CME 213, ME 339—Spring 2021

Eric Darve, ICME, Stanford

"A great lathe operator commands several times the wage of an average lathe operator, but a great writer of software code is worth 10,000 times the price of an average software writer." (Bill Gates)

OpenMP

Central for multicore scientific computing

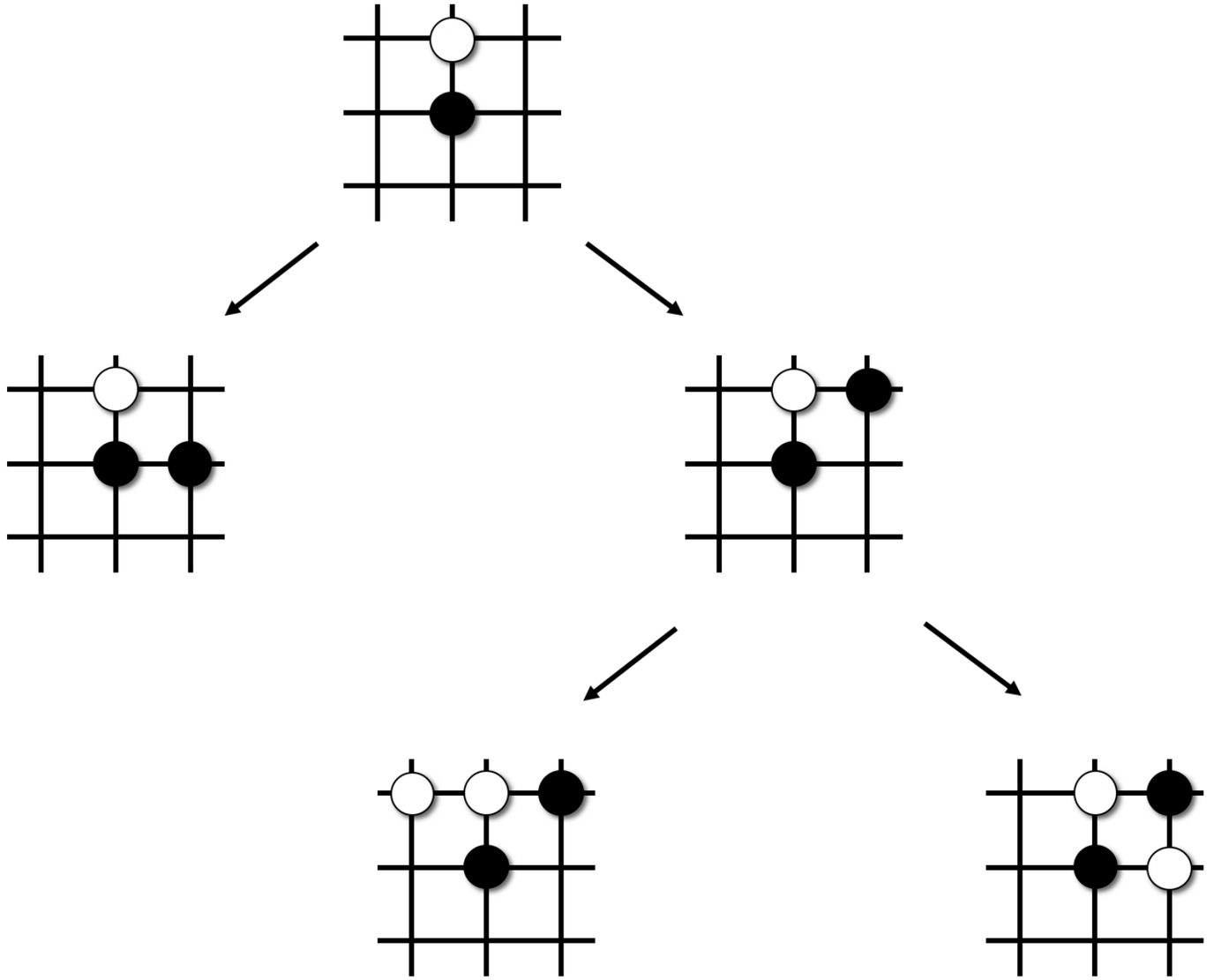
#pragma omp parallel for

Next topic

#pragma omp task

