Automatic Tiling of "Mostly-Tileable" Loop Nests

David Wonnacott Tian Jin Allison Lake

Haverford College, Haverford, Pa.

Slides from Dave's IMPACT 2015 presentation, with later annotations/corrections in red.

Loop Tiling [a.k.a. Blocking, Supernode Partitioning]

Idea

- Treat $n{*}n$ iteration space as $\lfloor \frac{n}{b} \rfloor * \lfloor \frac{n}{b} \rfloor$ tiles of size $b{*}b$

Purpose: Optimization

- Improve locality on uniprocessors
- Transfer blocks, reduce false sharing on multicore

Legality (classical conditions):

- “Fully permutable” loop nest, i.e.,
- All elements of all dependence vectors are $\geqslant 0$
- (May be enabled by prior loop transformation)

Are Reductions "Permutable"?

What are the dependences of this loop?

```
sums(i) = 0
for j = 0,size-1 do
    sums(i) = sums(i) + A(i,j)
endfor
```

The Omega Project's "petit" analysis tool says:

```
anti      6: sums(i) --> 6: sums(i)    (+)
flow      6: sums(i) --> 6: sums(i)    (+)
output    6: sums(i) --> 6: sums(i)    (+)
```

"petit -r":

```
reduce    6: sums(i) --> 6: sums(i)    (+)
```

Maybe this?

```
reduce    6: sums(i) --> 6: sums(i)    (*)
```

A Challenging Program with Reductions

Nussinov's algorithm (RNA secondary structure prediction)

$$N(i,j) = \max\left(N(i+1,j-1)+\delta(i,j), \max_{i \leqslant k < j}\left(N(i,k)+N(k+1,j)\right)\right)$$

(i.e., maximize number of base-pair matches.) In code:

```
! N initially all 0
for i = size-1,0,-1 do
  for j = i+1,size-1 do
    for k = i,j-1 do
      N(i,j) = max(N(i,j), N(i,k)+N(k+1,j))
    endfor
    if j-1 >= 0 and i+1 < size and i < j-1 then
      N(i,j) = max(N(i,j), N(i+1,j-1)+match(seq[i], seq[j]))
    endif
  endfor
endfor
```

Tiling Nussinov's Algorithm

Dependences (from petit -r, reductions as * not +):

```
reduce   19: N(i,j) --> 22: N(i,j)       (0,0)
reduce   19: N(i,j) --> 19: N(i,j)       (0,0,*)
flow     19: N(i,j) --> 19: N(i,k)       (0,+,*)  //(0,+,+)
flow     19: N(i,j) --> 19: N(k+1,j)     (+,0,*)
flow     19: N(i,j) --> 22: N(i+1,j-1)   (-1#,1)
flow     22: N(i,j) --> 19: N(i,k)       (0,+)
flow     22: N(i,j) --> 19: N(k+1,j)     (+,0)
flow     22: N(i,j) --> 22: N(i+1,j-1)   (-1#,1)
```

So, is this tileable?

