $T_p = \Omega\left(\frac{|\mathbb{V}|+|\mathbb{E}|}{p} + D\right)$
    - ▶  $T_1, W_p = \Omega(T_s) = \Omega(|\mathbb{V}| + |\mathbb{E}|)$
  - ▶ Non Level-Synchronous BFS algorithms
    - ▶ Distinguished-BFS [Ullman and Yannakakis, 1990]



# Cilk Model

- ▶ Large shared memory
- ▶ Consistent caches between cores
- ▶ Synchronizing (or launching)  $n$  tasks takes  $T_\infty = \Theta(\log n)$  time
- ▶ Cilk+ has this model with the addition of randomized work stealing [Frigo et al., 2009]
- ▶ A parallel algorithm has the following properties
  - ▶  $T_p$  = running time using  $p$  cores
  - ▶  $T_1$  = running time using 1 core
  - ▶  $W_p$  = work done using  $p$  cores



## Non Cilk Model

- ▶ Level Synchronous BFS algorithms not using the Cilk Model
    - ▶ Cray-BFS uses the Cray MTA-2 hardware model  
[Bader and Madduri, 2006]
    - ▶ Block-Queue-BFS uses the Intel Mic hardware model  
[Saule and Catalyurek, 2012]
    - ▶ Blue-BFS uses the BlueGene/L hardware model  
[Yoo et al., 2005]



# Level-Synchronous Parallel BFS in the Cilk Model

- ▶ In the Cilk Model, Level-Synchronous BFS has a lower bound of  $T_p = \Omega\left(\frac{|\mathbb{V}|+|\mathbb{E}|}{p} + D \log p\right)$
- ▶ MIT-Bag-BFS uses penants, bags, and reducer hyperobjects [Leiserson and Schardl, 2010]
- ▶ Our new TBN-BFS algorithm
  - ▶ Runs on standard hardware implemented using Cilk+
  - ▶ Can do its own scheduling deterministically
  - ▶ Achieves the lower bound for  $T_p, T_1, W_p$  in the worst case independently of  $\Delta$  (scale-free)



## Serial-BFS

1. for each vertex  $u \in \mathbb{V}$
  2.      $Dist_u \leftarrow \infty$
  3.      $Dist_s \leftarrow 0$
  4.      $Q \leftarrow \emptyset$
  5.     ENQUEUE( $Q, s$ )
  6.     while  $Q \neq \emptyset$  do
    7.          $u \leftarrow \text{DEQUEUE}(Q)$  {Potential source of parallelism: lines 7–8}
    8.         for each vertex  $v$  in  $\Gamma(u)$  do
      9.             if  $Dist_v = \infty$  then {Potential issues for parallelism: lines 9–11}
        10.                  $Dist_v \leftarrow Dist_u + 1$
        11.                 ENQUEUE( $Q, v$ )

# TBN-BFS - Summary

- ▶  $T_1 = \mathcal{O}(n + m)$  (optimal)
- ▶  $W_p = \mathcal{O}(n + m)$  (optimal)
- ▶  $T_p = \mathcal{O}(n/p + m/p + D \log p)$  (optimal)
- ▶ Scale-free
- ▶ Runs on standard hardware implemented using Cilk+
- ▶ Can do its own scheduling deterministically



# TBN-BFS - High Level

- ▶ To get optimal  $T_1, T_p$  the algorithm is
  1. For each level  $\ell \in [0 \dots D)$ 
    - 1.1 Prepare to split work
    - 1.2 Split work and process edges
    - 1.3 Deduplicate vertexes and combine queues
- ▶ To get optimal  $W_p$  modify the algorithm to
  1. Reduce search space
  2. Dynamically choose number of cores



# TBN-BFS - Prepare to Split Work

1. Receive input  $Q_{in} \subseteq \mathbb{V}$  where
  - 1.1 Each vertex in  $Q_{in}$  is unique
  - 1.2  $\forall u \in Q_{in} : Dist_u = \ell$
  - 1.3  $\forall u \in Q_{in} : |\Gamma(u)| > 0$
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	...	...
$OutDegrees_{before} =$	5	7	1	2	...	...
$OutDegrees_{after} =$	5	12	13	15	...	$m_\ell$



