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.

- 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 descendant node
2. follow the other descendant node
3. process the node you’re at

---

```c
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 );
}
```

---

```c
#pragma omp parallel
#pragma omp single
Traverse( root );
#pragma omp taskwait
```

Without this, each thread does a full traversal

Without this, thread #0 has to do everything

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
Parallelizing a Binary Tree Traversal with Tasks: Tied

Tied

Parallelizing a Binary Tree Traversal with Tasks: Untied

Untied

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>23</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>

Benchmarking a Binary Task-driven Tree Traversal

```c
void Process( Node *n )
{
    for( int i = 0; i < 1024; i++ )
    {
        n->value = pow( n->value, 1.1 );
    }
}
```

Performance vs. Number of Threads

![Performance vs. Number of Threads Graph]

Number of Nodes Processed per Second vs. Number of Threads

Performance vs. Number of Levels

![Performance vs. Number of Levels Graph]

Number of Nodes Processed per Thread vs. Number of Levels and Threads
• 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.

```c
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](chart1)

8-thread Speed-up = 6.7
\[ F_p \approx 77\% \]
Max Speed-up = ??

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

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