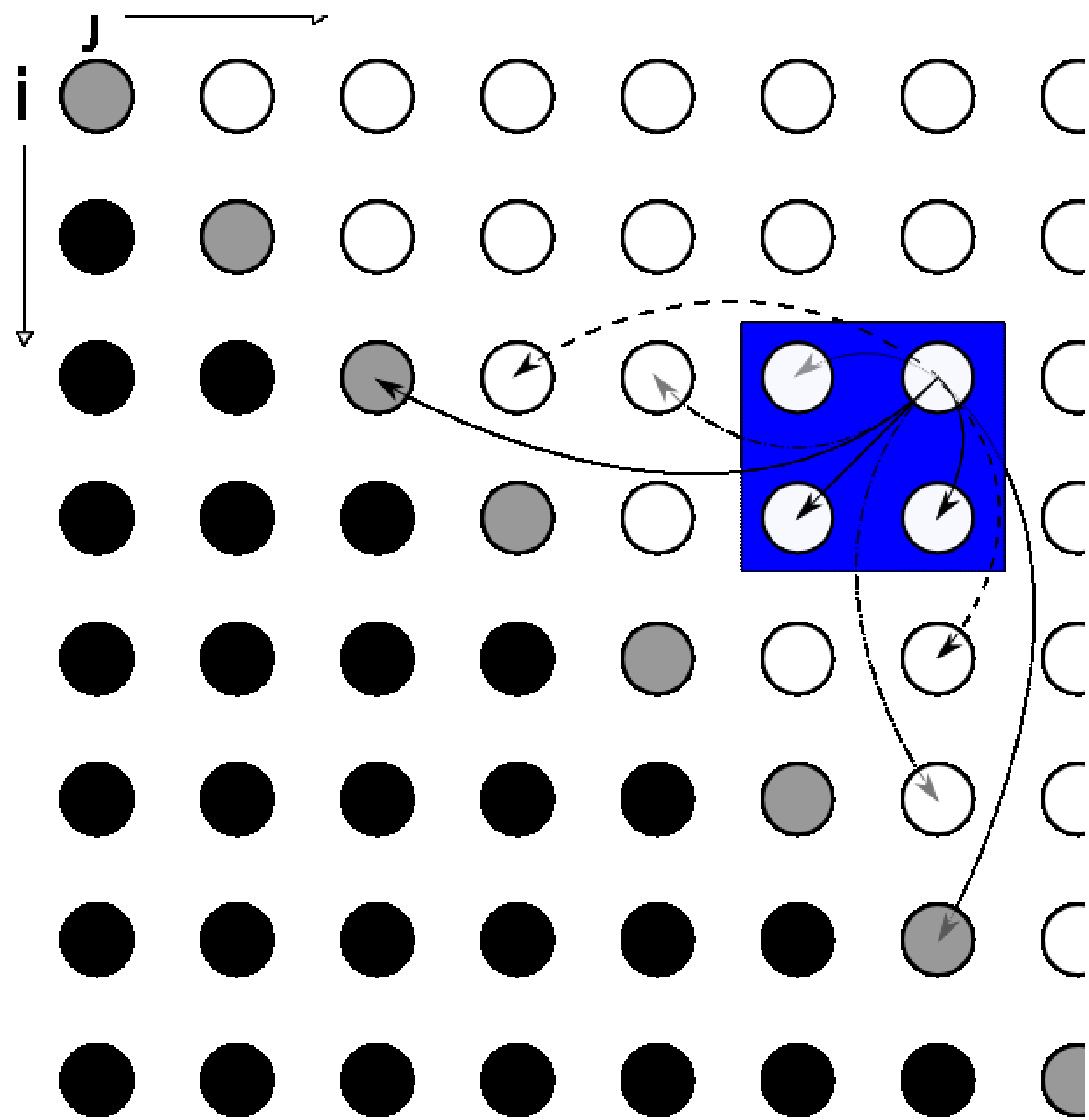
- ? No (or, only i/j), since (0,0,*) is not all $\geqslant 0$
- ? Yes, since (0,0,*) should be (0,0,+) for $\delta, \delta^{-}, \delta^{o}$ note: (+,0,*) also blocks tiling; the dep marked (0,+,*) by petit is actually (0,+,+).
- ? "Mostly", as we shall see...

Tiling Nussinov's Algorithm *Well*

So, is this tileable?

