DOACROSS Loops

DOACROSS Loops

- “DOACROSS” loops are loops with special loop schedules
  - → Restricted form of loop-carried dependencies
  - → Require fine-grained synchronization protocol for parallelism

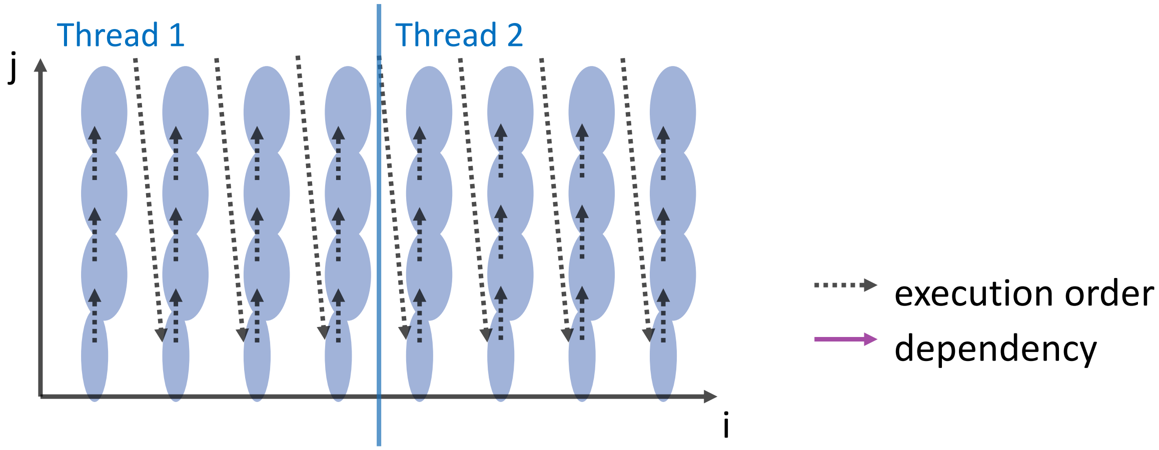
- Loop-carried dependency:
  - → Loop iterations depend on each other
  - → Source of dependency must scheduled before sink of the dependency

- DOACROSS loop:
  - → Data dependency is an invariant for the execution of the whole loop nest

Parallelizable Loops

- A parallel loop cannot not have any loop-carried dependencies (simplified just a little bit!)

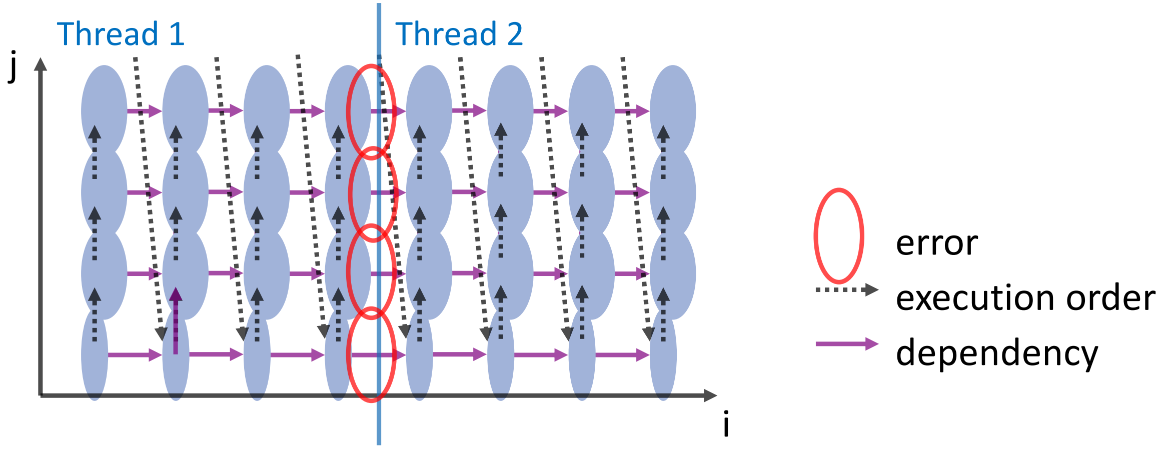
```
for (int i = 1; i < N; ++i) {
    for (int j = 1; j < M; ++j) {
        b[i][j] = f(b[i][j],
                    b[i][j], a[i][j]);
    }
}
```



Non-parallelizable Loops

- If there is a loop-carried dependency, a loop cannot be parallelized anymore (“easily” that is)

