PARALLEL AND DISTRIBUTED ALGORITHMS
BY
DEBDEEP MUKHOPADHYAY
AND
ABHISHEK SOMANI

http://cse.iitkgp.ac.in/~debdeep/courses_iitkgp/PAlgo/index.htm

PRAM ALGORITHMS: LIST RANKING AND COLORING

2

THE LIST RANKING PROBLEM

Given a linked list L of n nodes whose order is specified by an array S or Succ) such that S(i) contains a pointer to the node following i on L, for $1 \leq i \leq n$

We assume S(i)=0 when i is the end of the list.

The List Ranking problem is to determine the distance of each node i from the end of the list.

The List ranking problem is one of the most elementary problems in list processing whose sequential complexity is trivially linear.

The pointer jumping (PJ) technique can be used to derive a parallel algorithm for the list ranking problem.

The corresponding running time is O(log n), and the corresponding total number of operations is O(n log n) => Non-optimal solution.

3

OPTIMAL LIST RANKING

PJ can be made optimal if we can somehow reduce the size of the list to O(n/log n) nodes using a linear number of operations.

The standard approach to achieve optimality would be:

1. Partition the input list into approximately n/log n blocks, each containing O(log n) nodes.

2. Rank each node within each block by using an optimal sequential algorithm, called the preliminary rank.

3. Combine the preliminary ranks using an O(log n) time parallel algorithm.

Unfortunately each block can have O(log n) sublists due to PJ, in which case the size of the input list to the O(log n) time parallel algorithm would not have been reduced to O(n/log n) nodes.

4

ALTERNATIVE STRATEGY: SYMMETRY BREAKING AND DETERMINISTIC COIN TOSSING (COLE, VISHKIN'86)

Step 1: Shrink the linked list L to L' until only O(n/log n) nodes remain.

Step 2: Apply the pointer jumping technique on the short list L'.

- Requires O(lg n) time, with cost O(n)

Step 3: Restore the original list and rank all the nodes removed in step 1.

Step 1 is the main difficult step, which needs to be performed in O(log n) time with a cost of O(n)

5

INDEPENDENT SETS

The method for shrinking L consists of removing a selected set of nodes from L and updating the intermediate R values of the remaining nodes. (R(i) is the rank or distance of node i from the end of the list).

The key to a fast parallel algorithm lies in using an Independent Set of nodes which can be deleted in parallel.

A set I of nodes is independent if, whenever i∈I, if $\mathrm{s(i)} \neq i, s(i) \notin I$.

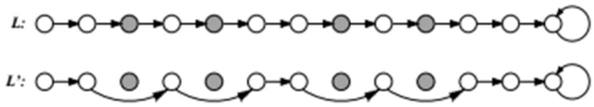
We can remove each node i∈I, by adjusting the successor pointer of the predecessor of i. Since, I independent this process can be applied concurrently to all the nodes in I.

6

DELETION OF THE INDEPENDENT NODES

$I \subset L\ is\ an\ independent\ set(IS) \Rightarrow \forall i \in I\ can\ be\ deleted$
$from\ L\ in\ parallel.$

Proof: If $I \subset L$ is an IS, then $\forall i \in I, P[i] \in I$.



7

IDENTIFYING THE INDEPENDENT SET

We can handle the problem of finding an independent set by coloring the nodes of the list L.

Recall that a k-coloring of L is a mapping from the set of nodes in L into {0,1,…,k-1} such that no adjacent vertices are assigned the same color.

A node u is a **local minimum (or maximum)** wrt. this coloring if the color of u is smaller (larger) than the colors of its predecessor and successor.

8

A RESULT

Let k≥2 be a constant and consider any k-coloring c of elements x of L, ie. $\forall x \in L, 1 \leq c(x) \leq k$ and $c(Pred[x] \neq c(x) \neq c(Succ[x])$. Then the set of local minima of coloring c is an IS of size Ω(n/k) and there is a work-optimal parallel algorithm to determine the local minima.

Proof: Let u and v be 2 local minima of c such that no other local minima exists between them.

Then u and v cannot be adjacent.

Colors of elements between u and v must form a bitonic sequence of at most (k-2)+1+(k-2)=2k-3 colors.

Thus, $|I| \geq \frac{n}{2k-2} = \Omega(\frac{n}{k})$.

Given a coloring, determining its local minima is trivial on EREW PRAM just by inspecting predecessor's color and successor's color for all elements in parallel.