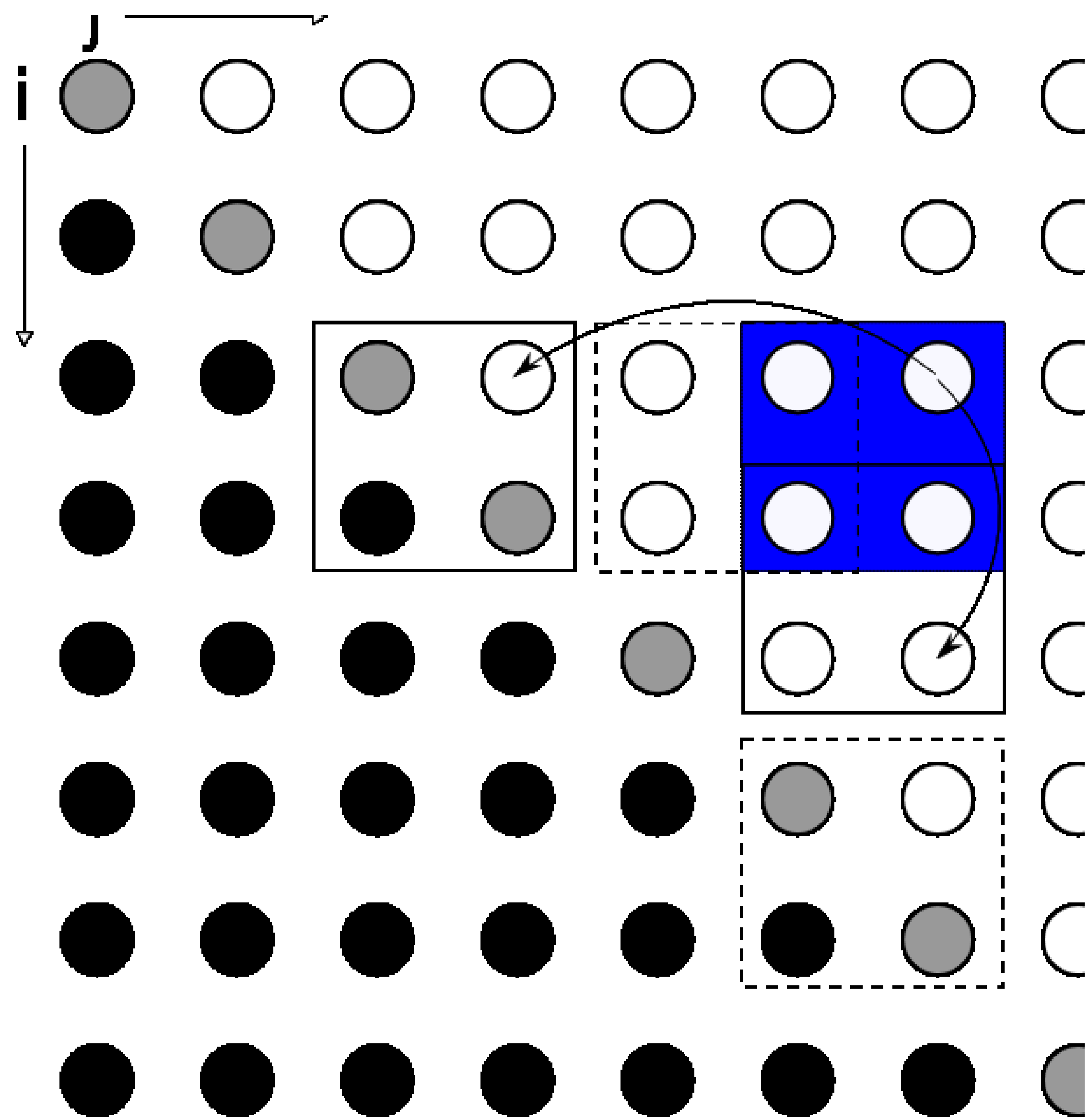
- ? No (or, only i/j), since (0,0,*) is not all $\geqslant 0$
  - correct code, but could be faster...
- ? Yes, since (0,0,*) should be (0,0,+) for $\delta, \delta^-, \delta^o$
  - ***incorrect*** code produced by classical tiling due to the (+, 0, *) flow dependence
- ? "Mostly"? What do I mean by "mostly-tileable"?
  - asymptotically small number of problematic dependences (grow w/tile size, not problem)

