Spring 2023 ITCS-3145 Homework: Extracting PTGs

Name: Jaxson Meisenhelter

Instructions: Do your own work. Follow the instructions for each question. **Submit via Canvas in a zip file**, containing your code for problem (5) and a pdf of your text solutions. If you choose to hand-write solutions, you may submit them as a *legible* scan or photo; otherwise, consider using tools like draw.io to illustrate them electronically.

# Transform

Consider the transform function:

void transform ( int\* a, int\* b, int n) { for ( int i=0; i<n; ++i)

b[ i] = f( a[ i]);

}

* 1. Illustrate the parallel task graph. Be sure to label the costs, dependencies, and the type of dependency. Assume the call to f cost O(1). Assume calls to f are always independent; aka for all intent and purpose, assume f(x) just returns x. (Note: Yes, this problem is simple.) (i=0, a[0] R, b[0] W, f(a[0]) R, cost 2) (i=1, a[1] R, b[1] W, f(a[1]) R, cost 2) … (i=n, none, a[n] R, b[n] W, f(a[n]) R, cost 2) (assuming the cost of everything except for f is a cost of 1 for a total cost of 2)
  2. What is the width? 1
  3. What is the work? n
  4. What is the critical path? What is its length?

(i = 0), length: 1

# Prefix sum

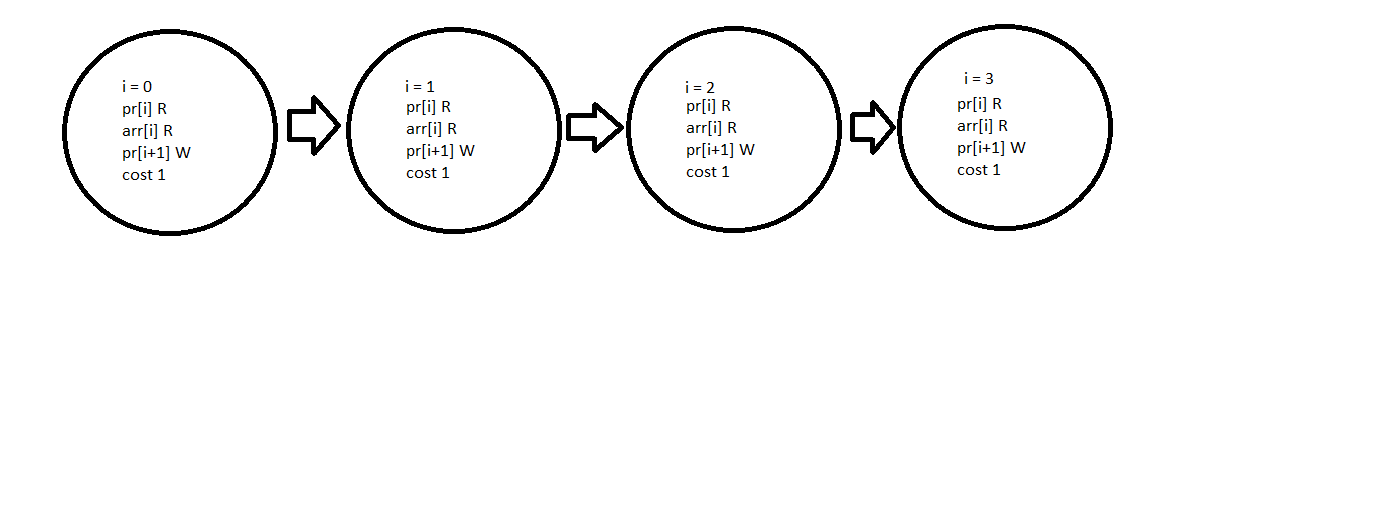
Prefix sum is an algorithm that has many uses in parallel computing. The algorithm computes *pr*[*i*] = z*j<i arr*[*j*]*, ∀*0 *≤ i ≤ n* and is often written sequentially:

void prefixsum ( int\* arr , int n, int\* pr) { pr[0] = 0;

for ( int i=0; i<n; ++i) pr[ i+1] = pr[ i] + arr[ i];

}

* 1. Illustrate the parallel task graph. Be sure to label the costs, dependencies, and the type of dependency.



* 1. What is the width? 1
  2. What is the work? n
  3. What is the critical path? What is its length?

