Loop parallelization

Loop-carried dependence

- The key concept for parallelization is the ***loop carried dependence***
  - A dependence that crosses loop iterations
- If there is a loop carried dependence, then that loop *cannot* be parallelized
  - Some iterations of the loop depend on other iterations of the same loop

Examples

```
for (i = 0; i < N; i++)
 a[2*i] = a[i];
```

Later iterations of i loop depend on earlier iterations

```
for (i = 0; i < N; i++)
 for (j = 0; j < N; j++)
  a[i+1][j] = a[i][j+2] + 1
```

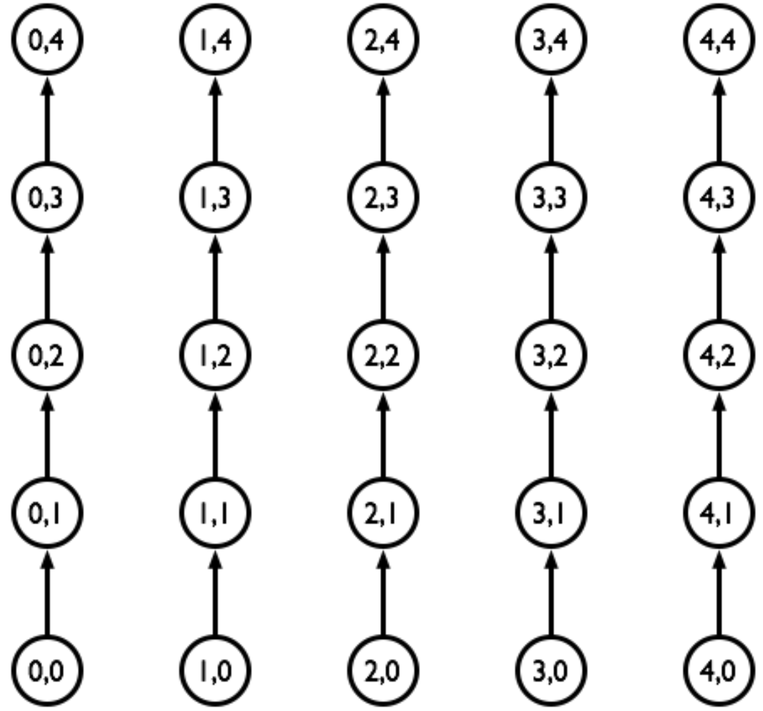
Later iterations of both i and j loops depend on earlier iterations

Some subtleties

- Dependences might only be carried over one loop!

```
for (i = 0; i < N; i++)
 for (j = 0; j < N; j++)
  a[i][j+1] = a[i][j] + 1
```

- Can parallelize i loop, but not j loop

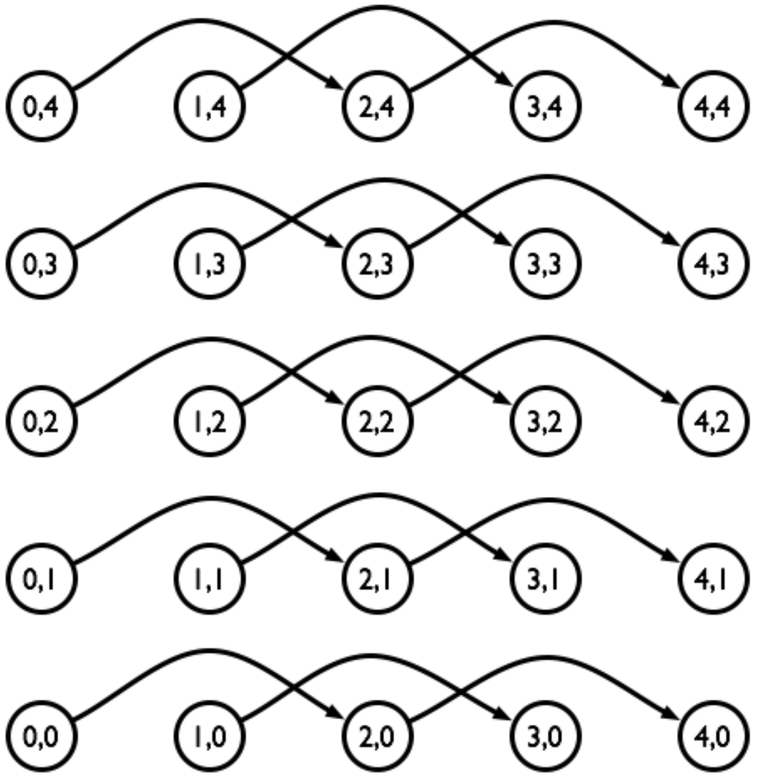


Some subtleties

- Dependences might only be carried over one loop!

```
for (i = 0; i < N; i++)
 for (j = 0; j < N; j++)
  a[i+1][j] = a[i-1][j] + 1
```

- Can parallelize j loop, but not i loop



Direction vectors

- So how do direction vectors help?
  - If there is a non-zero entry for a loop dimension, that means that there is a loop carried dependence over that dimension
  - If an entry is zero, then that loop can be parallelized!
- May be able to parallelize inner loop even if entry is not zero, but you have to carefully structure parallel execution