# New Tiling Techniques to Improve Cache Temporal Locality

**Yonghong Song and Zhiyuan Li**

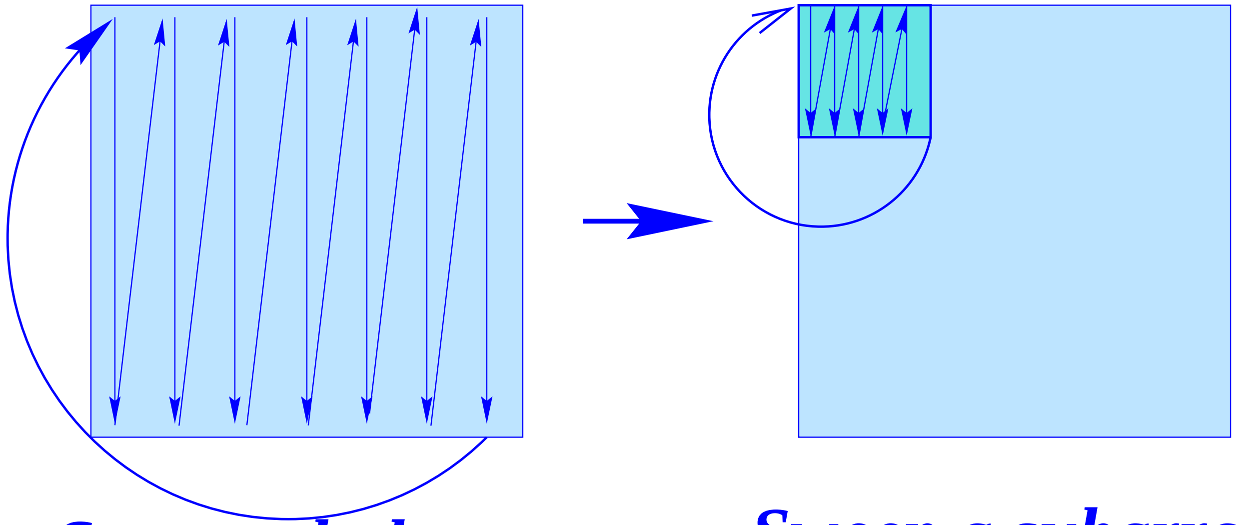
**Department of Computer Science**

**Purdue University**

## Objective

- Improve cache temporal locality on uniprocessor workstations.
- Target data structures: arrays.

# What Is Tiling?



*Sweep whole array* *Sweep a subarray*

- Loop strip-mining + interchange
- To make the working set fit in the cache

## Contributions

- Tiling imperfectly-nested loops
- Speculative execution to enable tiling
- Speedup on real machines:

  9% to 164% speedup on a uniprocessor workstation,
  with two SPECfp benchmarks and a jacobi kernel