Mostly-Tileable Loops of Nussinov's Algorithm



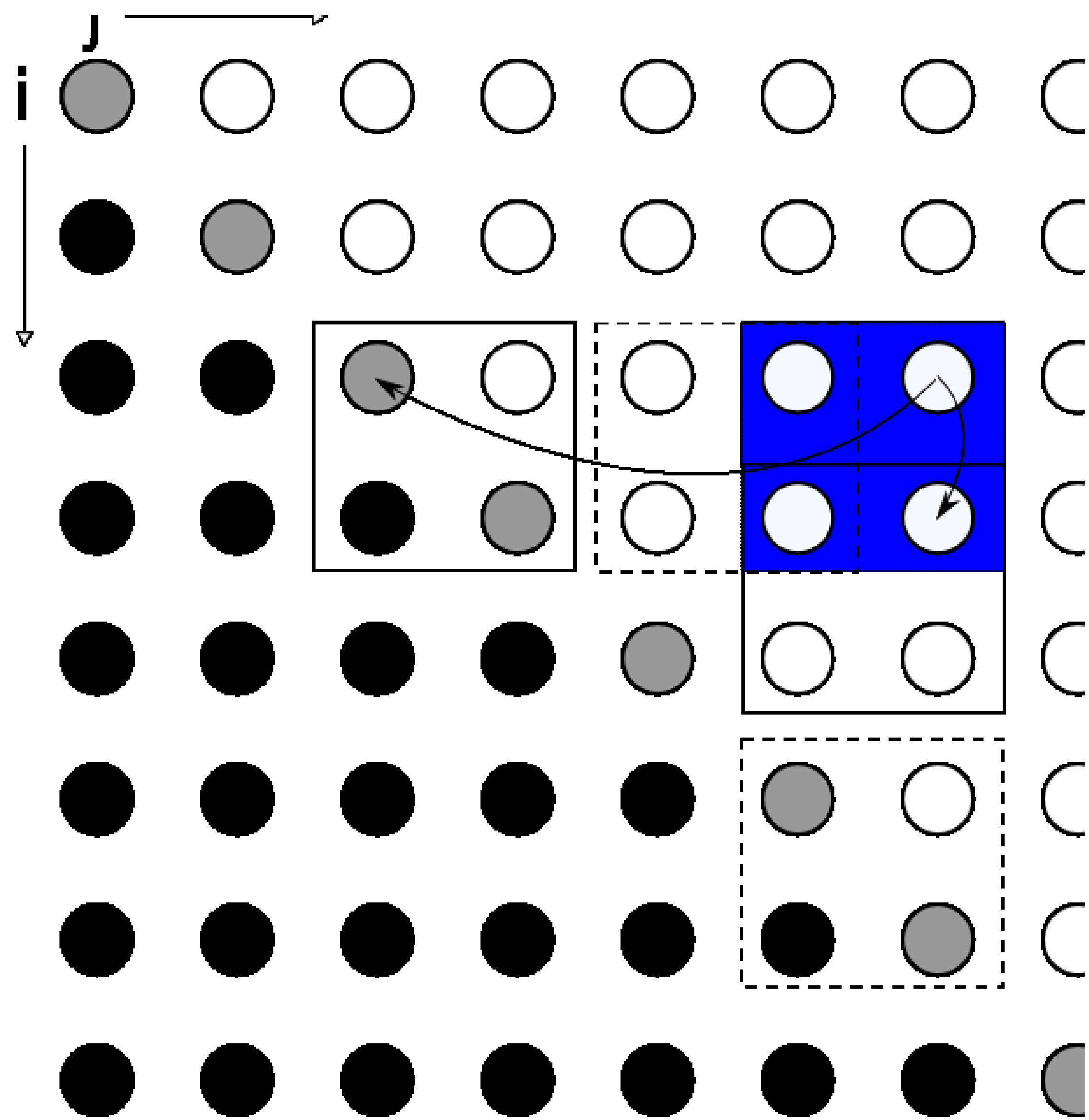
Tiling only the i/j nest works fine, as noted before.

