CS475 data dependence
Wim Bohm, Sanjay Rajopadhye
Loop carried dependence

\[
\text{for } (i=1; i<N; i++)
\]
\[
A[i] = f(A[i-1])
\]

There is a dependence from loop iteration \(i-1\) to iteration \(i\):

\(A[i]\) is defined in terms of \(A[i-1]\) and thus

loop iteration \(i\) needs to come after loop iteration \(i-1\)

therefore, we cannot parallelize this loop!
Data dependence

Consider two statement instances \( x \) and \( y \), where \( y \) executes after \( x \) in the sequential version of the program, that we want to parallelize:

- \( y \) depends on \( x \) if \( x \) and \( y \) access (read or write) the same memory location, notation: \( x \leftarrow y \)

- there are different kinds of dependences:
  - \( x:write \leftarrow y:read \) RAW read after write, true dependence
  - \( x:read \leftarrow y:write \) WAR write after read, anti dependence
  - \( x:write \leftarrow y:write \) WAW write after write, output dependence
  - \( x:read \leftarrow y:read \) RAR read after read, input dependence

For the first 3, order matters (changing the order changes the outcome of the program).

For the 4\(^{th}\) it does not, so why do we talk about it? (memory)
Parallelization

When true, anti, or output dependences occur in a sequential program, their order cannot be changed when parallelizing the program. **WHY?**

changing the order changes the outcome of the program
Examples

EX1: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        A[i,j] = f(A[i,j-1], A[i-1,j])

EX2: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        B[j] = f(B[j-1], B[j])

EX3: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        C[i] = f(C[i-1], C[i])
Iteration & Data Space

**Iteration Space**: set of index values that the loop iterators can take, in our 3 examples:
- a rectangular region, with “corners” [1,1] and [N-1, M-1]

**Data Space**: set of values of array indices accessed by the statements in the program
- Ex 1: 2-D array, similar to the iteration space
- Ex 2: 1-D array, bounded by [0, M-1]
- Ex 3: 1-D array, bounded by [0, N-1]

We will concentrate on the iteration space here.
Drawing Spaces

There are many ways to draw a space

We will draw iteration and data spaces like this:

row index i goes down, column index j goes right

<table>
<thead>
<tr>
<th>j=1</th>
<th>j=2</th>
<th>j=3</th>
<th>j=4</th>
<th>j=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
<tr>
<td>i=2</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
<tr>
<td>i=3</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
<tr>
<td>i=4</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
</tbody>
</table>

for (i=1; i<N; i++)
for (j=1; j<M; j++)
...A[i,j]...
Row major execution order
References and Dependences

**Reference**: an occurrence of an (array) variable on either
- left hand side of an assignment (write)
- right hand side of an assignment or in an expression (read)

of a statement in the loop body

**Dependences** specify which iteration points depend on which others
Finding dependences

Very hard problem (undecidable in general) but we have special (decidable) cases, e.g. when the dependences are expressed by linear expressions in loop indices. e.g.,

\[ i-1, \ j+1, \ 2*i+3, \ i,j-k \]

For example, finding true dependences:

An iteration point \([i, j]\) reads a memory location, that one or more iterations may have written.

- Find the writers as a function of \([i,j]\).
- Find the “most recent writer” in this set (again, as a function of \([i,j]\))
Examples

EX1: for (i=1; i<N; i++)
   for (j=1; j<M; j++)
      A[i,j] = f(A[i,j-1], A[i-1,j])

EX2: for (i=1; i<N; i++)
   for (j=1; j<M; j++)
      B[j] = f(B[j-1], B[j])

EX3: for (i=1; i<N; i++)
   for (j=1; j<M; j++)
      C[i] = f(C[i-1], C[i])
Ex1: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        A[i,j] = f(A[i,j-1], A[i-1,j])

Iteration [i,j] depends on:
    [i, j-1] and [i-1, j]
    neighbors on west and north
    and writes A[i,j]
    (read by [i+1,j] and [i,j+1])

Can iterations [i,j] and [i-1,j] execute in parallel?
Can iterations [i,j] and [i,j-1] execute in parallel?
can iterations [i,j] and [i-1,j+1] execute in parallel?
Dependence Graph (Ex 2)

Ex2: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        B[j] = f(B[j-1], B[j])

Iteration [i,j] true-depends on:
    [i, j-1] and [i-1, j]
neighbors on west and north

Iteration [i-1, j+1] reads a memory location that iteration [i, j] will overwrite, so
[i,j] anti-depends on [i-1,j+1], and thus, [i-1,j+1] must be executed before [i, j],
and hence, [i,j] and [i-1,j+1] cannot execute in parallel
for (i=1; i<N; i++)
  for (j=1; j<M; j++)
    C[i] = f(C[i-1], C[i])

totally sequential dependence
Wavefront parallelization

When we know the dependences between iterations (the dependence graph)
analyze to determine which iterations can happen at which time step (hopefully many iterations can happen at the same time step)
rewrite the program to represent this new order
Reorder the loop:
Outer loop executes the diagonals sequentially from top left to bottom right
Inner loop executes in parallel all nodes in one diagonal \( d = i + j \)

Therefore diagonals become time-steps \( t \), and nodes on the diagonals virtual processors \( p \) executing array assignments.
Ex1 wavefront parallelization

t: time, p: processor dependence arrows point Back in Time
Ex1: loop transformation

for (i=1; i<N; i++)
  for (j=1; j<M; j++)
    A[i,j] = f(A[i,j-1], A[i-1,j])

Transform (i,j) into t, p (time, processor)
  t is timestep is diagonal in i,j
  in timestep t we have **parallel** operations
    on processors: p
      // p starts at top side, right side
      // finishes left side, bottom
    for t = 0 to N+M-2
      for p = \text{max}(0,t-M+1) to \text{min}(t,N-1)

      // loop body(t,p)

The diagonals are the timesteps,
The points on the diagonals are the processors
executing the array operations
We transformed the loop from i,j space to p,t space

Before transformation

After transformation

\[ t = 0..M-1: \text{p starts at 0} \]
\[ t \geq M: \text{p starts at t-M+1} \]

\[ t = 0..N-1: \text{p ends at t} \]
\[ t \geq N: \text{p ends at N-1} \]
Ex2: for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        B[j] = f(B[j-1], B[j])

Iteration \([i,j]\) true-depends on:
- \([i, j-1]\) and \([i-1, j]\) neighbors on west and north

Iteration \([i-1, j+1]\) reads a memory location that iteration \([i, j]\) will overwrite, so \([i,j]\) anti-depends on \([i-1,j+1]\), and thus, \([i-1,j+1]\) must be executed before \([i, j]\), and hence, \([i,j]\) and \([i-1,j+1]\) cannot execute in parallel
Ex2 wavefront parallelization

\[ d = 2*i + j \]
Ex2 wavefront parallelization

t: time,
p: processor
dependence arrows point Back in Time