Many situations require a more flexible way of expressing parallelism

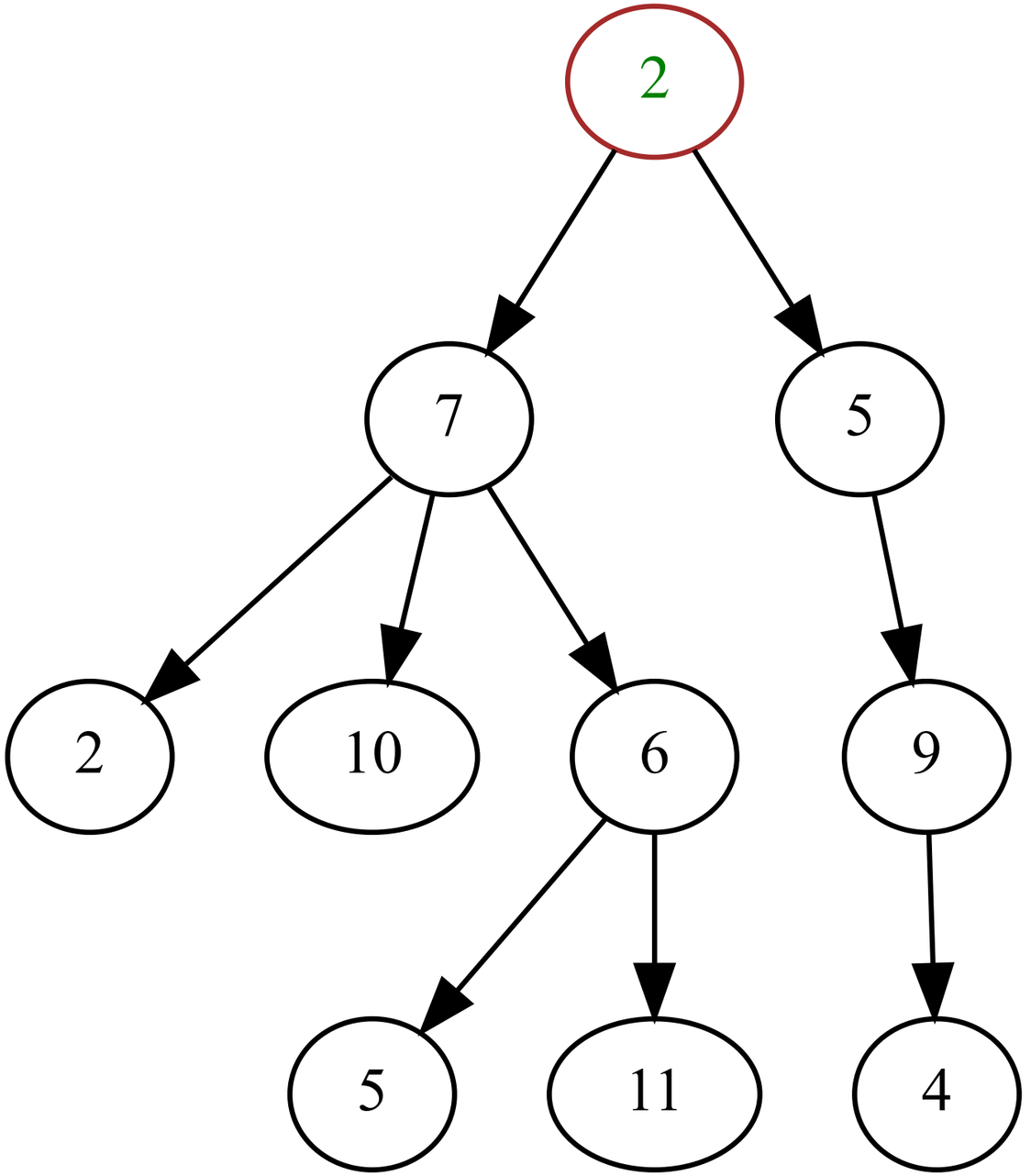
Example: tree traversal



Tree traversal

Go through each node and execute some operation

Tree is not full, e.g., number of child nodes varies



tree.cpp

```
void Traverse(struct Node *curr_node)
{
   // Pre-order = visit then call Traverse()
   Visit(curr_node);

   if (curr_node->left)
#pragma omp task