0 -> 1 -> 2 -> … n, length: n

# Bubble Sort

The bubble sort algorithm can be written like this:

**void** bubbl esort ( **int** ∗ A, **int** n ) *{*

**for** ( **int** i =0; i *<*n ; ++i ) *{*

**for** ( **int** j =1; j *<*n ; ++j ) *{*

**i f** (A[ j ] *<* A[ j −1]) *{*

**int** temp = A[ j ] ;

A[ j ] = A[ j − 1 ] ;

A[ j −1] = temp ;

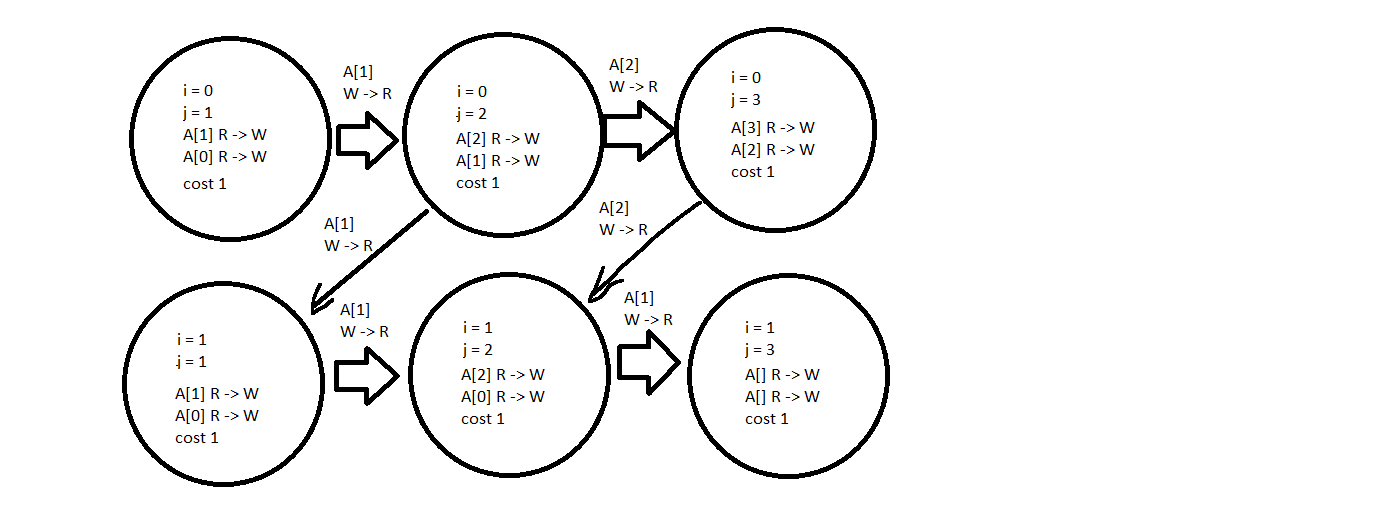
*}*

*}*

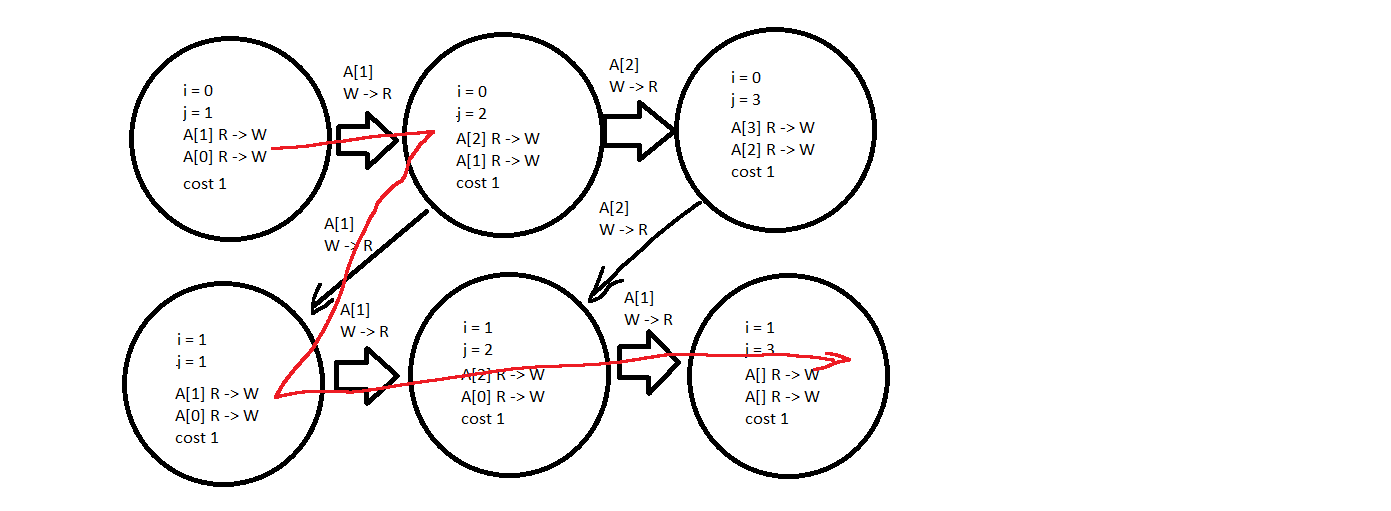
*}*

*}*

1. Illustrate the parallel task graph. Be sure to label the costs, dependencies, and the type of dependency.



1. What is the width? n
2. What is the work? n^2
3. What is the critical path? What is its length? Length n



# Knapsack

The Knapsack problem aims at finding the best set of objects to pack in a bag. Often the following dynamic programming algorithm is used to solve the problem.