# Program Model

DO $T = 1, ITMAX$ *Time-Step Loop*

DO $J_1 = L_1 + b_1 * T, U_1 + b_1 * T$ *Tiling Candidate*

$\dots$

END DO

$\dots$

DO $J_m = L_m + b_m * T, U_m + b_m * T$ *Tiling Candidate*

$\dots$

END DO

END DO

## After Tiling

DO $JJ = f1(SLOPE), f2(SLOPE), NSTEP$

  DO $T = g1(JJ, SLOPE, NSTEP), g2(JJ, SLOPE, NSTEP)$

      DO $J_1 = L'_1, U'_1$   *Tiled Loop*

        $\ldots$

      END DO

      $\ldots$

      DO $J_m = L'_m, U'_m$   *Tiled Loop*

        $\ldots$

      END DO

  END DO

END DO

# Example: Jacobi

DO $T = 1, ITMAX$

  DO $J_1 = 2, N-1$

    DO $I_1 = 2, N-1$

      $L(I_1, J_1) = (A(I_1+1, J_1) + A(I_1-1, J_1) + A(I_1, J_1+1) + A(I_1, J_1-1))/4$

    END DO

  END DO

  DO $J_2 = 2, N-1$

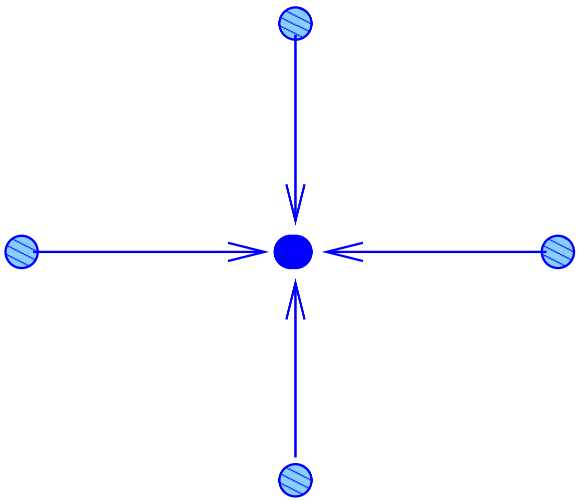