Mostly-Tileable Loops of Nussinov's Algorithm



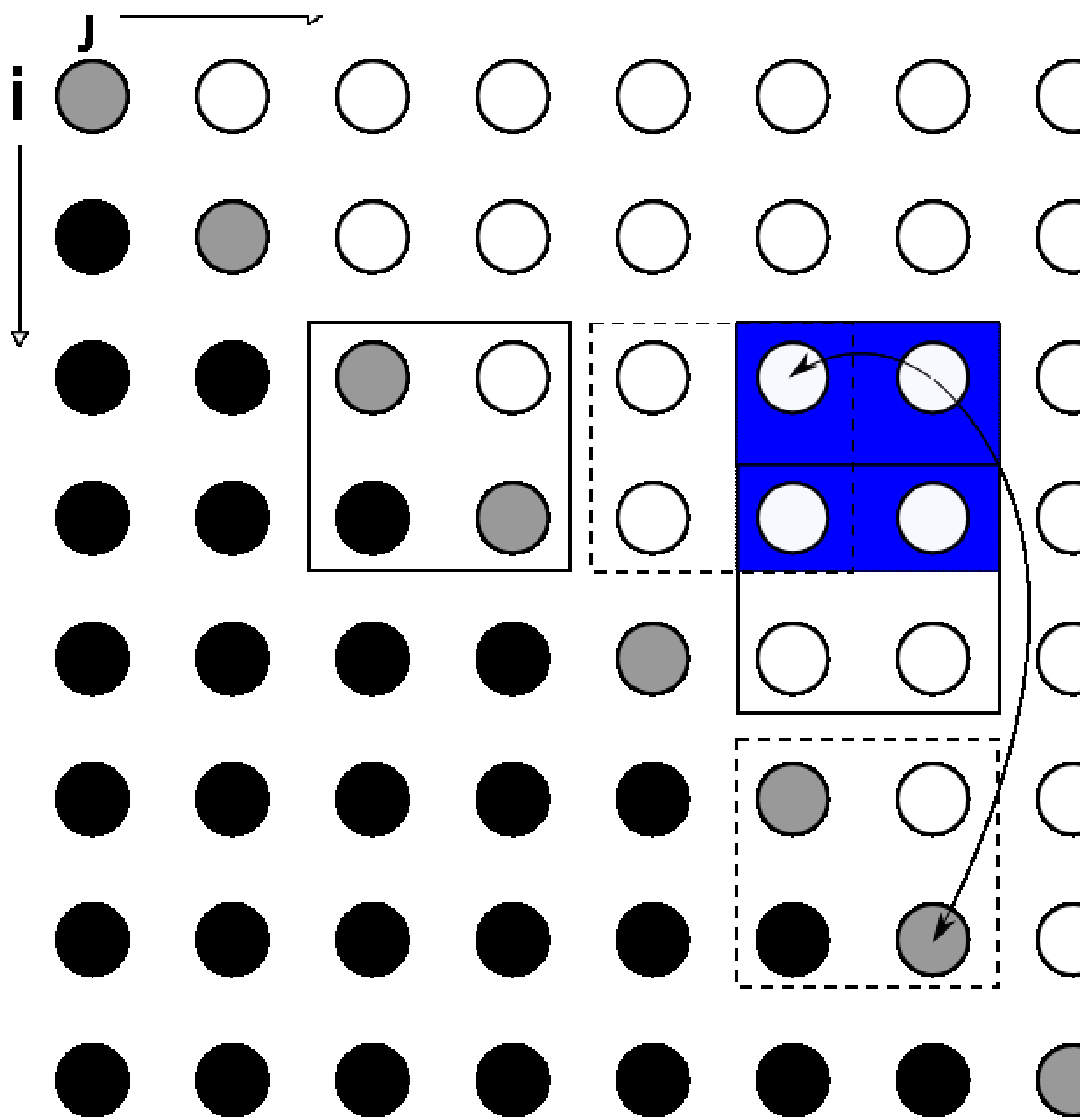
If we group updates from consecutive k, *some* are o.k.

