Parallel Depth-First Search on a DAG

Divyanshu Talwar[1] Viraj Parimi [2]

[1]2015028

[2]2015068

Course Project - GPU Computing, Winter 2018

Outline

Outline

1. Problem Definition
2. Computational Complexity
3. Problem Subdivisions
4. Analysis of the Problem
5. Performance Comparision Plan
6. Deliverables

Problem Definition

- Let a graph $G = (V, E)$, be defined by its vertex $V = \{1, 2, .., n\}$ and edges $E = \{(i_1, j_1), (i_2, j_2), .., (i_m, j_m)\}$ sets, with $|V| = n$ and $|E| = m$.

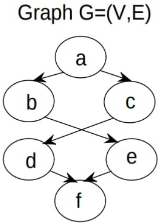

Figure: 1

Problem Definition

- Let a graph $G = (V, E)$, be defined by its vertex $V = \{1, 2, .., n\}$ and edges $E = \{(i_1, j_1), (i_2, j_2), .., (i_m, j_m)\}$ sets, with $|V| = n$ and $|E| = m$.
- Lexicographic Depth-First Search (DFS) traversal problem requires computation of parent information , pre-order (start time) and post-order (end time) for every node in G.

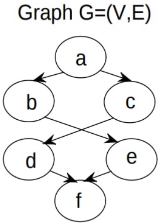

Figure: 1

$$
\begin{aligned}
\text{node} &= \{a, b, c, d, e, f\} \\
\text{pre-order} &= \{0, 1, 4, 5, 2, 3\} \\
\text{post-order} &= \{5, 2, 4, 3, 1, 0\} \\
\text{parent} &= \{\emptyset, a, a, c, b, e\}
\end{aligned}
$$

Figure: 2

Outline

Computational Complexity

- `DFS` traversal - in a general sense - is *P*-complete where class *P*, typically consists of all the "tractable" problems for a sequential computer.

Computational Complexity

- `DFS` traversal - in a general sense - is *P*-complete where class *P*, typically consists of all the "tractable" problems for a sequential computer.
- `DFS` for DAGs $\in NC$ class, where the class NC (for "Nick's Class") is the set of decision problems decidable in poly-logarithmic ($O(log^{\alpha} n)$ for some constant α) time on a parallel computer with a polynomial number of processors.

Outline

Outline

Definition 1

Let ς_p and ζ_p denote the number of nodes reachable under and including node p, where if a sub-graph is reachable from k multiple parents then its nodes are counted once and k times, respectively.

- For example, in Fig. 1 we have $\zeta_a = 7$ and $\varsigma_a = 6$, because we double counted the node f in the former case.

Definition 1

Let ς_p and ζ_p denote the number of nodes reachable under and including node p, where if a sub-graph is reachable from k multiple parents then its nodes are counted once and k times, respectively.

- For example, in Fig. 1 we have $\zeta_a = 7$ and $\varsigma_a = 6$, because we double counted the node f in the former case.
- Also, notice the recursive relationship :

$$\zeta_p = 1 + \sum_{i \in C_p} \zeta_i$$

where C_p is ordered set of children of p.

Definition 2

Let $\tilde{\zeta}_l$ where l is an index of exclusive prefix sum list, of the list ζ_i, where i $\in C_p$.

$$\tilde{\zeta}_l = \sum_{i<l, i \in C_p} \zeta_i$$

- For example, in Fig. 1 we have $\tilde{\zeta}_b = 0$ and $\tilde{\zeta}_c = 3$.

Definition 3

Let us define a directed tree (DT) to be a DAG, where every node has a single parent.

Problem Subdivision

Pre-Order and Post-Order Time

Algorithm 2 Sub-Graph Size (bottom-up traversal)

Initialize all sub-graph sizes to 0.
Find leafs and insert them into queue Q.
while $Q \neq \{\emptyset\}$ **do**
  for node $i \in Q$ **do in parallel**
    Let P_i be a set of parents of i and queue $C = \{\emptyset\}$
    for node $p \in P_i$ **do in parallel**
      Mark p outgoing edge (p, i) as visited
      Insert p into C if all outgoing edges are visited
    end for
  end for
  for node $p \in C$ **do in parallel**
    Let C_p be an ordered set of children of node p
    Compute a prefix-sum on C_p, obtaining ζ_p
    (use lexicographic ordering of elements in C_p)
  end for
  Set queue $Q = C$ for the next iteration
end while

Problem Subdivision

Pre-Order and Post-Order Time

- Notice that for DT, the sub-graph size at node $p = \zeta_p$.

Problem Subdivision

Pre-Order and Post-Order Time

- Notice that for DT, the sub-graph size at node $p = \zeta_p$.

Definition 4

Let a path from root r to node p be an ordered set of nodes $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$, where k is the depth of the node p.