    DO $I_2 = 2, N-1$

      $A(I_2, J_2) = L(I_2, J_2)$

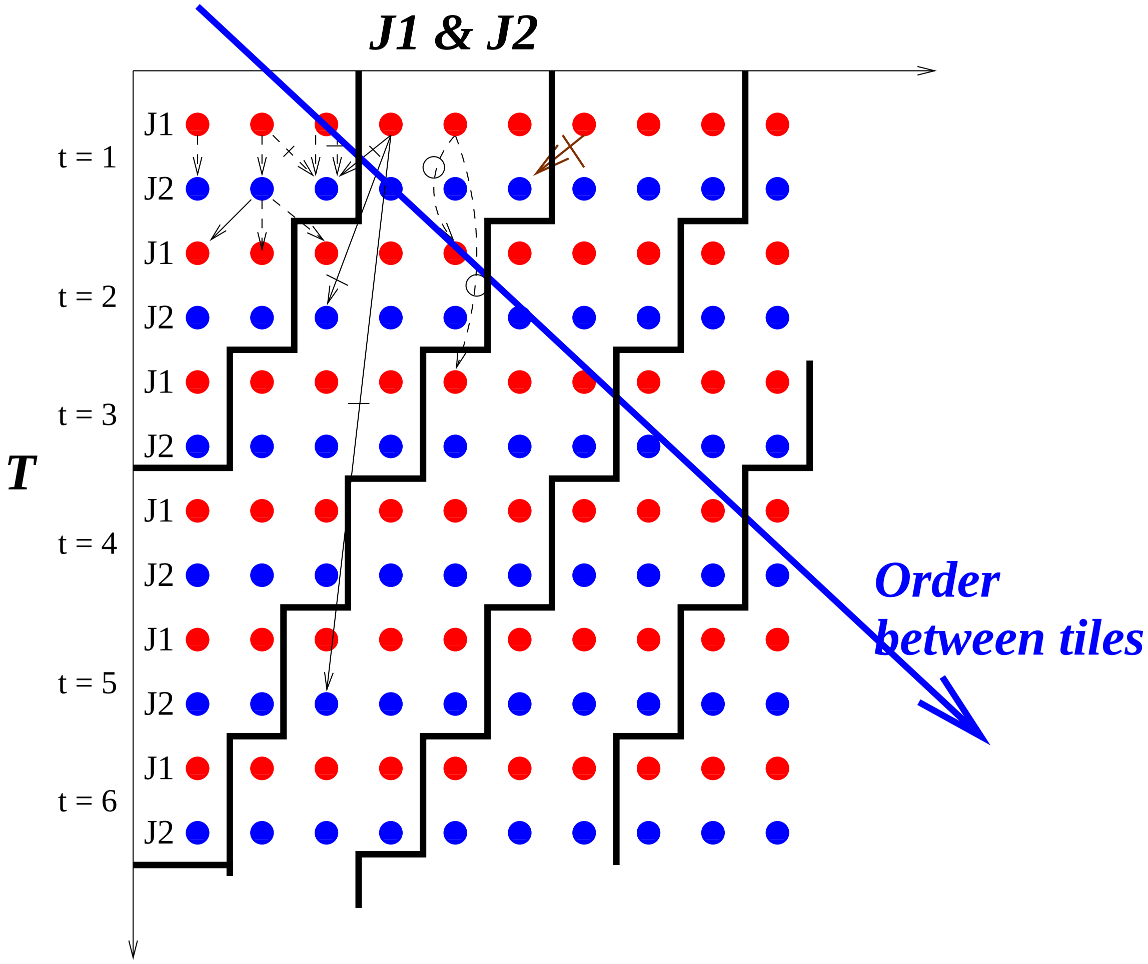
    END DO

  END DO

END DO

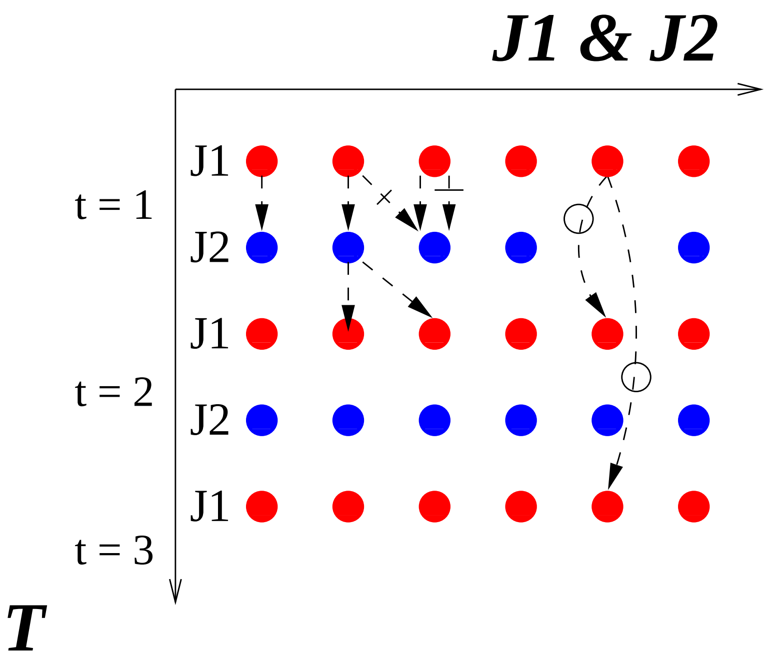


# Tiling



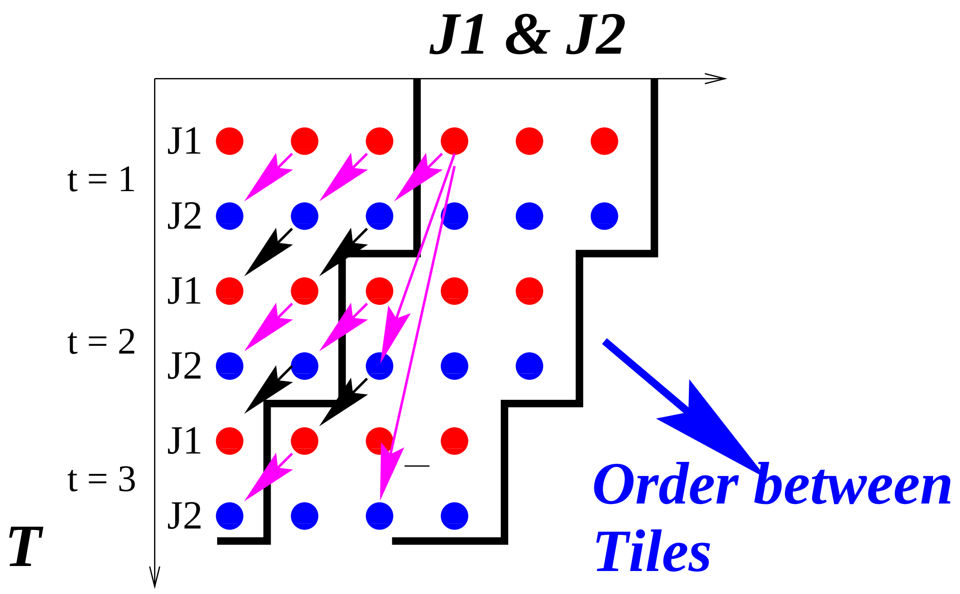
# Forward Dependences

DO $T = 1, ITMAX$
DO $J_1 = 2, N-1$
DO $I_1 = 2, N-1$
$L(I_1, J_1) = (A(I_1+1, J_1) + A(I_1-1, J_1) + A(I_1, J_1+1) + A(I_1, J_1-1))/4$
END DO
END DO
DO $J_2 = 2, N-1$
DO $I_2 = 2, N-1$
$A(I_2, J_2) = L(I_2, J_2)$
END DO
END DO
END DO



# Tile Slope, Width and Offset

```
DO T = 1, ITMAX
  DO J1 = 2, N-1
    DO I1 = 2, N-1
      L(I1,J1) = (A(I1+1,J1) + A(I1-1,J1) + A(I1,J1+1) + A(I1,J1-1))/4
    END DO
  END DO
  DO J2 = 2, N-1
    DO I2 = 2, N-1
      A(I2,J2) = L(I2,J2)
    END DO
  END DO
END DO
```