**void** knapsack ( **int** n , **int** W, **int** value [ ] , **int** weight [ ] , **int** val [ ] [ ] ) *{*

**for** ( **int** a = 0 ; a*<*=W; ++a ) *{*

val [ 0 ] [ a ] = 0 ;

*}*

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

**for** ( **int** j =0; j*<*=W; ++j ) *{*

val [ i ] [ j ] = val [ i − 1 ] [ j ] ;

**i f** ( weight [ i −1] *<*= j ) *{*

val [ i ] [ j ] = max ( val [ i − 1 ] [ j ] , value [ i −1]+val [ i − 1 ] [ j−weight [ i − 1 ] ] ) ;

*}*

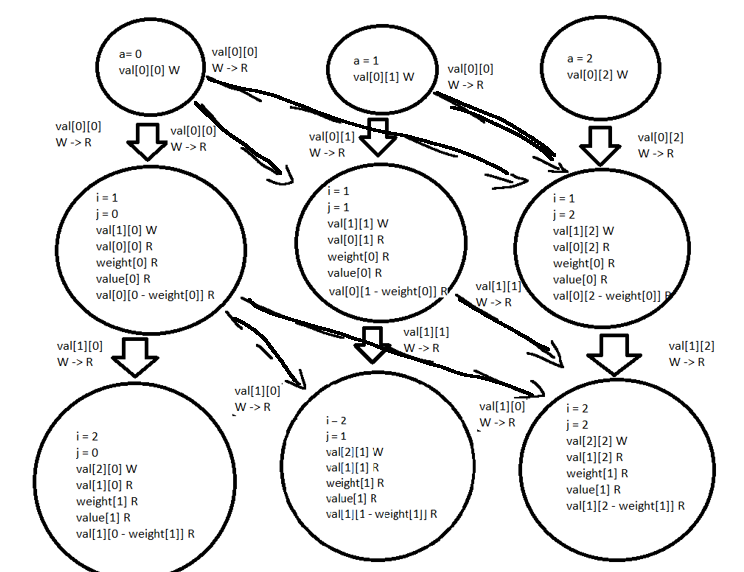
*}*

*}*

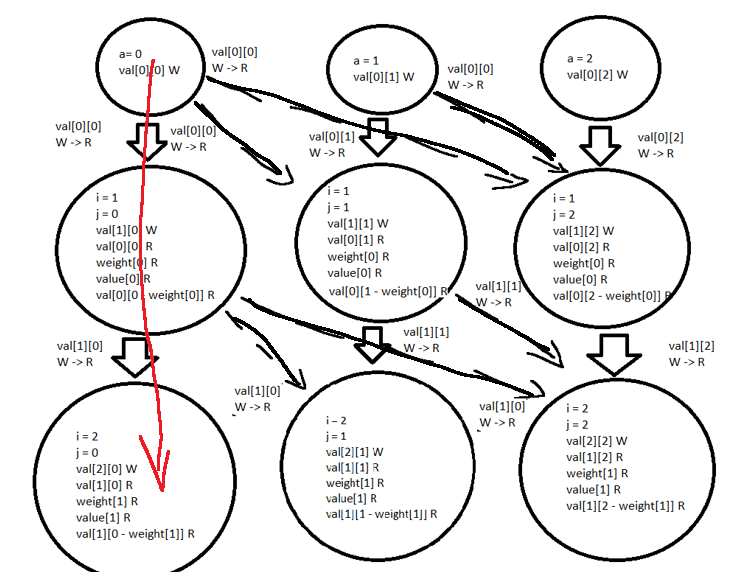
*}*

(You can assume weight is positive.)

* 1. Illustrate the parallel task graph. Be sure to label the costs, dependencies, and the type of dependency. Values chosen: n = 2, W = 2. Cost of each is unknown, assuming a cost of 1 for each.



* 1. What is the width? n + 1
  2. What is the work? n^2
  3. What is the critical path? What is its length? Length: n



# Vector Addition

See vecadd.zip on Canvas. This contains a **completed** *sequential* code for doing vector addition, svecadd.c. It also contains an incomplete pthread implementation, pvecadd.c.

* 1. Finish the pvecadd implementation. (hint: this should take no more than 5 lines of code).
  2. Starting at 100, execute both svecadd and pvecadd with 4 threads for powers of

10. Record the time it takes to execute both codes in a table. Do this until you achieve 4*x* speedup. Submit this table alongside your answer to earlier questions. **Make sure you do this performance experiment on Centaurus compute nodes or your timing will be invalid. (Tables are very buggy for some reason, they are forced to be unnecessarily long)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | 1,000 | 10,000 | 100,000 |  |  |  |
| svecadd | .000001s | .000006s | .000059s | .000634s |  |  |  |
| pvecadd | .000257s | .000388s | .000711s | .005483s |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 1,000,000 | 10,000,000 | 1E8 |
| svecadd |  | .016213s | .164500s | killed |
| pvecadd |  | .136862s | 1.659546s | killed |