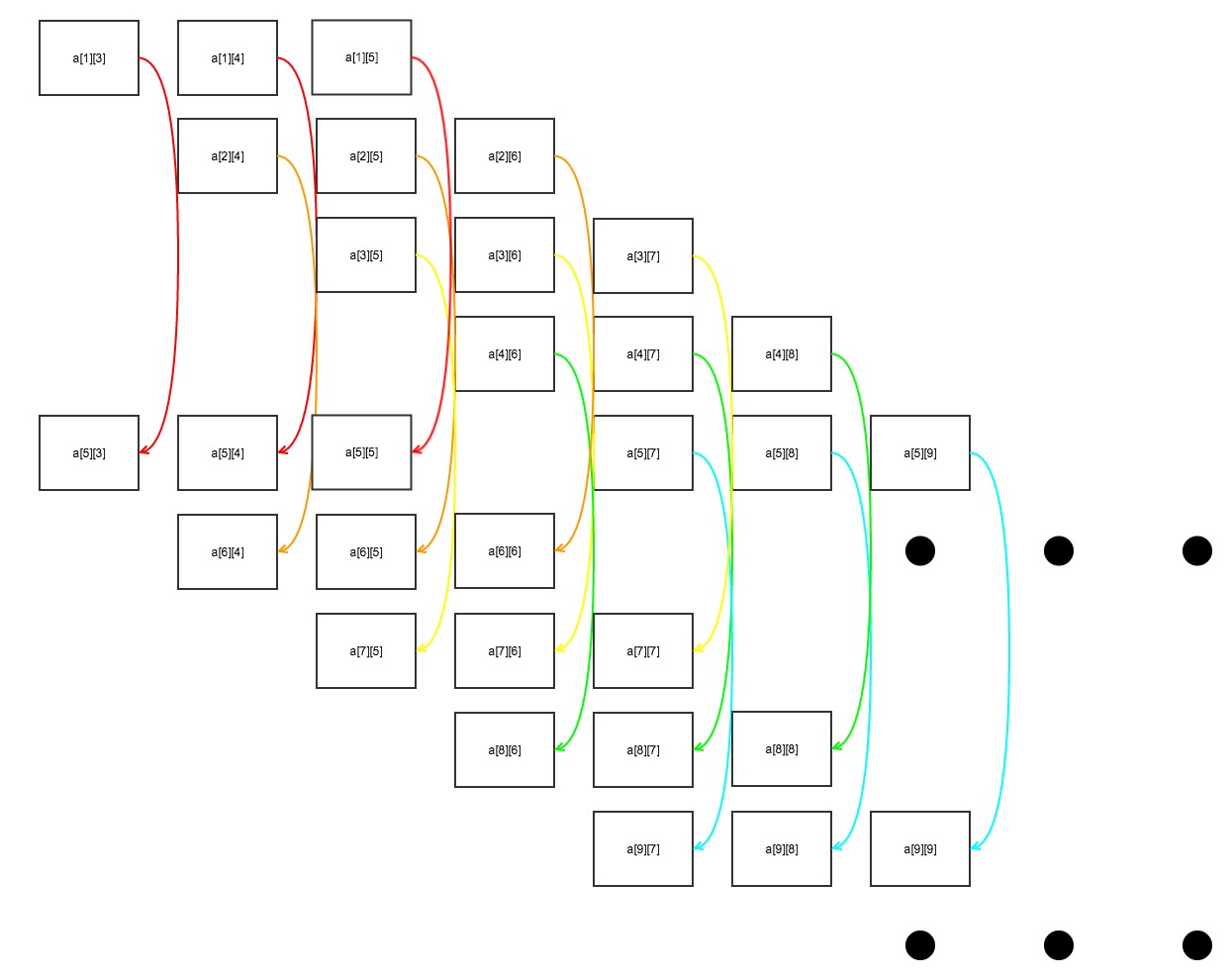
Dependencies:

1)



2)

The outer loop can be parallelized as a for-all loop without dependency.

The inner loop can be parallelized in groups of three indexes. For each outer loop iteration, a[i][j] is dependent on a[i-4][j]; a[i][j+1] is dependent on a[i-4][j+1]; a[i][j+2] is dependent on a[i-4][j+2].

So we can run the inner loop in parallel.

3)

We can parallel both of the outer loop and the inner loop since there is no dependence between the index a[i][j], a[i][j+1], and a[i][j+1]; nor between a[i] and a[i+1].

cilk\_for(int i = 5; i <= 100; i++){

cilk\_for(int j = i-2; j <= i; j++){

a[i][j] = a[i-4][j];

}

}

Patterns

1)

Stencil, reduce.

Stencil pattern is a map where each output depends on a “neighborhood” of inputs;

So we can use stencil pattern to gather the values of the current pixel’s neighbors with stencil.

Reduce pattern is used when we want to combines every element in a collection using an associative “combiner function”;

So we can use reduce pattern to get the average into the current pixel once we gather the values with stencil.

2)

Gather, scatter.

Gather pattern creates a collection of data by reading from another input data collection with a given collection of indices,

Scatter pattern takes a set of input data and a set of indices, and writes each element to the corresponding indices. And we can use collision to handle when we don't want some of the elements.

Map pattern

1)

Map performs a function over every element of a collection;

It replicates a serial iteration pattern where each iteration is independent of the others when the number of iterations is known in advance;

Which allows the elemental function to run in parallel (so that i iteration does not have to wait until i-1 iteration finishes before starting computation).

2)

// input: Am,n, bn, cn,1

// For each row

cilk\_for(int i = 0; i < m; i++){

// Get the sum for each index i, j

int sum = 0;

for (int = j = 0; j < n; j++){

sum += A[i][j] \* b[i];

}

// Stores the sum into the first index of each row in c

c[i][0] = sum;

}

3)

We might want to “fuse” the map and reduce patterns to increase arithmetic intensity in order to reduce memory/ cache usage (to improve data locality).

Collection

1)

**function** *mergesort*(m)

**var** list left, right, result

**if** length(m) ≤ 1

**return** m

**else**

**var** middle = length(m) / 2

left = **split** from 0 to middle - 1;

right =  **split** from middle to length(m)

left = **spawn** mergesort(left)

right = **spawn** mergesort(right)

**sync**

**if** last(left) ≤ first(right)

**append** right **to** left

**return** left

result = merge(left, right)

**return** result

**function** *merge*(left,right)

**var** list result

**while** length(left) > 0 and length(right) > 0

**if** first(left) ≤ first(right)

**append** first(left) **to** result

left = rest(left)

**else**

**append** first(right) **to** result

right = rest(right)

**if** length(left) > 0

**append** rest(left) **to** result

**if** length(right) > 0

**append** rest(right) **to** result

**return** result

2)

Tiling is used to break chunks of work up for workers to reduce serially;

Tiling can improves data locality, increase parallelism and reduce synchronization in parallel programs when performing a reduce or a scan.

No, it leads to less parallelization. Since tiling break the work into chunks (tiles) and operate on each tile separately and enable the parallelization; without the tiling, there is the dependency that requires synchronization and limit the parallelization.

3)

When adding and multiplying floating-point numbers, the floating-point numbers have only a certain amount of precision. After some numbers are rounded, the result of left associative computation might be different than the one with right associative computation and creates inconsistency.

And in parallel, the result might also be different than the one from serial computation because the work is computed in chunks and not in the order of the serial computation. Therefore, might also create inconsistency.