# Before Duplication

DO $T = 1, ITMAX$ *Time-Step Loop*

DO $J_1 = L_1 + b_1 * T, U_1 + b_1 * T$ *Tiling Candidate*

$\ldots \leftarrow B[\ldots]$

END DO

$\ldots$

DO $J_m = L_m + b_m * T, U_m + b_m * T$ *Tiling Candidate*

$B[\ldots] \leftarrow \ldots$

END DO

END DO

## Odd-Even Duplication

IF $(MOD(T,2).EQ.1)$ THEN

$\ldots$ *Tiling Candidate*

$\ldots \leftarrow B[\ldots]$

$\ldots$ *Tiling Candidate*

$B'[\ldots] \leftarrow \ldots$

ELSE

$\ldots$ *Tiling Candidate*

$\ldots \leftarrow B'[\ldots]$

$\ldots$ *Tiling Candidate*

$B[\ldots] \leftarrow \ldots$

$\ldots$

## After Tiling

DO $JJ = f1(SLOPE), f2(SLOPE), NSTEP$

DO $T = g1(JJ, SLOPE, NSTEP), g2(JJ, SLOPE, NSTEP)$

IF $MOD(T, 2).EQ.1)$ THEN

DO $J_1 = L'_1, U'_1$ *Tiled Loop* (Odd Copy)

DO $J_m = L'_m, U'_m$ *Tiled Loop* (Odd Copy)

ELSE

DO $J_1 = L'_1, U'_1$ *Tiled Loop* (Even Copy)

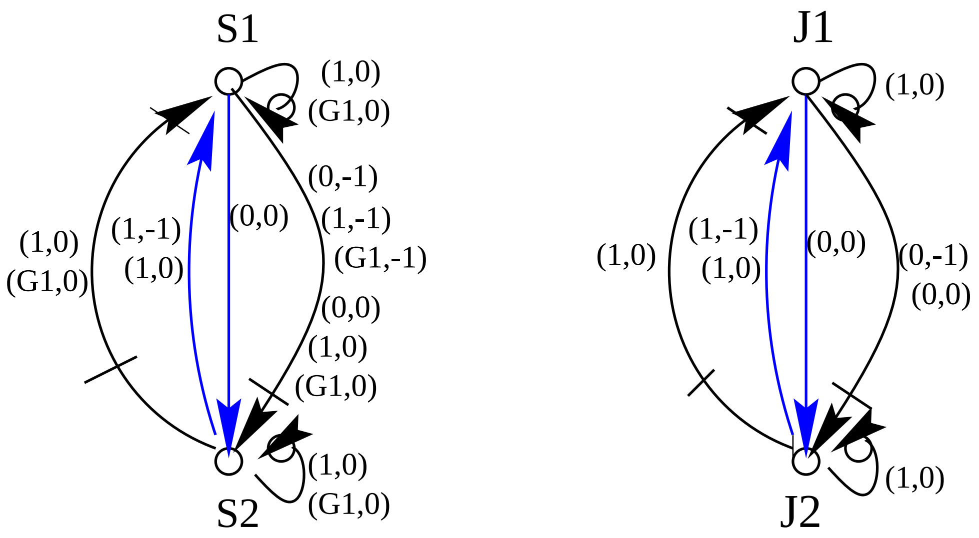
DO $J_m = L'_m, U'_m$ *Tiled Loop* (Even Copy)

END IF

# Outline

- Objective and Contributions
- Basic transformation
- Basic analysis
- Advanced analyses and transformations
- Experimental results
- Related work
- Conclusions and future work

