Parallel Breadth First Search

Yonatan Fogel
Computer Science
Stony Brook University
Stony Brook, New York, 11744
Email: yfogel@cs.sunysb.edu

***Abstract*—This is an $n - 1$ level abstract**

I. Introduction

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

II. Related Work

[1]

III. PPS(RENAME)

Parallel prefix sum

IV. BFS(RENAME)

BFS algo

```
parallel for u gets 0 .. n - 1

parallel for i = 0 to p-1
    if i = 0
        var{offset} = 0
    else
    var{offset} = D[\frac{iN}{p}
    var{offset}
```

V. Analysis

VI. Conclusion

VII. Future Work

Acknowledgment

The authors would like to thank...

References

[1] C. E. Leiserson and T. 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*, ser. SPAA '10. New York, NY, USA: ACM, 2010, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1810479.1810534

SERIAL-PREFIX-SUM($V,\ start,\ limit$)

$V[start] \leftarrow V[start] + partial\_sum$
for $i \leftarrow start + 1$ ***to*** $limit - 1$ ***do***
    $V[i] \leftarrow V[i] + V[i-1]$
return $V[limit - 1]$

PARALLEL-PREFIX-SUM-UP(V, GRAIN-SIZE, $start,\ limit$)

$size \leftarrow limit - start$
if $size \leq$ GRAIN-SIZE ***then***
    return SERIAL-PREFIX-SUM($V,\ start,\ limit$)
else
    $mid \leftarrow \left\lfloor \frac{start+limit}{2} \right\rfloor$
    $x \leftarrow$ ***spawn*** PARALLEL-PREFIX-SUM-UP(V, GRAIN-SIZE, $start,\ mid$)
    $y \leftarrow$ PARALLEL-PREFIX-SUM-UP(V, GRAIN-SIZE, $mid,\ limit$)
    sync
    $V[limit - 1] \leftarrow x + y$
    return $x + y$

SERIAL-PREFIX-SUM-DOWN($V,\ start,\ limit,\ rightmost\_excluded,\ partial\_sum$)

for $i \leftarrow start$ ***to*** $limit - 2$ ***do***
    $V[start] \leftarrow V[start] + partial\_sum$
if $\neg rightmost\_excluded$ ***then***
    $V[limit - 1] \leftarrow V[limit - 1] + partial\_sum$

PARALLEL-PREFIX-SUM-DOWN($V,\ start,\ limit,\ rightmost\_excluded,\ partial\_sum$)

$size \leftarrow limit - start$
if $size \leq$ GRAIN-SIZE ***then***
    SERIAL-PREFIX-SUM-DOWN($V,\ start,\ limit,\ partial\_sum,\ rightmost\_excluded$)
    return
else
    $mid \leftarrow \left\lfloor \frac{start+limit}{2} \right\rfloor$
    $sum\_left \leftarrow V[mid - 1]$
    spawn PARALLEL-PREFIX-SUM-DOWN(V, GRAIN-SIZE, $start,\ mid$, ***false***, $partial\_sum$)
    if $\neg rightmost\_excluded$ ***then***
        $V[limit - 1] \leftarrow V[limit - 1] + partial\_sum$
    if $limit - mid > 1$
        PARALLEL-PREFIX-SUM-DOWN(V, GRAIN-SIZE, $mid,\ limit$, ***true***, $partial\_sum + sum\_left$)

PARALLEL-PREFIX-SUM(V, GRAIN-SIZE)
$V[0 : n-1]$ is a sequence of n integers. This function replaces $V[i]$ with $\sum_{0 \leq j \leq i} V[j]$

if $|V| > 1$ ***then***
    PARALLEL-PREFIX-SUM-UP(V, GRAIN-SIZE, $0,\ n$)
    PARALLEL-PREFIX-SUM-DOWN(V, GRAIN-SIZE, $0,\ n$, ***false***, 0)