Mostly-Tileable Loops of Nussinov's Algorithm



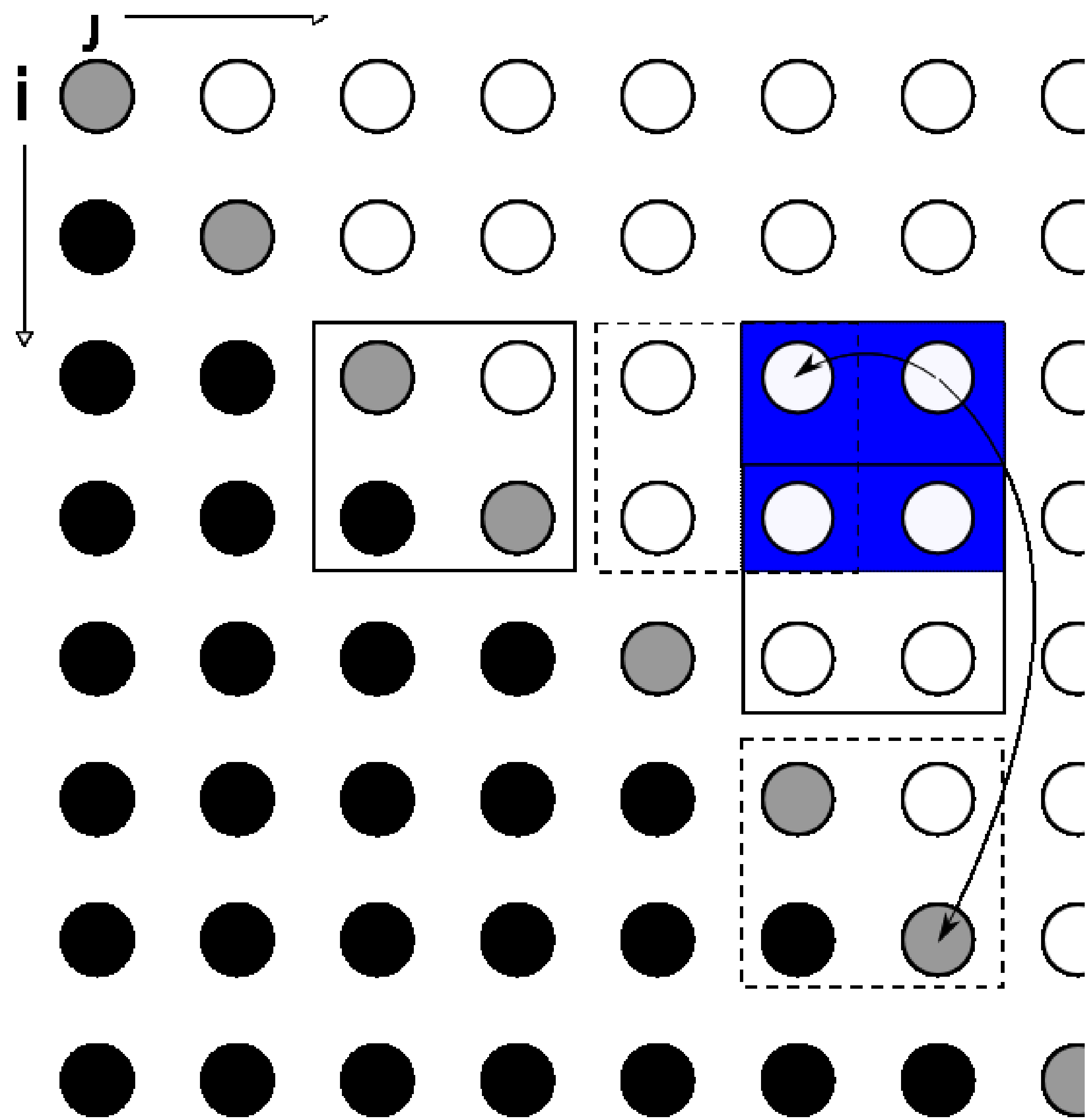
However, some read unfinished elements of updating tile...

Mostly-Tileable Loops of Nussinov's Algorithm



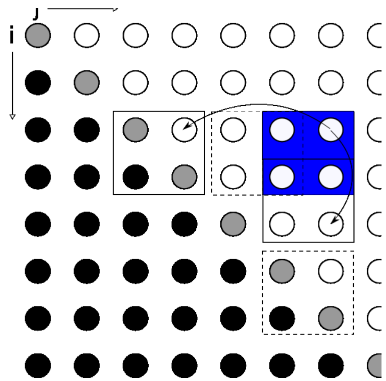
... for *any order* of the k-loop's tiles.

Mostly-Tileable Loops of Nussinov's Algorithm



... for *any order* of the k-loop's tiles. **:-(**

Tiling Mosty-Tileable Loop Nests

Recall that some updates were fine: 

As problem size grows, these outnumber problems, so:

- Tile loop nest *ignoring the reduction*
- "Peel" problematic iterations of k (index-set splitting)
- Execute
  - tiled non-problematic iterations
  - then peeled iterations

How Best to Generalize This

What should we ignore to find mostly-tileable nests?

- Just (all) reductions? actually, these aren't the problem
- Identify direction of reductions as in [GR06]?
- Ignore some other "problematic" dependences?
- Current plan: check all not-fully-tileable nests to see if O(card(problem iterations))$<O$(card(non-problem iterations))

Best choice may depend on which problems can benefit...
So, what other problems look interesting?

- Other dynamic programming (e.g., bioinformatics)
  - Note: some is fully tileable without peeling
- Circular-Stencils? Yes? No? Still thinking....
- *Your thoughts?*