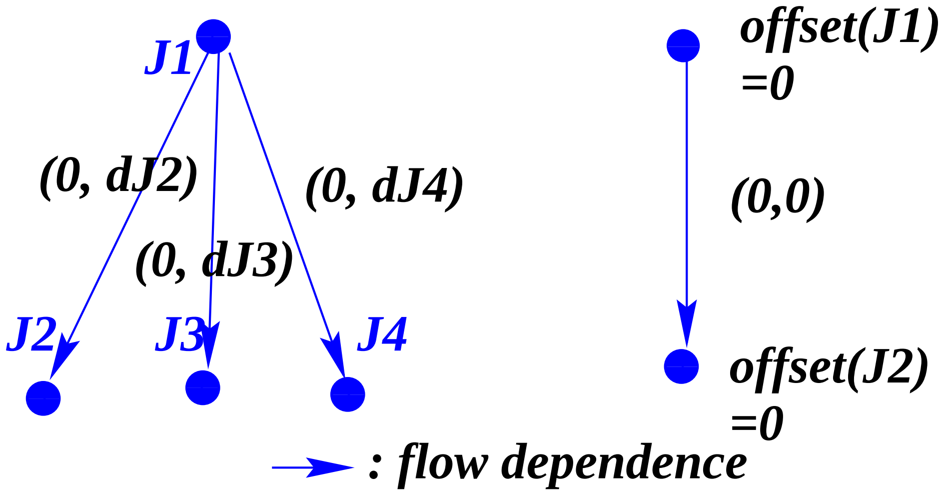
# Dependence Distance Graph




***(dT, dJ)***

# Computing Offsets

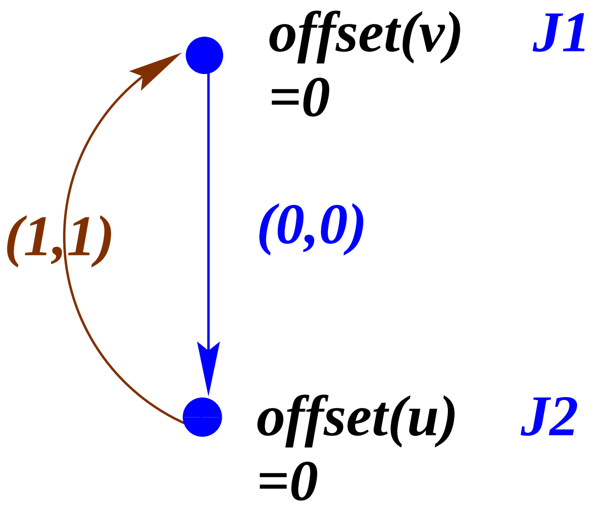
***offset(J1)=max{offset(J2)+|dJ2|, offset(J3)+|dJ3|,offset(J4)+|dJ4|}***




*Examine flow subgraph with dT=0 only.*

# Computing Tile Slope and Width

- $SLOPE = max\{offset(v) + \mid d_J \mid - offset[u]\}$ for all flow edges $< u, v >$ with $d_T = 1$
- $NSTEP = max\{x \mid WSet(x) \leq \alpha * cache\_size\}$



## Profitability Tests

- An overlap of iteration points between $T$ iterations for some $J_i$ loop $(L_i + b_i \leq U_i)$
- No anti- or output backward dependences cross more than one tile boundary
- $NSTEP > SLOPE$

$SLOPE$ must be small.
Backward dependence distance must be short.

## Incompatible Tiled Loops

```
DO T = 1, ITMAX
  DO J1 = 1, N
  DO I1 = 1, N
    ... ← A(I1, J1)
  END DO
  END DO
  DO J2 = 1, N
  DO I2 = 1, N
    A(J2, I2) ← ...
  END DO
  END DO
END DO
```

$$
\begin{array}{l}
\text{DO } T = 1, ITMAX \\
\quad \text{DO } J_1 = 1, N \\
\quad \text{DO } I_1 = 1, N \\
\qquad \ldots \leftarrow A(I_1, J_1) \\
\quad \text{END DO} \\
\quad \text{END DO} \\
\quad \text{DO } J_2 = 1, N \\
\quad \text{DO } I_2 = 1, N \\
\qquad A(J_2, I_2) \leftarrow \ldots \\
\quad \text{END DO} \\
\quad \text{END DO} \\
\text{END DO}
\end{array}
$$

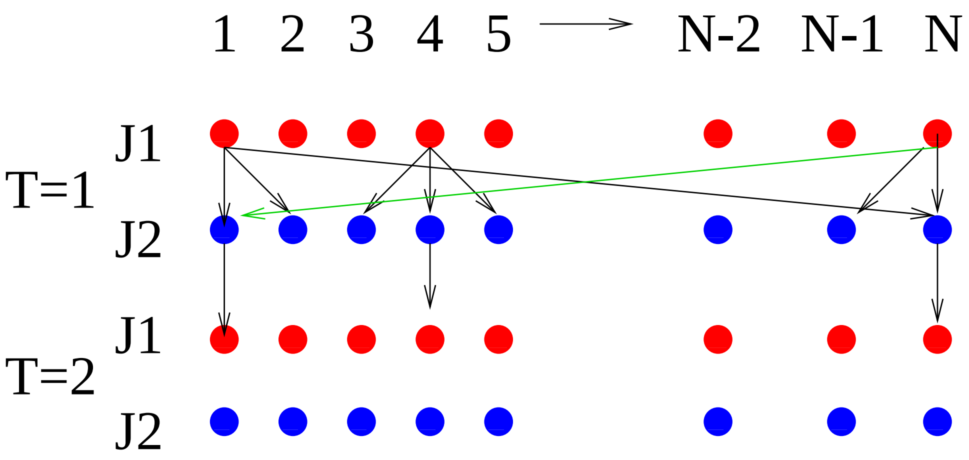
# Making Tiled Loops Compatible

DO $T = 1, ITMAX$
  DO $J_1 = 1, N$
  DO $I_1 = 1, N$
    $\ldots \leftarrow A(I_1, J_1)$
  END DO
  END DO
  $DOI_2 = 1, N$
  $DOJ_2 = 1, N$
    $A(J_2, I_2) \leftarrow \ldots$
  END DO
  END DO
END DO