# TBN-BFS - Split Work and Process Edges

## 1. Each core $i$ in parallel

1.1 searches  $OutDegrees$  for  $1 + \left\lfloor \frac{i \cdot m_\ell}{p} \right\rfloor$  to find starting edge

$$\left\{ \mathcal{O}\left(\log \frac{n_\ell}{p} + \log p\right) \text{ work}\right\}$$

1.2 processes  $\left\lfloor \frac{m_\ell}{p} \right\rfloor$  consecutive edges

1.2.1  $Q_i \leftarrow \emptyset$

1.2.2 for each edge  $(u, v)$

1.2.3 if  $Dist_v = \infty$  then

{Benign race condition}

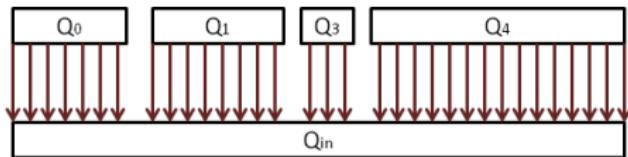
1.2.4  $Dist_v \leftarrow Dist_u + 1$

1.2.5  $Owner_v \leftarrow i$

1.2.6 ENQUEUE( $Q_i, v$ )

## TBN-BFS - Deduplicate Vertices and Combine Queues

1. Each core  $i$  in parallel
    - 1.1  $Q_i \leftarrow \{u \in Q_i : Owner_u = i\}$  {Remove duplicate vertexes}
    - 1.2  $Size_i \leftarrow |Q_i|$
  2. Perform a parallel prefix sum on  $Size$
  3. Each core  $i$  copies its queue back into  $Q_{in}$  at offset  $Size_{i-1}$   
 $\{Size_{-1}$  is treated as 0 $\}$

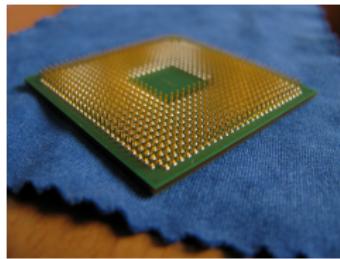


# TBN-BFS - Reduce Search Space

1.  $N \leftarrow |OutDegrees|$
2. Each core  $i$ 
  - 2.1  $FirstDegree \leftarrow OutDegrees[\left\lfloor \frac{iN}{p} \right\rfloor - 1]$
  - 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  $SubList_i \leftarrow i$
3. Using  $SubList_i$ , core  $i$  can search only  $n_\ell/p$  indexes
4.  $W_p$  reduces from  $\mathcal{O}(|\mathbb{V}| + |\mathbb{E}| + DP \log p)$  to  $\mathcal{O}(|\mathbb{V}| + |\mathbb{E}| + DP)$

## TBN-BFS - Dynamically Choose Number of Cores

- ▶ In “Prepare to Split Work”, right after the parallel prefix sum,  
 $p_\ell \leftarrow \text{MIN}(m_\ell, p)$
- ▶ Use at most  $p_\ell$  cores until next time  $m_\ell$  is calculated
- ▶ This ensures the  $\mathcal{O}(p_\ell)$  work every level is  $\mathcal{O}(m_\ell)$
- ▶  $W_p$  reduces from  $\mathcal{O}(|\mathbb{V}| + |\mathbb{E}| + DP)$  to  
 $\mathcal{O}(|\mathbb{V}| + |\mathbb{E}| + D) = \mathcal{O}(|\mathbb{V}| + |\mathbb{E}|)$  (optimal)



# Summary

- ▶ Our new TBN-BFS algorithm
  - ▶  $T_1 = \mathcal{O}(n + m)$  (optimal)
  - ▶  $W_p = \mathcal{O}(n + m)$  (optimal)
  - ▶  $T_p = \mathcal{O}(n/p + m/p + D \log p)$  (optimal)
  - ▶ Scale-free
  - ▶ Runs on standard hardware implemented using Cilk+
  - ▶ Can do its own scheduling deterministically
  - ▶ Preliminary experiments show merit



## Future Work

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

## References I

-  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.