      Traverse(curr_node->left);

   if (curr_node->right)
#pragma omp task
      Traverse(curr_node->right);
}
```

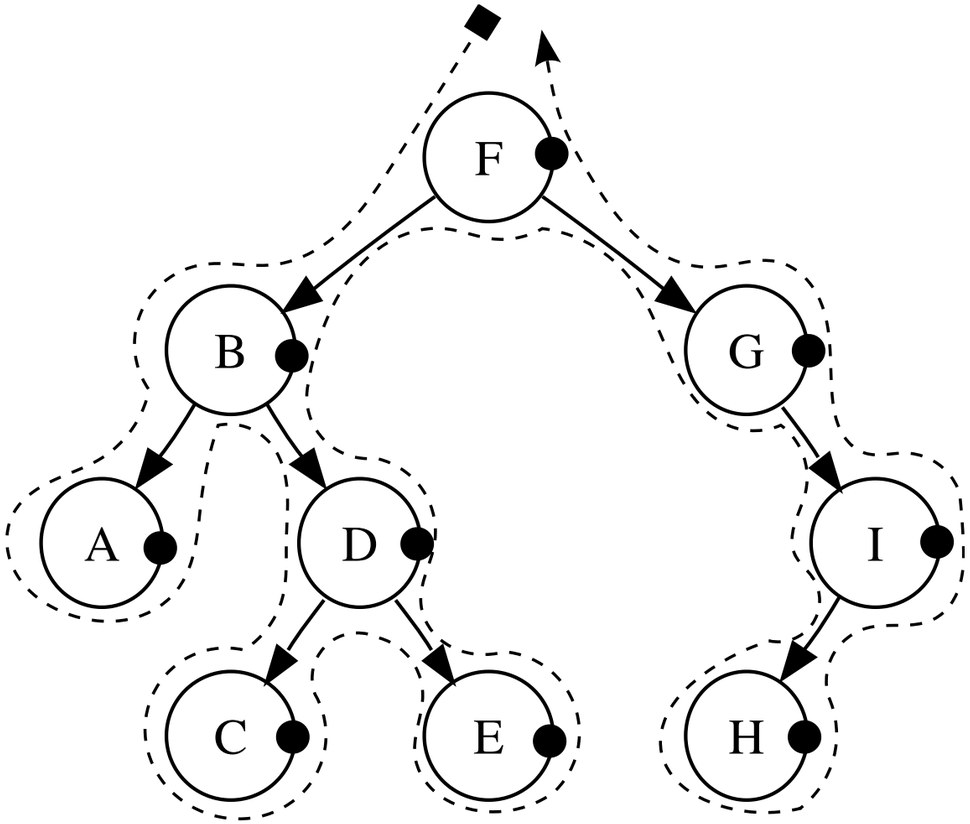
In main()

```
#pragma omp parallel
#pragma omp single
{
   // Only a single thread should execute this
   Traverse(root);
}
```

The encountering thread may immediately execute the task, or defer its execution.

Any thread in the team may be assigned the task.