DO $T = 1, ITMAX$
  DO $I_1 = 1, N$
  DO $J_1 = 1, N$
    $\ldots \leftarrow A(I_1, J_1)$
  END DO
  END DO
  DO $J_2 = 1, N$
  DO $I_2 = 1, N$
    $A(J_2, I_2) \leftarrow \ldots$
  END DO
  END DO
END DO

# Circular Loop Skewing (Before)

DO $T = 1, ITMAX$
  DO $J_1 = 1, N$
    $< BODY_{J_1} >$
  END DO
  DO $J_2 = 1, N$
    $< BODY_{J_2} >$
  END DO
END DO



# Circular Loop Skewing (After)

DO $T = 1, ITMAX$

  DO $J_1 = T, T + N - 1$

  $J_1 = MOD(J_1 - 1, N) + 1$

    $< BODY_{J_1} >$

  END DO

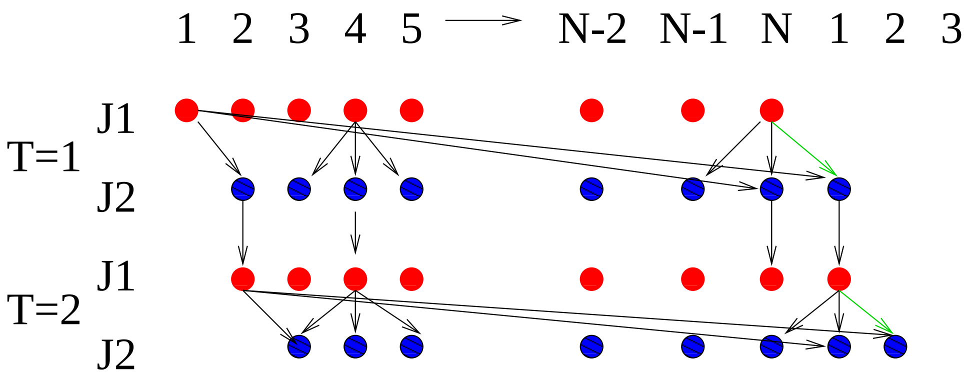
  DO $J_2 = T + 1, T + N$

  $J_2 = MOD(J_2 - 1, N) + 1$

    $< BODY_{J_2} >$

  END DO

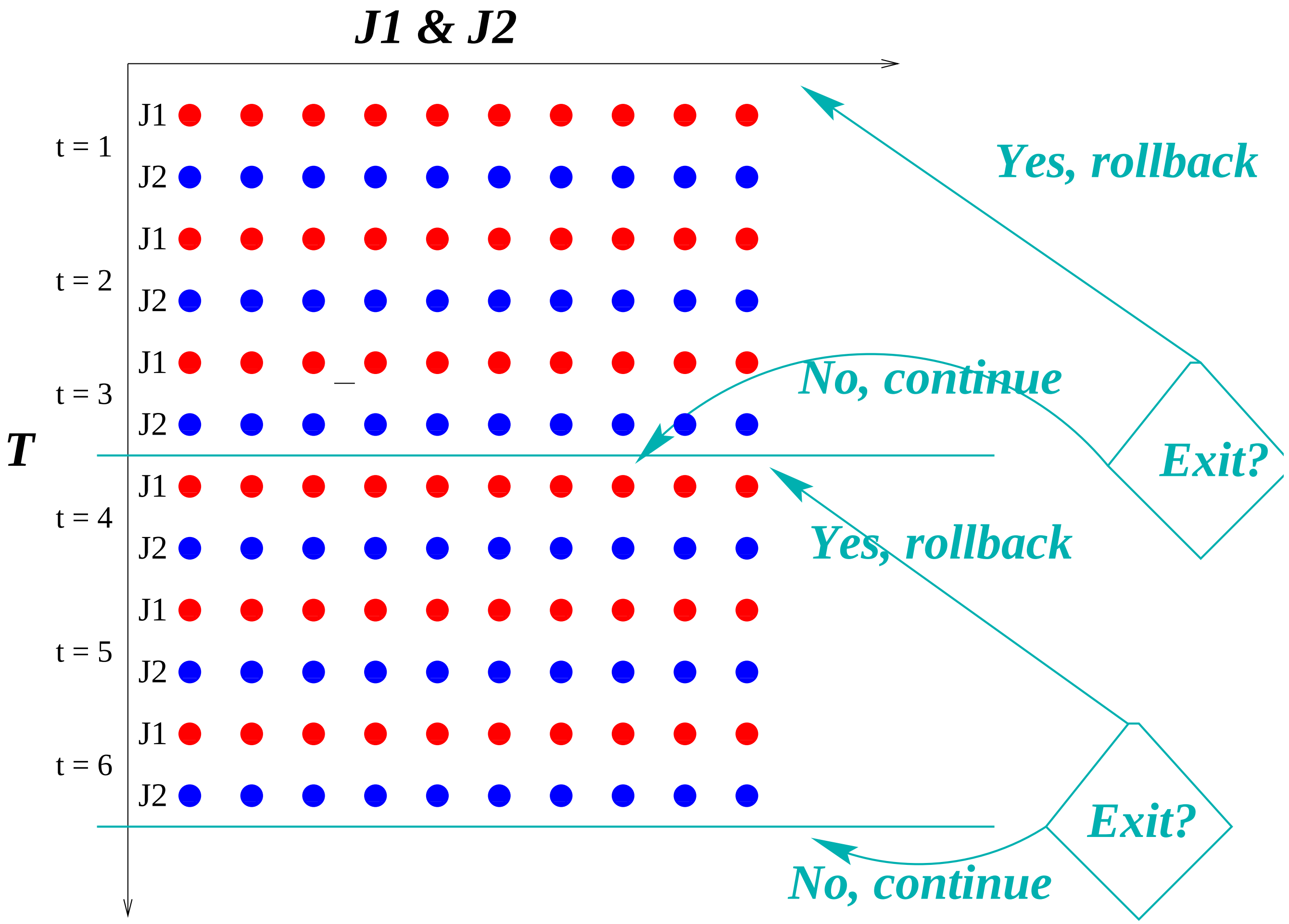
END DO



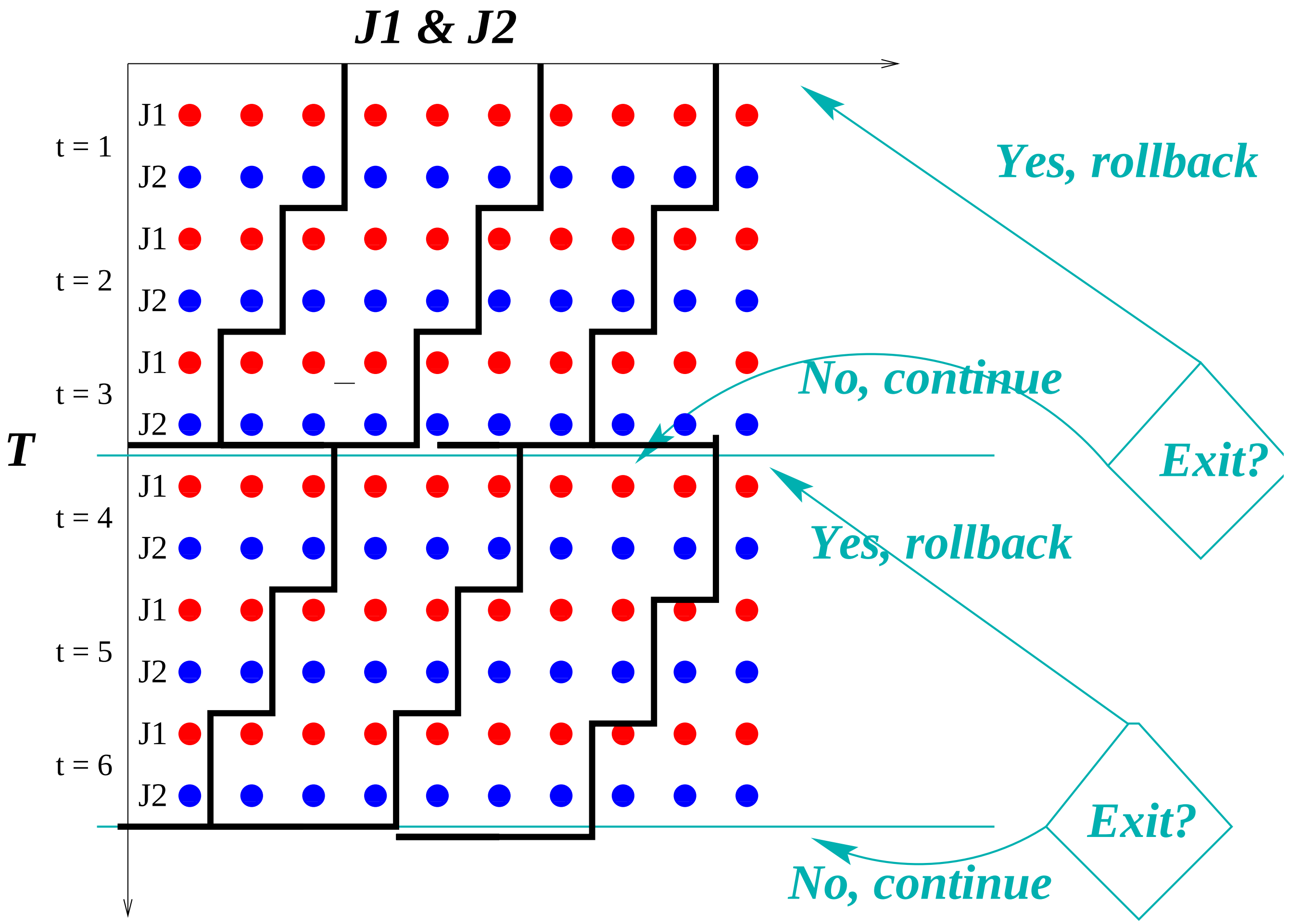
## The Problem of Premature Exits

```
DO T = 1, ITMAX
  DO J_1 = L_1 + b_1 * T, U_1 + b_1 * T
    ...
  END DO
  ...
  DO J_m = L_m + b_m * T, U_m + b_m * T
    ...
  END DO
  IF (exit condition = TRUE) THEN
    GOTO next
  END IF
END DO
next:
```