9

REDUCING TO 3-COLORING

A large IS can be obtained by 3-coloring the list.

For a 3-coloring $|I| \geq \left\lfloor \frac{n}{4} \right\rfloor$.

In order to reduce n-element list L to L' with n/(log n) elements, we must remove ISs based on local minima of 3-coloring repeatedly.

10

BOUND ON NUMBER OF ITERATIONS

$O(\log logn)$ iterations consisting in removing ISs of local minima of 3-colorings are needed to reduce L to L' with $|L'| \leq n/\log n$.

Proof: Let m be the number of iterations required to reduce L to L'.

Let L_k be the length of L after k iterations and I_k be the IS of local minima of a 3-coloring of L_k.

Then $|I_k| \geq |L_k|/4$, and $|L_{k+1}| = |L_k| - |I_k| \leq (3/4)|L_k|$.

By recursive definition for $|L_k|$ and using $|L| = |L_0| = n$, we have $|L_k| \leq (3/4)^k n$.

Since, $|L_m| \leq n/\log n$, m must fulfil condition $(3/4)^m n \leq n/\log n$, which is equivalent to $m \geq \log_{4/3} logn = \frac{loglogn}{\log\left(\frac{4}{3}\right)} = 2.4loglogn.$

SOLVING THE 3-COLORING PROBLEM

The problem has reduced to the problem of 3-coloring of a linked list.

- Sequentially this is trivial. We just need to traverse the list, assigning alternate colors 0 and 1 (add a color 2 in case of a cycle)
- To do it in parallel we need to break the symmetry of the nodes assigned to every processor

Due to the fact that the indices are random (ie. Succ does not have any locality), nodes in a sublist of size log n assigned to each processor looks alike.

We need to partition them into classes such that all nodes can be assigned the same color in parallel.

We describe an elegant deterministic method called Deterministic Coin Tossing (DCT) to break the symmetry.

- Based on the idea that the only nonsymmetry among elements of the list is their unique identification numbering.
- The identifications are used as an initial n-coloring and it is then transformed into a 3-coloring.

A BASIC PARALLEL COLORING SCHEME USING DCT FOR A DIRECTED CYCLE

Assume the arcs of G are specified by an array S st:

- If (i,j)ϵE, we have s(i)=j, for 1≤i,j≤n
- We start with an initial coloring of c(i)=I for all i.
- The binary expansion of the color c is $c_{t-1}\ldots c_k\ldots c_1c_0$
- The kth LSB is c_k.

Parallel Reduction of the number of initial colors:

For 1≤i≤n, in parallel we:

1. Set k to the LSB in which c(i) and c(S(i)) differ.
2. Set $c'(i)=2k+c(i)_k$

Note if initial coloring is a t-bit value, max value of c'=2(t-1)+1=2t-1, which can be represented by a $\lceil \log(t) \rceil + 1$. Thus there is an exponential reduction in the number of colors!

Is the coloring correct?

13

CORRECTNESS

As the starting coloring is correct such a differing k must exist.

Suppose by contradiction the derived coloring is incorrect.

Thus for an edge (i,j)ϵE, c'(i)=c'(j).

- Thus, $2k+c(i)_k=2l+c(j)_l$.
- Note this can be only possible if k=l, but then $c(i)_k=c(j)_k$, which defies the definition of k.
- Hence, c'(i)≠c'(j), for any (i,j)ϵE.

Assuming that the LSB in which two binary numbers differ can be found on O(1) time, when the binary values are of size O(logn) bits, the algorithm is a constant time algorithm.

How do you convert this to a 3-coloring algorithm?

14

RECURSIVE APPLICATION OF THE ALGORITHM

The algorithm can be recursively applied reducing the number of colors till t>3.

- Note for t=3 bits, the max color value is 2.3-1=5, which also requires 3 bits.
- Thus the number of colors is $0 \leq c'i) \leq 5$.

Iterations of DCT can reduce the number of colors of a coloring only to 6.

We next estimate the number of iterations required to reach this stage.

- Let $\log^{(i)}(x)=\log(\log^{(i-1)}(x))$, $\log^{(1)}(x)=\log(x)$.
- Let $\log^*x=\min\{i \mid \log^{(i)}(x) \leq 1\}$
- The function $\log^*x$ is an extremely slowly growing function that is bounded by 5 for all x $\leq 2^{65536}$.