Post-order traversal



This algorithm requires waiting for traversal of children to be complete.

tree_postorder.cpp

```
int PostOrderTraverse(struct Node* curr_node) {
    int left = 0, right = 0;

    if(curr_node->left)
        #pragma omp task shared(left)
        left = PostOrderTraverse(curr_node->left);
    // Default attribute for task constructs is firstprivate

    if(curr_node->right)
        #pragma omp task shared(right)
        right = PostOrderTraverse(curr_node->right);

    #pragma omp taskwait
    curr_node->data = left + right; // Number of children nodes
    return 1 + left + right;
}
```

firstprivate

Private but value is initialized with the original value when the construct is encountered

taskwait

Wait on the completion of the child tasks of the current task

Next example

Processing entries in a list

list.cpp

```
#pragma omp parallel
#pragma omp single
{
   Node* curr_node = head;
   while(curr_node) {
      #pragma omp task
      {
         // curr_node is firstprivate by default
         Visit(curr_node);
      }
      curr_node = curr_node->next;
   }
}
```

More recent features of task

Priority

```
for (i=0;i<N; i++) {
   #pragma omp task priority(i)
   compute_array(&array[i*M], M);
}
```

Higher priority = task is a candidate to run sooner

Dependence

taskwait

Can we specify dependencies between tasks in a more fine-grained fashion?

depend(dep-type: x)

dep-type is one of

in, out, inout, mutexinoutset

```
#pragma omp parallel
#pragma omp single
{
   #pragma omp task shared(x) depend(out: x)
   x = 2;
   #pragma omp task shared(x) depend(in: x)
   printf("x = %d\n", x);
}
```

Always prints x = 2

dep-type	waits on	waits on	waits on
in		out/inout	mutexinoutset
out/inout	in	out/inout	mutexinoutset
mutexinoutset	in	out/inout	

```
int x = 1;
#pragma omp parallel
#pragma omp single
{
   #pragma omp task shared(x) depend(in: x)
   printf("x = %d\n", x);
   #pragma omp task shared(x) depend(out: x)
   x = 2;
}
```

Always prints x = 1

mutexinoutset

Defines mutually exclusive tasks

See OpenMP examples for more information.

```
#pragma omp parallel
#pragma omp single
{
   #pragma omp task depend(out: c)
   c = 1; /* Task T1 */
   #pragma omp task depend(out: a)
   a = 2; /* Task T2 */
   #pragma omp task depend(out: b)
   b = 3; /* Task T3 */
   #pragma omp task depend(in: a) depend(mutexinoutset: c)
   c += a; /* Task T4 */
   #pragma omp task depend(in: b) depend(mutexinoutset: c)
   c += b; /* Task T5 */
   #pragma omp task depend(in: c)
   d = c; /* Task T6 */
}
printf("d = %1d\n", d);
```

Matrix-matrix product with tasks

```
#pragma omp parallel
#pragma omp single
for (int i = 0; i < N; i += BS)
{
   // Note 1: i, A, B, C are firstprivate by default
   // Note 2: A, B and C are pointers
   #pragma omp task depend(in: A[i*N:BS*N], B) depend(inout: C[i*N:BS*N])
   for (int ii = i; ii < i + BS; ii++)
      for (int j = 0; j < N; j++)
         for (int k = 0; k < N; k++)
            C[ii * N + j] += A[ii * N + k] * B[k * N + j];
}
```

depend(in: A[i * N:BS * N])

Specifies the entries in A for which there is an in dependency.

Syntax: A[lower-bound : length : stride]

Array sections
depend clause
Blocked matrix multiplication example

Blocked Cholesky algorithm

```
for (int k = 0; k < NB; k++)
{
   #pragma omp task depend(inout: A[k][k])
   spotrf(A[k][k]);
   for (int i = k + 1; i < NB; i++)
      #pragma omp task depend(in: A[k][k]) depend(inout: A[k][i])
      strsm(A[k][k], A[k][i]);
   // update trailing submatrix
   for (int i = k + 1; i < NB; i++)
   {
      for (int j = k + 1; j < i; j++)
         #pragma omp task depend(in: A[k][i], A[k][j]) depend(inout: A[j][i])
         sgemm(A[k][i], A[k][j], A[j][i]);
      #pragma omp task depend(in: A[k][i]) depend(inout: A[i][i])
      ssyrk(A[k][i], A[i][i]);
   }
}
```