Fig. 1. Overwrite $V[0 : n-1]$ with its prefix sum.

```
FIND-SUBLIST( WORK, W, p )
    p_n ← MIN( p, N )
    SUBLIST ← array[0 : p − 1]
    RANGESSTART ← array[0 : p_n − 1]
    RANGESEND ← array[0 : p_n − 1]
    parallel for i ← 0 to p_n − 1 do
        if i = 0 then
            FIRSTDEGREE ← 0
        else
            FIRSTDEGREE ← WORK[⌊iN / p_n⌋ − 1]
        FIRSTDEGREENEXT ← WORK[⌊(i+1)N / p_n⌋ − 1]
        RANGESSTART[i] ← ⌈p_n·FIRSTDEGREE / W⌉
        RANGESEND[i] ← ⌈p_n·FIRSTDEGREENEXT / W⌉ − 1
    parallel for i ← 0 to p_n − 1 do
        if RANGESSTART[i] ≤ RANGESEND[i] then
            parallel for j ← RANGESSTART[i] to RANGESEND[i] do
                SUBLIST[j] ← i
    return SUBLIST
```

```
LEVEL-TO-QUEUES( INPUT, WORK, SUBLIST, D, Γ, γ, W, p, LEVEL )
    Q ← array[0 : p − 1]
    parallel for i ← 0 to p − 1 do
        Q[i].CLEAR( )
        FIRSTDEGREE ← ⌊iW / p⌋
        WORKITEMS ← ⌊(i+1)W / p⌋ − FIRSTDEGREE
        VERTEX ← BINARY-SEARCH-FOR-INDEX( WORK, ⌊N·SUBLIST[i] / p⌋, ⌊N·(SUBLIST[i]+1) / p⌋, FIRSTDEGREE + 1 )
        DEGREE ← FIRSTDEGREE − WORK[VERTEX − 1]
        while WORKITEMS > 0 do
            u ← INPUT[VERTEX]
            LIMIT ← MIN( WORKITEMS + DEGREE, γ[u] )
            for j ← DEGREE to LIMIT do
                v ← Γ[u][j]
                if D[v] = ∞ then
                    D[v] ← LEVEL
                    OWNER[v] ← i
                    if γ[v] > 0 then
                        Q[i].ENQUEUE( v )
            WORKITEMS ← WORKITEMS − DEGREE
            DEGREE ← 0
            VERTEX ← VERTEX + 1
    return Q
```

Fig. 2. Overwrite squirrels with its prefix sum.

PARALLEL-BFS($V,\ \Gamma,\ \gamma,\ s,\ p_{max}$)

$V[0:n-1]$ are the n nodes in the graph. $\Gamma[u]$ is the sequence of adjacent nodes to node u. $\gamma[u] = |\Gamma[u]|$. s is the source vertex from which distance is calculated. p_{max} is the maximum number of processors to use. Returns D$[0:n-1]$ which represents the distance from s to each vertex.

```
parallel for u ← 0 to n − 1 do
    D[u] ← ∞
    OWNER[u] ← ∞
D[s] ← 0
OWNER[s] ← 0
if γ[s] = 0 then
    return D
INPUT ← array[0 : 0]
INPUT[0] ← s
LEVEL ← 0
p ← 1
while |INPUT| ≠ 0 do
    LEVEL ← LEVEL + 1
    N ← |INPUT|
    WORK ← array[0 : N − 1]
    parallel for u ← 0 to N − 1 do
        WORK[u] ← γ[INPUT[u]]
    GRAIN-SIZE ← N/p
    PARALLEL-PREFIX-SUM( WORK, GRAIN-SIZE )
    W ← D[N − 1]
    p ← MIN( p_max, W )
    SUBLIST ← FIND-SUBLIST( WORK, W, p )
    Q ← LEVEL-TO-QUEUES( INPUT, WORK, SUBLIST, D, Γ, γ, W, p, LEVEL )
    SIZES ← array[0 : p − 1]
    parallel for i ← 0 to p − 1 do
        Q-NEW ← queue
        for v in Q[i] do
            if OWNER[v] = i then
                Q-NEW.ENQUEUE( v )
        Q[i] ← Q-NEW
        SIZES[i] ← |Q[i]|
    PARALLEL-PREFIX-SUM( SIZES, 1 )
    INPUT ← array[0 : SIZES[p − 1]]
    parallel for i ← 0 to p − 1 do
        if i = 0 then
            OFFSET ← 0
        else
            OFFSET ← SIZES[i − 1]
        for j ← OFFSET to OFFSET + |Q[i]| do
            INPUT[OFFSET] ← Q[i].DEQUEUE( )
```

Fig. 3. Overwrite squirrels with its prefix sum.