Starting with the initial coloring c(i)=i, for $1 \leq i \leq n$, then each iteration reduces the number of colors: after 1st iteration O(log n), after 2nd $O(\log^2(n))$.

Thus number of colors will be reduced to 6 after O(log*n) iterations.

15

THE FINAL CUT!

We apply a further recolor.

The additional recoloring procedure consists of 3 iterations, each of which handles vertices of a specific color.

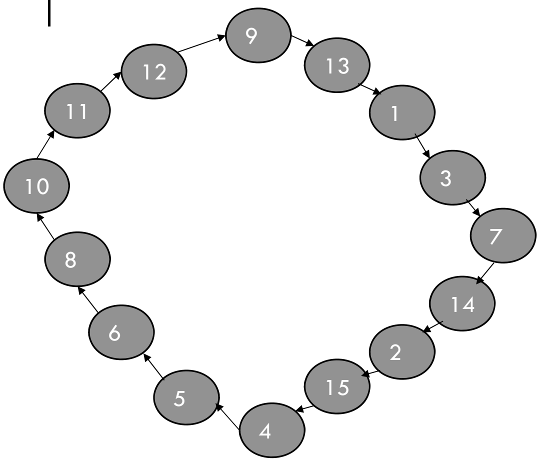
For each color which lies between 3 and 5, ie. $3 \leq l \leq 5$, we recolor all vertices i with color l with the smallest possible color from {0,1,2} (ie. Smallest color different from predecessor and successor).

Each iteration takes O(1) time with n processors.

- Note when two nodes with color 3 is handled, they are never adjacent.
- Thus the correctness is ensured.

16

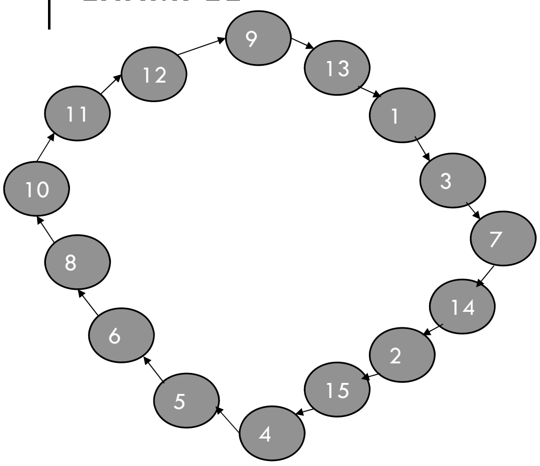
EXAMPLE



Note now there are 6 colors: 0-5

v	c	k	c'
1	0001	1	2
3	0011	2	4
7	0111	0	1
14	1110	2	5
2	0010	0	0
15	1111	0	1
4	0100	0	0
5	0101	0	1
6	0110	1	3
8	1000	1	2
10	1010	0	0
11	1011	0	1
12	1100	0	0
9	1001	2	4
13	1101	2	5

EXAMPLE



Note now there are 3 colors: 0-2

v	c	k	c'
1	0001	1	2
3	0011	2	0
7	0111	0	1
14	1110	2	2
2	0010	0	0
15	1111	0	1
4	0100	0	0
5	0101	0	1
6	0110	1	0
8	1000	1	2
10	1010	0	0
11	1011	0	1
12	1100	0	0
9	1001	2	1
13	1101	2	0

COMPLEXITY

Using DCT, we can construct a 3-coloring on p processors in time T(n,p)=O(nlog*n/p) with C(n,p)=O(nlog*n).

When p=n, T=O(log*n), with C=O(nlog*n).

Optimal Algorithm for 3-coloring:

Apply the 3-coloring once.

For the O(log n) remaining colors we apply the re-coloring scheme.

We can 3-color in time O(log n) time, with a cost of O(n).

19

LIST RANKING USING COLORING

1. Set n0=n, k=0
2. While nk>n/log n do

   2.1 Set k=k+1

   2.2 Color the list with 3 colors, and identify the set I of local minima

   2.3 Remove the nodes in I, and store the appropriate information regarding the removed nodes (discuss later)

   2.4 Let nk be the size of the remaining list. Compact list into consecutive memory locations.
3. Apply PJ to the resulting list.
4. Restore the original list and rank all the removed nodes by reversing the process in Step 2

Note step 2 needs to be repeated O(loglogn) times. 2.2 takes O(log n) time using O(n) operations.

We need to discuss Steps 2.3.

20