# Speculative Execution



# Tiled Speculative Execution



# Experiments

- Implemented in the Panorama Fortran Compiler
- Source-to-source transformation result compiled by SGI MIPSPro compiler, with “f77 -O2”
- Executed on MIPS R5K and R10K
  - R5K: 512KB 2-way unified L2 cache
  - R10K: 2MB 2-way unified L2 cache

## Execution Time (Seconds) of Jacobi

| Different Scheme | | Matrix Size for R5K | | | Matrix Size for R10K | | |
|---|---|---|---|---|---|---|---|
| | | 869 | 1024 | 1279 | 869 | 1024 | 1279 |
| Original | Time | 46 | 68 | 102 | 24.43 | 44.65 | 52.93 |
| | Speedup | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Peel-and-fusion | Time | 37 | 50 | 82 | 17.08 | 27.23 | 35.76 |
| | Speedup | 1.24 | 1.36 | 1.24 | 1.43 | 1.64 | 1.48 |
| Our Scheme | Time | 25 | 38 | 60 | 9.25 | 17.23 | 20.14 |
| | Speedup | 1.84 | 1.79 | 1.70 | 2.64 | 2.59 | 2.63 |

## Execution Time (Seconds) of Swim

| Test Program | R10K | | R5K | |
|---|---|---|---|---|
| | Exec. Time | Speedup | Exec. Time | Speedup |
| Orig. Prog. | 326.85 | 1.00 | 625 | 1.00 |
| Trans. Prog. | 215.01 | 1.52 | 573 | 1.09 |

## Execution Time (Seconds) of Tomcatv

| Iter. Orig. | Iter.Trans. | Exec. time | Orig. Time | Speedup |
|---|---|---|---|---|
| 11 | 16 | 25 | 26 | 1.04 |
| 38 | 43 | 47 | 52 | 1.11 |
| 58 | 63 | 63 | 71 | 1.13 |
| 78 | 83 | 79 | 90 | 1.14 |
| 112 | 117 | 105 | 122 | 1.18 |
| 159 | 164 | 143 | 169 | 1.18 |
| 179 | 183 | 158 | 187 | 1.18 |
| 199 | 204 | 175 | 207 | 1.18 |
| 731 | 736 | 593 | 717 | 1.28 |
| 750 | 750 | 434 | 732 | 1.69 |

# Related Work

- Perfect loop nest [Illinois, Rice, Wolf and Lam, Robert *et al* ...]
- Peel-and-fusion [Manjikian *et al.*]
- Data shackling [Kodukula *et al.*]
- Time skewing [McCalpin and Wonnacott]
- Minimum renaming techniques [Strout *et al.*]
- Speculative parallelization [Collard]
- Localizing exit tests [Pugh *et al.*]

# Conclusions and Future Work

- A general tiling scheme based on dependence distances
- Transformations to shorten distances
- Speculative execution for locality enhancement
- Real speedup on real machines
- Future work:
  - Better understanding of tile size
  - More transformations to shorten distances
  - Better understanding of speculative execution
  - Other data structures