Activity: Redesigning for parallelism

# Reduce

Consider the reduce function:

T reduce ( T\* array , size\_t n, T\* final\_result) {

T result = array [0];

for ( int i =1; i < n ; ++ i )

result = op ( result , array [ i]);

return op (final\_result, result);

}

T represents some type, such as int or float. op is some function.

If you define T as int and op as sum, this function computes the sum of the array. You could use op as

max and compute the maximum value of the array.

## int, sum

Consider first the int, sum case which computes the sum of an array of integers.

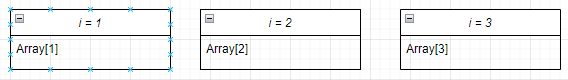
We have already shown that there is no parallelism in the parallel task graph extracted from the algorithm as it is written.

**Question:** Assuming you have *P* processors, rewrite the code to introduce one local variable per processor to store partial computation so as to achieve more parallelism. What is the width, critical path, and work? (hint: I am not asking you to make this code parallel. I am asking you to change the code structure to enable more parallelism in the task graph. View slides-redesigning.pdf scoping example.)

Width = n

critical path = 1

work = n



## Variants

(Note that by correct we mean that the parallel version would produce the exact same result as the sequential version in all cases.)

**Question:** Would that parallel version be correct for int, max? Why?

Yes, the final\_result will be compared against every result finding the largest number.

**Question:** Would that parallel version be correct for char\*, strcat? Why? No. strcat may add strings in different locations after every time the program is run.

**Question:** Would that parallel version be correct for float, sum? Why?

Yes. The final\_result will have every result added to it.

**Question:** Would that parallel version be correct for float, max? Why?

Yes, same as Q1

# Gantt Charts - Midterm Fall 2017

3



A

4

B

C

7

4 D

E

2

1

F

5 G

H

6

1

I

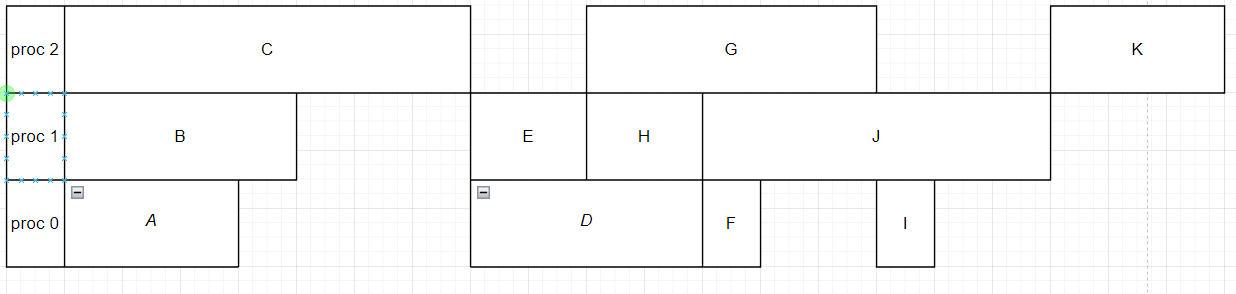
J

3 K

2

**Question:** Use List Scheduling (see class-scheduling.pdf) to build a schedule of this task graph on 2 pro- cessors.

**Question:** Use List Scheduling to build a schedule of this task graph on 3 processors.



# Mutexes

**Question:** Is the following code thread-safe? Explain.

# include < pthread .h> # include <stdio .h>

void func( int\* a, int\* b)

{

pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER ; for ( int i = 1; i < 10000 ; ++i)

{

pthread\_mutex\_lock (& mut);

\* a += b[i -1]; pthread\_mutex\_unlock (& mut);

}

}

Yes. the mutex lock prevents all other threads from adding to a\* so it is not possible for it to be overwritten inside this function.