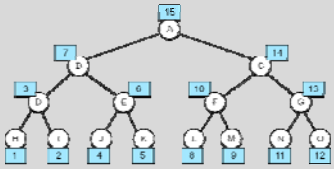
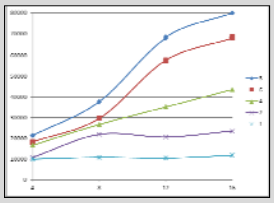
Tree Traversal using OpenMP Tasks

Mike Bailey

mjb@cs.oregonstate.edu

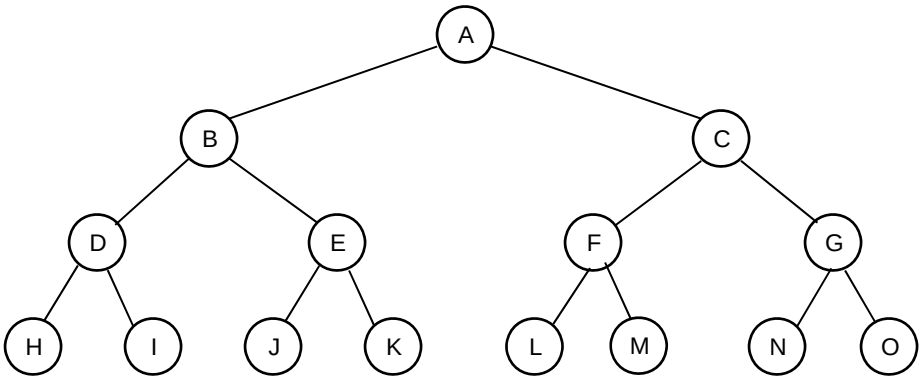
Oregon State University





Tree Traversal Algorithms

Given a tree . . .



. . . we would like to traverse it as quickly as possible!

Tree Traversal Algorithms

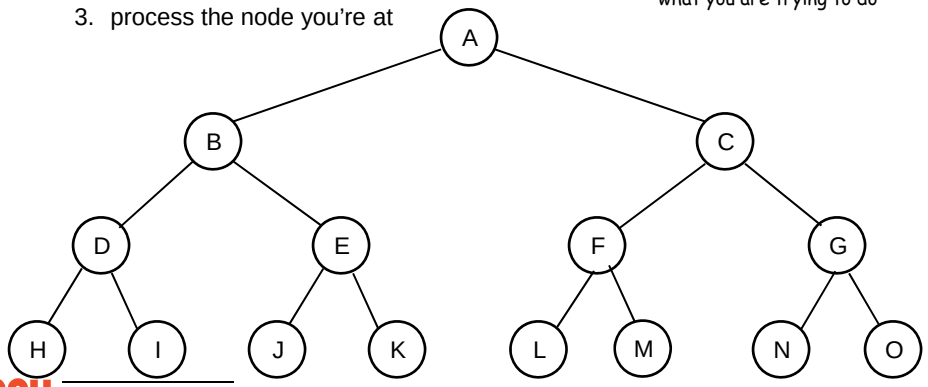
• Common in graph algorithms, such as searching.

• If the tree is binary and is balanced, then the maximum depth of the tree is $\log_2$(# 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



Tree Traversal Algorithms

Without this, thread #0 has to do everything

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

Without this, each thread does a full traversal

Put this here if you want to wait for all nodes to be traversed before proceeding

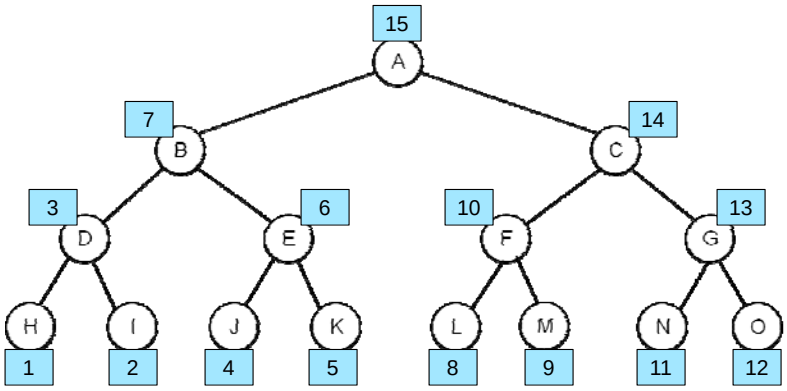
Parallelizing a Binary Tree Traversal with Tasks

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

        if( n->right  !=  NULL )
        {
                #pragma omp task 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

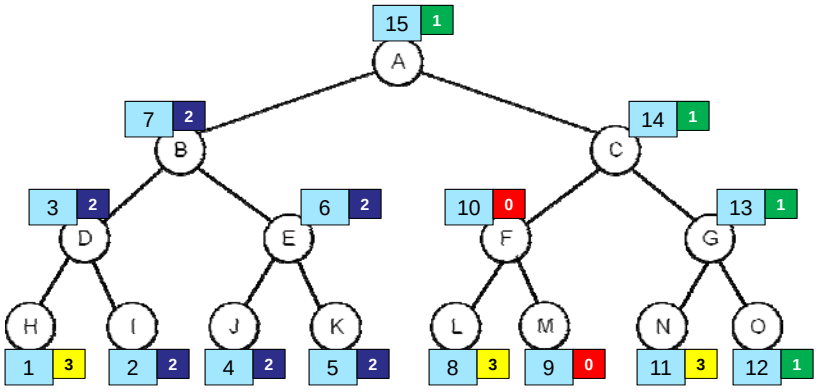
Traverse(A);