- Notice that for DT, the sub-graph size at node $p = \zeta_p$.

Definition 4

Let a path from root r to node p be an ordered set of nodes $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$, where k is the depth of the node p.

Theorem

Let ζ_i be the sub-graph size for node i in a DT and $\tilde{\zeta}_l$ be the corresponding prefix-sum value. Then,

$$\texttt{preorder}(p) = k + \tau_p \tag{1}$$

$$\texttt{postorder}(p) = (\zeta_p - 1) + \tau_p \tag{2}$$

where $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$ *and,* $\tau_p = \sum_{l \in \mathfrak{P}_{r,p}} \zeta_l$

Problem Subdivision

Pre-Order and Post-Order Time

Algorithm Pre- and Post-Order (top-down traversal)

Initialize pre and post-order of every node to 0.
Find roots and insert them into queue Q.
while $Q \neq \{\emptyset\}$ **do**
  for node $p \in Q$ **do in parallel**
    Let pre = pre-order(p)
    Let post= post-order(p)
    Let C_p be a set of children of p and queue $P = \{\emptyset\}$
    for node $i \in C_p$ **do in parallel**
      Set pre-order(i) = pre + $\widetilde{\zeta_i}$
      Set post-order(i)= post+ ζ_i
      Mark i incoming edge (p, i) as visited
      Insert i into P if all incoming edges are visited
    end for
    Set pre-order(p) = pre + depth(p)
    Set post-order(p)= post+ ζ_p
  end for
  Set queue $Q = P$ for the next iteration
end while

Outline

Definition 5

Let $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$ and $\mathfrak{Q}_{r,p} = \{r, j_1, j_2, .., j_{k-1}, p\}$ be two paths of potentially different length to node p. We say that path $\mathfrak{P}$ has the first lexicographically smallest node and denote it by

$$\mathfrak{P}_{\mathfrak{r},\mathfrak{p}} < \mathfrak{Q}_{\mathfrak{r},\mathfrak{p}} \tag{3}$$

when during the pair-wise comparison of the elements in the two paths going from left-to-right the path $\mathfrak{P}_{\mathfrak{r},\mathfrak{p}}$ has the lexicographically smallest element in the first mismatch.

Definition 5

Let $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$ and $\mathfrak{Q}_{r,p} = \{r, j_1, j_2, .., j_{k-1}, p\}$ be two paths of potentially different length to node p. We say that path $\mathfrak{P}$ has the first lexicographically smallest node and denote it by

$$\mathfrak{P}_{\mathfrak{r},\mathfrak{p}} < \mathfrak{Q}_{\mathfrak{r},\mathfrak{p}} \tag{3}$$

when during the pair-wise comparison of the elements in the two paths going from left-to-right the path $\mathfrak{P}_{\mathfrak{r},\mathfrak{p}}$ has the lexicographically smallest element in the first mismatch.

For example, in Fig. 1 the two paths to node f are

$$\mathfrak{P}_{\mathfrak{r},\mathfrak{p}} = [a, b, e, f]$$

$$\mathfrak{Q}_{\mathfrak{r},\mathfrak{p}} = [a, c, d, f]$$

Problem Subdivision

DAG to Directed Tree

Theorem

Let $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$ *and* $\mathfrak{Q}_{r,p} = \{r, j_1, j_2, .., j_{k-1}, p\}$ *be two paths of potentially different length to node p. If* $\mathfrak{P}_{r,p} < \mathfrak{Q}_{r,p}$ *then* $\mathfrak{P}_{\mathfrak{r},\mathfrak{p}}$ *is the path taken by DFS traversal.*

Theorem

Let $\mathfrak{P}_{r,p} = \{r, i_1, i_2, .., i_{k-1}, p\}$ *and* $\mathfrak{Q}_{r,p} = \{r, j_1, j_2, .., j_{k-1}, p\}$ *be two paths of potentially different length to node p. If* $\mathfrak{P}_{r,p} < \mathfrak{Q}_{r,p}$ *then* $\mathfrak{P}_{\mathfrak{r},\mathfrak{p}}$ *is the path taken by DFS traversal.*

Corollary

Let $\mathfrak{G}$ be the set of all paths from root r to node p. The DFS traversal takes

$$\mathfrak{P}_{\mathfrak{r},\mathfrak{p}} = \min_{\mathfrak{Q}_{r,p} \in \mathfrak{G}} \mathfrak{Q}_{r,p} \tag{4}$$

Problem Subdivision

DAG to Directed Tree

Algorithm Compute DFS-Parent by Comparing Path (top-down traversal)

