Other Loop Optimizations

Loop interchange

- Interchange doubly-nested loop to
  - Improve locality
  - Improve parallelism
    - Move parallel loop to outer loop (coarse grained parallelism)

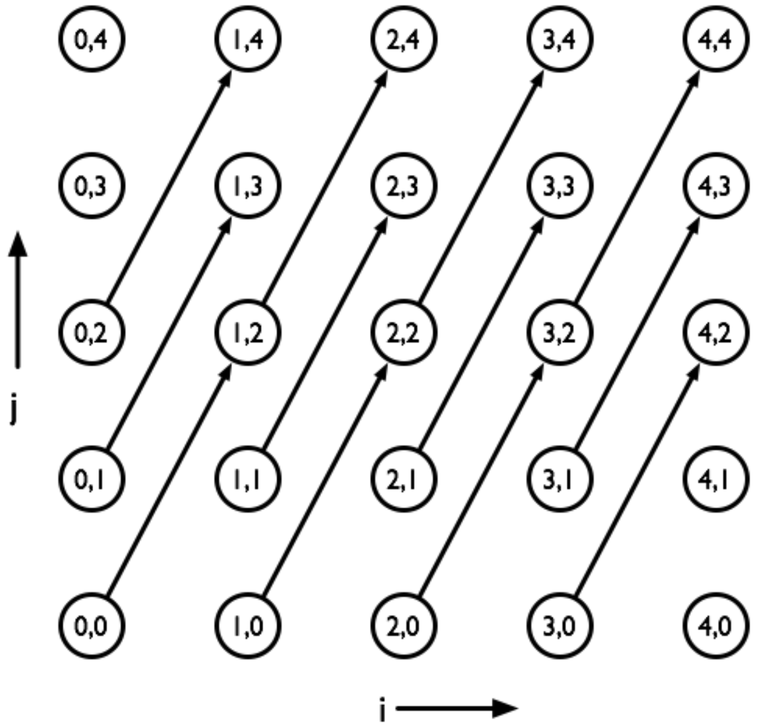
Loop interchange legality

- We noted that loop interchange is not always legal, because it reorders a computation
- Can we use dependences to determine legality?

Loop interchange dependences

- Consider interchanging the following loop, with the dependence graph to the right:

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



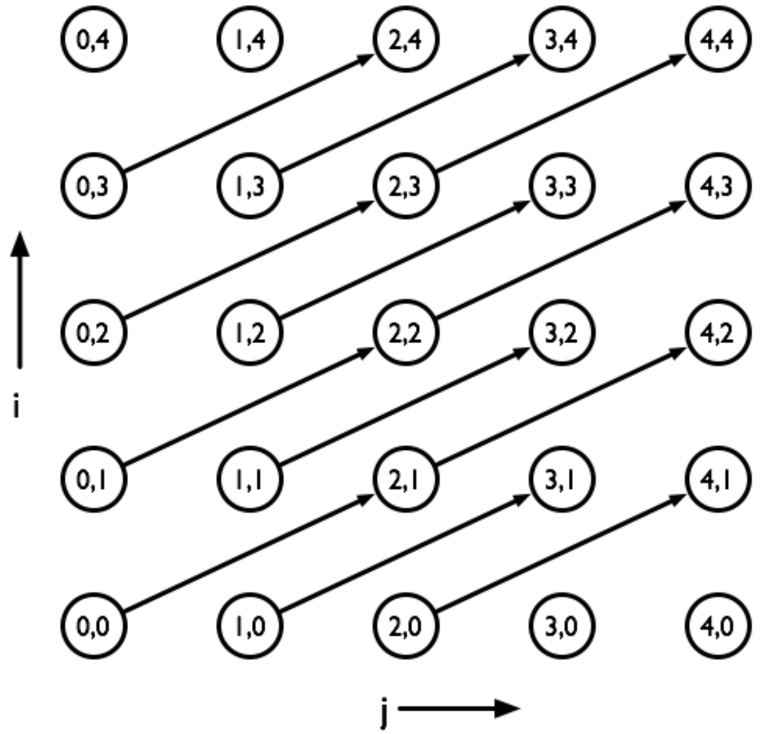
- Distance vector (1, 2)

- Direction vector (+, +)

Loop interchange dependences

- Consider interchanging the following loop, with the dependence graph to the right:

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



- Distance vector (2, 1)
- Direction vector (+, +)
- Distance vector gets swapped!