OpenMP synchronization constructs

Reduction

```
#pragma omp parallel for reduction (+:sum)
for(int i = 0; i < size; i++) {
    sum += a[i];
}
```

Prevent a race condition when updating `sum`

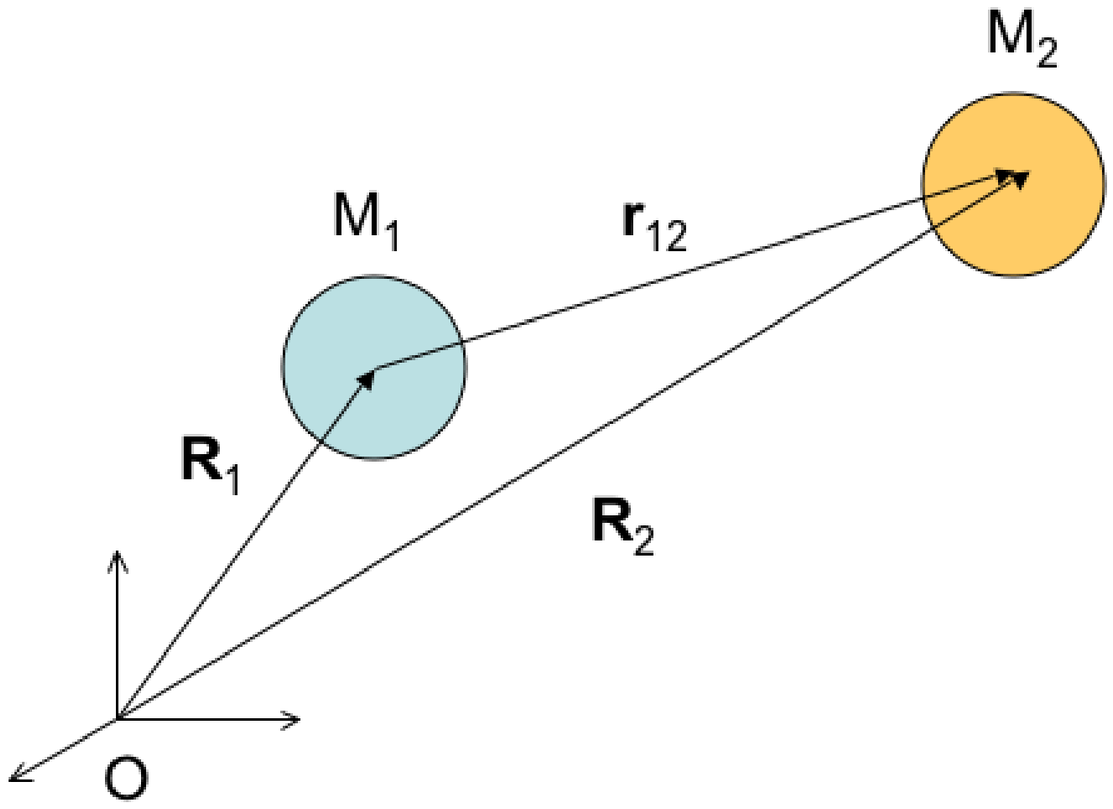
Improved efficiency

Exercise: entropy.cpp

Atomic

Allows: +=, *=, /=, ...

Not as efficient as reduction


M2
M1
r12
R1
R2
O

atomic.cpp

```
#pragma omp parallel for
for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
    {
        const float x_ = x[i] - x[j];
        const float f_ = force(x_);
        #pragma omp atomic
        f[i] += f_;
        #pragma omp atomic
        f[j] -= f_;
    }
```

critical

Restricts execution of the associated structured block to a single thread at a time

critical.cpp

```
set<int> m;
#pragma omp parallel for
for (int i = 2; i <= n; ++i)
{
    bool is_prime = is_prime_test(i);

    #pragma omp critical
    if (is_prime)
        m.insert(i); /* Save this prime */
}
```

Other topics (not covered)

Affinity, `target`, `simd`, locks

OpenMP examples, OpenMP specifications