Parallelizing a Binary Tree Traversal with Tasks: *Tied*

Threads: 0 1 2 3

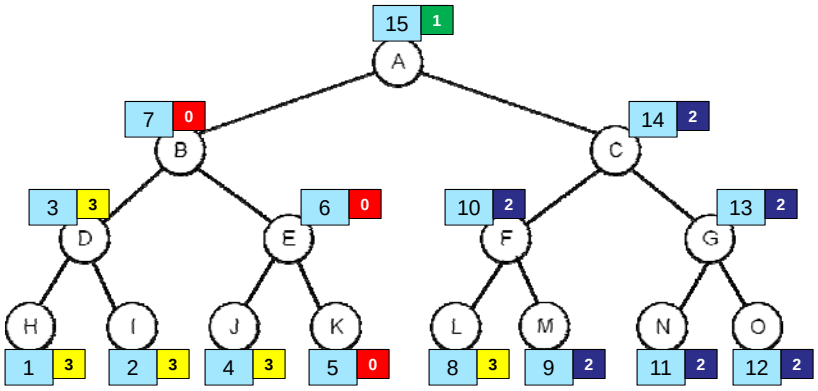
Traverse(A);



Parallelizing a Binary Tree Traversal with Tasks: *Untied*

Threads: 0 1 2 3

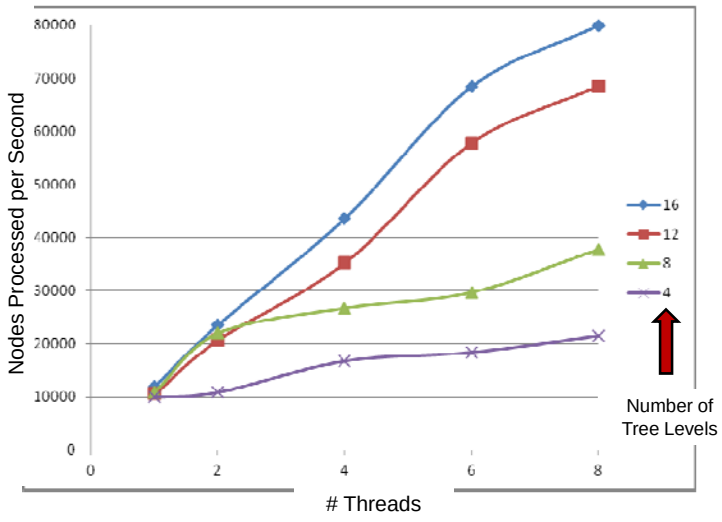
Traverse(A);



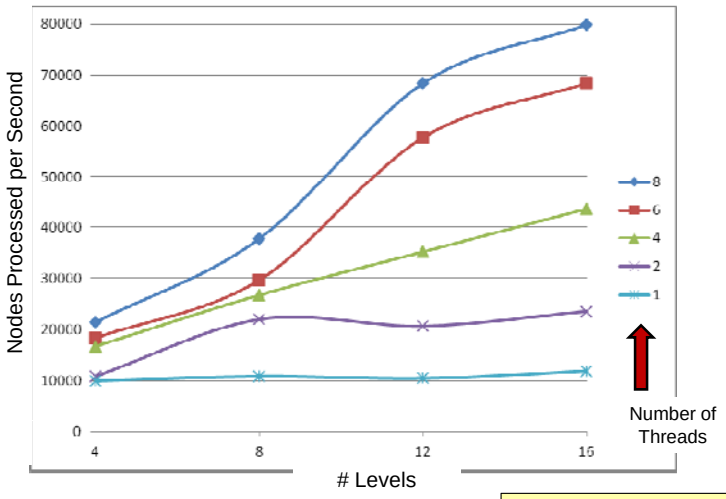
Benchmarking a Binary Task-driven Tree Traversal

```
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 Levels

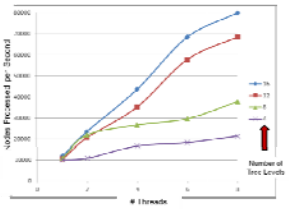


16-thread Speed-up ≈ 8

F_p ≈ ??%

Max Speed-up ≈ ??

Performance vs. Number of Threads



16-thread Speed-up ≈ 8

F_p ≈ ??%

Max Speed-up ≈ ??

$$F_p = \frac{n}{(n-1)}\left(1 - \frac{1}{Speedup}\right) = 93\%$$

$$\max Speedup = \frac{1}{1 - F_p} = 15x$$

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.

Parallelizing an N-Tree Traversal with Tasks

```
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 );
}
```