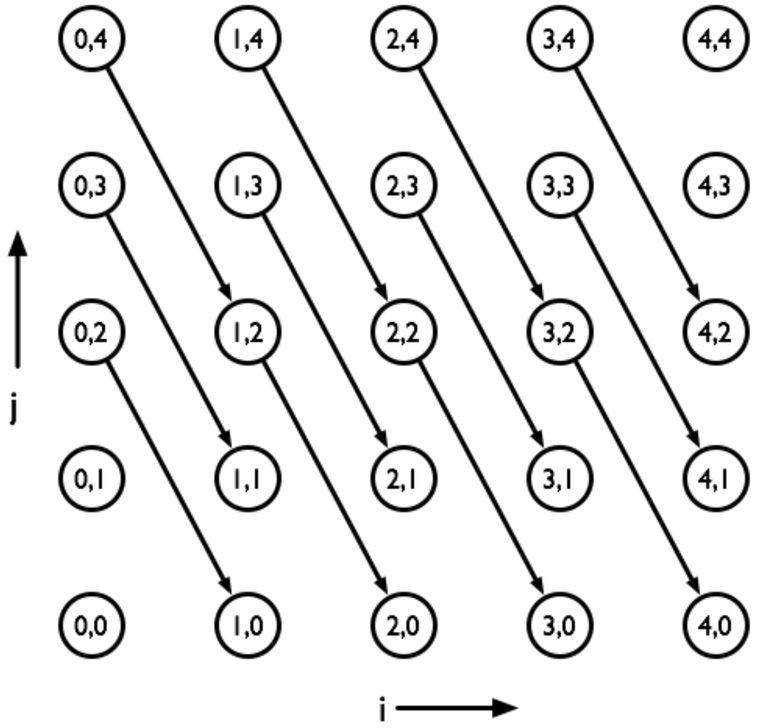
Loop interchange legality

- Interchanging two loops swaps the order of their entries in distance/direction vectors
  - (0, +) → (+, 0)
  - (+, 0) → (0, +)
- But remember, we can't have backwards dependences
  - (+, –) → (–, +)
  - Illegal dependence → Loop interchange not legal!

Loop interchange dependences

- Example of illegal interchange:

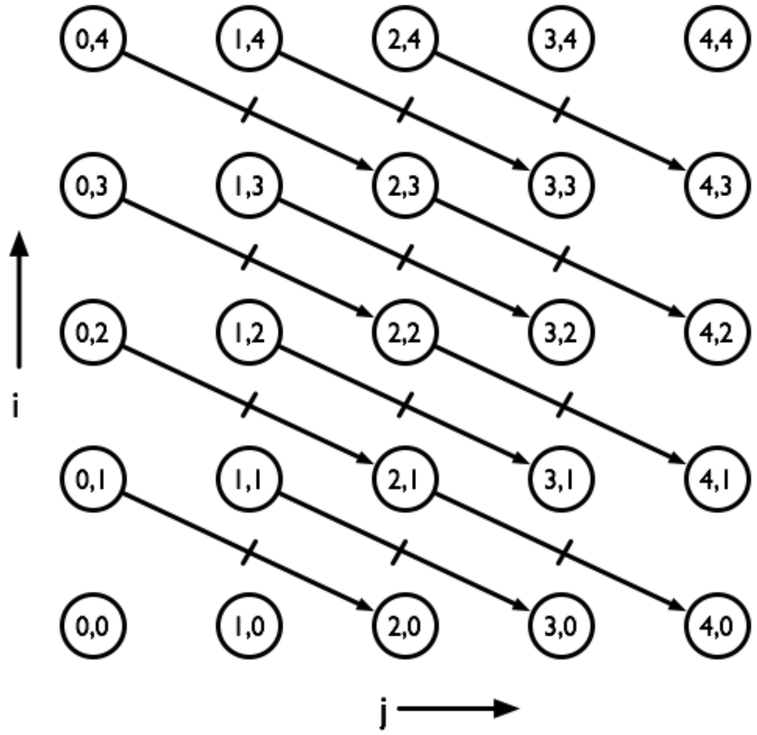
```
for (i = 0; i < N; i++)
 for (j = 0; j < N; j++)
  a[i+1][j-2] = a[i][j] + 1
```



Loop interchange dependences

- Example of illegal interchange:

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



- Flow dependences turned into anti-dependences
  - Result of computation will change!

Loop fusion/distribution

- Loop fusion: combining two loops into a single loop
  - Improves locality, parallelism
- Loop distribution: splitting a single loop into two loops
  - Can increase parallelism (turn a non-parallelizable loop into a parallelizable loop)
- Legal as long as optimization maintains dependences
  - Every dependence in the original loop should have a dependence in the optimized loop
  - Optimized loop should not introduce new dependences

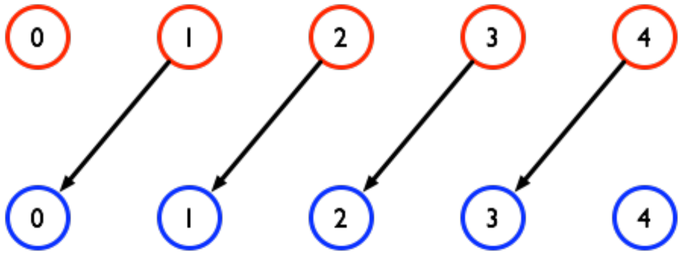
Fusion/distribution example

- Code 1:

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

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

- Dependence graph

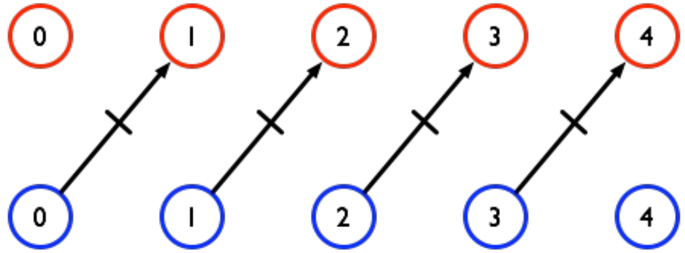


- All red iterations finish before blue iterations → flow dependence

- Code 2:

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

- Dependence graph



- i iterations finish before i+1 iterations → flow dependence now an anti dependence!

Fusion/distribution utility

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

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

Fusion →

← Distribution

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

- Fusion and distribution both legal
- Right code has better locality, but cannot be parallelized due to loop carried dependences
- Left code has worse locality, but blue loop can be parallelized

fin!