Colorado State University

CS 475 GRAD 510
Fall 2019 Week 3+4
Scan Computations

Sanjay Rajopadhye
Colorado State University

Recap analysis of parallel programs (tasks or loops)

- Sequential execution time (aka work performed by the program/algorithm)
  - Depends on parameter *N* (input size)
- Parallel execution time
  - Depends on both N and *p* (number of threads/processors/cores)
- Best case parallel execution time (aka critical path or depth)
- General rule of thumb *N* grows much faster than *p*
- Link to Amdahl's Law

Colorado State University

Inherently Sequential Programs

```
// foo is some function with no (internal parallelism)
   Y = X[0];
   for (i=1; i<N; i++){
        Y = foo(Y, X[i]);
}
```

- Impossible to parallelize
  - Result of the previous iteration is needed in order to start the next one
- Under what conditions (properties of foo), is parallelization possible

Colorado State University

Another cousin (scan)

```
// foo is some function with no (internal parallelism)
   Y[0] = X[0];
   for (i=0; i<N; i++){
        Y[i] = foo(Y[i], X[i]);
}
```

- What if foo is associative?

- That's all that's needed

Colorado State University

Divide & Conquer: expose parallelism

- First a simple problem: reduction (only last answer needed)

```
Y = X[0];
for (i=0; i<N; i++)
    Y = foo(Y, X[i]);
```

```
Y = reduce(lo, hi)

reduce (lo, hi){
  if (lo=hi) return X[lo];
  else {
    mid = (lo+hi)/2;
    left = reduce (lo, mid);
    right = reduce (mid+1, hi);
    return foo(left, right);
  }
```

Colorado State University

Analysis of D&C reduction

- Work complexity: O(n)
- Span (critical path): lg n
- On a real machine (OpenMP): n/p
- Throttle parallelism with `if clause`
  - From the bottom, e.g., `if ( hi-lo < 2000)`
  - From the top e.g., `if ( "# siblings < p")`
    - Exercise to solve in PA2

Colorado State University

D&C scan: extra work to expose parallelism

```
Y[0]= X[0];
for (i=1; i<N; i++)
    Y[i] = foo(Y, X[i]);
```

```
Y = scan(lo, hi)

scan (lo, hi){
  if (lo=hi) {Y[lo] = X[lo]; return}
  else {
    mid = (lo + hi)/2;
    Y[lo : mid] = scan (lo, mid);
    Y[mid+1 : hi] = scan (mid+1, hi);
    Y[mid+1 : hi] = foo(Y[mid], Y[mid+1 : hi]);
    return Y;
  }
```

Colorado State University

Parallel D&C Scan

```
Y = reduce(lo, hi)

reduce (lo, hi){
  if (lo=hi) return X[lo];
  else {
    mid = (lo+hi)/2;
#pragma omp task
    Y[lo : mid] = scan (lo, mid);
    Y[mid+1 : hi] = scan (mid+1, hi);
#pragma omp taskwait
#pragma omp parallel for
    Y[mid+1 : hi] = foo(Y[mid], Y[mid+1 : hi]);
 }
```

Colorado State University

Analysis of D&C scan

Same recursion pattern, work is done "on the way up" after the children return

- Work complexity: $O(n \lg n)$
- Span (critical path): $\lg n$
  - if the calls to `foo` are in a `parallel for`
- In practice (OpenMP): $(n \lg n)/p$

must throttle (from the root) for efficiency

Colorado State University