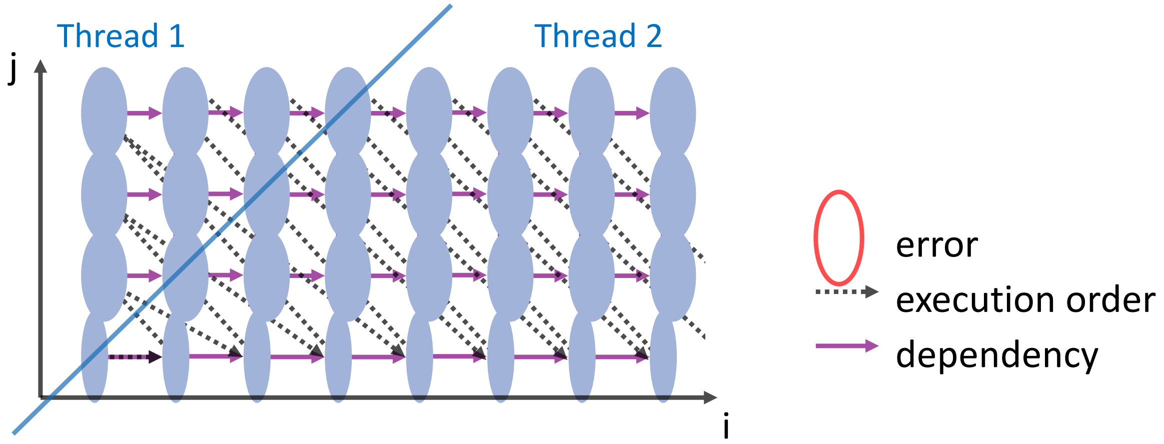
```
for (int i = 1; i < N; ++i) {
    for (int j = 1; j < M; ++j) {
        b[i][j] = f(b[i-1][j],
                    b[i][j-1], a[i][j]);
    }
}
```



Wavefront-Parallel Loops

- If the data dependency is invariant, then skewing the loop helps remove the data dependency

```
for (int i = 1; i < N; ++i) {
    for (int j = i+1; j < i+N; ++j) {
        b[i][j-i] = f(b[i-1][j-i],
                      b[i][j-i-1], a[i][j]);
    }
}
```



DOACROSS Loops with OpenMP

- OpenMP 4.5 extends the notion of the ordered construct to describe loop-carried dependencies
- Syntax (C/C++):

```
#pragma omp for ordered(d) [clause[[,] clause],…]
for-loops
```

  and

```
#pragma omp ordered [clause[[,] clause],…]
```

  where clause is one of the following:

```
  depend(source)
  depend(sink:vector)
```

- Syntax (Fortran):

```
!$omp do ordered(d) [clause[[,] clause],…]
do-loops
!$omp ordered [clause[[,] clause],…]
```

Example

- The ordered clause tells the compiler about loop-carried dependencies and their distances

```
#pragma omp parallel for ordered(2)
for (int i = 1; i < N; ++i) {
    for (int j = 1; j < M; ++j) {
#pragma omp ordered depend(sink:i-1,j) depend(sink:i,j-1)
        b[i][j] = f(b[i-1][j],
                    b[i][j-1], a[i][j]);
    }
#pragma omp ordered depend(source)
}
```

Example: 3D Gauss-Seidel

```
#pragma omp for ordered(2) private(j,k)
for (i = 1; i < N-1; ++i) {
  for (j = 1; j < N-1; ++j)   {
#pragma omp ordered depend(sink: i-1,j-1) depend(sink: i-1,j) \
                    depend(sink: i-1,j+1) depend(sink: i,j-1)
    for (k = 1; k < N-1; ++k) {
      double tmp1 = (p[i-1][j-1][k-1] + p[i-1][j-1][k] + p[i-1][j-1][k+1]
                     + p[i-1][j][k-1] + p[i-1][j][k] + p[i-1][j][k+1]
                     + p[i-1][j+1][k-1] + p[i-1][j+1][k] + p[i-1][j+1][k+1]);
      double tmp2 = (p[i][j-1][k-1] + p[i][j-1][k] + p[i][j-1][k+1]
                     + p[i][j][k-1] + p[i][j][k] + p[i][j][k+1]
                     + p[i][j+1][k-1] + p[i][j+1][k] + p[i][j+1][k+1]);
      double tmp3 = (p[i+1][j-1][k-1] + p[i+1][j-1][k] + p[i+1][j-1][k+1]
                     + p[i+1][j][k-1] + p[i+1][j][k] + p[i+1][j][k+1]
                     + p[i+1][j+1][k-1] + p[i+1][j+1][k] + p[i+1][j+1][k+1]);
      p[i][j][k] = (tmp1 + tmp2 + tmp3) / 27.0;
    }
#pragma omp ordered depend(source)
  }
}
```