Tree Traversal using OpenMP Tasks

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University
We would like to traverse it as quickly as possible. We are assuming that we do not need to traverse it in order. We just need to visit all nodes.
Tree Traversal Algorithms

- Common in graph algorithms, such as searching.

- If the tree is binary and is balanced, then the maximum depth of the tree is $\log_2(\# \text{ of Nodes})$

- Strategy at a node:
  1. follow one descendent node
  2. follow the other descendent node
  3. process the node you’re at

This order could be re-arranged, depending on what you are trying to do.
#pragma omp parallel

#pragma omp single

Traverse(root);

#pragma omp taskwait
void Traverse(Node *n) {
    if (n->left != NULL) {
        #pragma omp task private(n) untied
        Traverse(n->left);
    }

    if (n->right != NULL) {
        #pragma omp task private(n) untied
        Traverse(n->right);
    }
}

#pragma omp taskwait
Process(n);

Put this here if you want to wait for both branches to be taken before processing the parent.
Parallelizing a Binary Tree Traversal with Tasks

Traverse( A );
Parallelizing a Binary Tree Traversal with Tasks: *Tied*

Threads:

0 1 2 3

Traverse( A );

```
A
  B
    D
      H
        1
          3
  E
    J
      4
        2
    K
      5
        2
  C
    F
      10
        0
    G
      13
        1
```
Parallelizing a Binary Tree Traversal with Tasks: Untied

Traverse(A);

Threads:

0 1 2 3

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

15 1

14 2

13 2

12 2

11 2

9 2

8 3

5 0

4 3

2 3

1 3

0 2

3 3

6 0

10 2

3 0

2 1

0 0

2 3

15 1

14 2

13 2

12 2

11 2

9 2

8 3

5 0

4 3

2 3

1 3

0 2

3 3

6 0

10 2

3 0

2 1

0 0
### How Evenly Tasks Get Assigned to Threads

#### 6 Levels – g++ 4.9:

<table>
<thead>
<tr>
<th>Thread #</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
</tr>
</tbody>
</table>

#### 6 Levels – icpc 15.0.0:

<table>
<thead>
<tr>
<th>Thread #</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
</tr>
</tbody>
</table>

#### 12 Levels – g++ 4.9:

<table>
<thead>
<tr>
<th>Thread #</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2561</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2813</td>
</tr>
<tr>
<td>3</td>
<td>2815</td>
</tr>
</tbody>
</table>

#### 12 Levels – icpc 15.0.0:

<table>
<thead>
<tr>
<th>Thread #</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1999</td>
</tr>
<tr>
<td>1</td>
<td>2068</td>
</tr>
<tr>
<td>2</td>
<td>2035</td>
</tr>
<tr>
<td>3</td>
<td>2089</td>
</tr>
</tbody>
</table>
void Process( Node *n )
{
    for( int i = 0; i < 1024; i++ )
    {
        n->value = pow( n->value, 1.1 );
    }
}
Performance vs. Number of Threads

![Graph showing performance vs. number of threads.](image)

- X-axis: # Threads
- Y-axis: Nodes Processed per Second
- Legend:
  - 16 threads
  - 12 threads
  - 8 threads
  - 4 threads

The graph illustrates the increase in nodes processed per second as the number of threads increases. The performance is highest with 16 threads, followed by 12, 8, and 4 threads, respectively.
Performance vs. Number of Levels

- **Number of Levels** vs. **Nodes Processed per Second**
- Lines represent different numbers of threads: 8, 6, 4, 2, 1
- The graph shows how the number of nodes processed per second increases with the number of levels and the number of threads.
Parallelizing a Tree Traversal with Tasks

- Tasks get spread among the current “thread team”

- Tasks can execute immediately or can be deferred. They are executed at “some time”.

- Tasks can be moved between threads, that is, if one thread has a backlog of tasks to do, an idle thread can come steal some workload.

- Tasks are more dynamic than sections. The task paradigm would still work if there was a variable number of children at each node.
void Traverse( Node *n )
{
    for( int i = 0; i < n->numChildren; i++ )
    {
        if( n->child[i] != NULL )
        {
            #pragma omp task
            Traverse( n->child[i] );
        }
    }

    #pragma omp taskwait
    Process( n );
}
Performance vs. Number of Levels

Number of Levels

Nodes Processed per Second

8-thread Speed-up ≈ 6.7

$F_p \approx ??\%$

Max Speed-up ≈ ??
Performance vs. Number of Threads

8-thread Speed-up ≈ 6.7

\[ F_p \approx ??\% \]

Max Speed-up ≈ ??

\[ F_p = \frac{n}{(n - 1)} \left(1 - \frac{1}{\text{Speedup}}\right) = 97\% \]

\[ \text{max Speedup} = \frac{1}{1 - F_p} = 33x \]