Initialize path to $\{\emptyset\}$ and parent to -1 for every node.
Find roots and insert them into queue Q.
while $Q \neq \{\emptyset\}$ **do**
    for node $p \in Q$ **do in parallel**
        Let C_p be a set of children of p and queue $P = \{\emptyset\}$
        for node $i \in C_p$ **do in parallel**
            Let the existing path be $\mathfrak{Q}_{r,i}$
            Let the new path be $\mathfrak{P}_{r,i}$
            ($\mathfrak{P}_{r,i}$ is a concatenation of path to p & node i)
            if $\mathfrak{P}_{r,i} \leq \mathfrak{Q}_{r,i}$ **then**
                Set $\mathfrak{Q}_{r,i} = \mathfrak{P}_{r,i}$
                Set $\text{parent}(i) = p$
            end if
            Mark i incoming edge (p, i) as visited
            Insert i into P if all incoming edges are visited
        end for
    end for
    Set queue $Q = P$ for the next iteration
end while

Outline

Analysis of the Problem

The aforementioned parallel algorithm to compute the DFS traversal of a DAG is work-efficient.

- Parallel prefix-sum can be computed in $O(logn)$, by doing $O(n)$ work.
- The parallel sorting can be computed in $O(logn)$, by doing $O(nlogn)$ work.

Analysis of the Problem

The aforementioned parallel algorithm to compute the DFS traversal of a DAG is work-efficient.

- Parallel prefix-sum can be computed in $O(logn)$, by doing $O(n)$ work.
- The parallel sorting can be computed in $O(logn)$, by doing $O(nlogn)$ work.

Lemma 1

Let $n = min(n_1, n_2)$, then identifying the first left-to-right pair of digits in two sequences of n_1 and n_2 numbers can be performed in $O(logn)$ steps, by doing $O(n)$ work.

Analysis of the Problem

The aforementioned parallel algorithm to compute the DFS traversal of a DAG is work-efficient.

- Parallel prefix-sum can be computed in $O(logn)$, by doing $O(n)$ work.
- The parallel sorting can be computed in $O(logn)$, by doing $O(nlogn)$ work.

Lemma 1

Let $n = min(n_1, n_2)$, then identifying the first left-to-right pair of digits in two sequences of n_1 and n_2 numbers can be performed in $O(logn)$ steps, by doing $O(n)$ work.

Lemma 2

The queue can be implemented such that parallel insertion and extraction of n numbers, can be performed in $O(logn)$ and $O(1)$ steps, respectively. Also, the algorithm performs $O(n)$ work.

Analysis of the Problem

Theorem

Alg.2 takes $O(\eta(logd + logk))$ steps and performs $O(m + n)$ total work to traverse a DAG. The number of processors $t \leq m + n$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG, d is maximum degree in DAG and k is the maximum number of elements inserted into a queue.

Analysis of the Problem

Theorem

Alg.2 takes $O(\eta(logd + logk))$ steps and performs $O(m + n)$ total work to traverse a DAG. The number of processors $t \leq m + n$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG, d is maximum degree in DAG and k is the maximum number of elements inserted into a queue.

Theorem

Alg.3 takes $O(\eta logk)$ steps and performs $O(n)$ total work to traverse a DAG. The number of processors $t \leq n$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG and k is the maximum number of elements inserted into a queue.

Analysis of the Problem

Theorem

Alg.4 takes $O(\eta(log\eta + logk))$ steps and performs $O(\eta m + n)$ total work to traverse a DAG. The number of processors $t \leq \eta d + n$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG, d is maximum degree in DAG and k is the maximum number of elements inserted into a queue.

Analysis of the Problem

Theorem

Alg.4 takes $O(\eta(log\eta + logk))$ steps and performs $O(\eta m + n)$ total work to traverse a DAG. The number of processors $t \leq \eta d + n$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG, d is maximum degree in DAG and k is the maximum number of elements inserted into a queue.

Corollary

Path based DFS takes $O(\eta(logd + logk + log\eta))$ steps and performs $O(m + n + \eta m)$ total work to traverse a DAG. The number of processors $t \leq m + n + \eta d$ actively doing work varies at each step of the algorithm. Here η is the length of longest path in DAG and k is the maximum number of elements inserted into a queue.

Outline

Performance Comparision Plan

- Since, there aren't any existing codes available, thus, we would be comparing our implementation with a serial implementation of DFS traversal.

Outline

Deliverables

- One
  - Serial Implementation
  - Parents
- Two
  - Pre-order and Post-order Time Calculations
- Three
  - Optimizations
    - Path Pruning
    - Path Compression
    - Path Data Structure (as mentioned in the paper).
    - SSSP based DFS (if time permits)

References I

Maxim Naumov, Alysson Vrielink, and Michael Garland
Parallel Depth-First Search for Directed Acyclic Graphs
Technical Report, NVR 2017

Maxim Naumov, Alysson Vrielink, and Michael Garland
Parallel Depth-First Search for Directed Acyclic Graphs
Presentation, GTC Presentations 2017