

On the Interaction of Tiling and Automatic Parallelization

**Zhelong Pan, Brian Armstrong, Hansang Bae
Rudolf Eigenmann
Purdue University, ECE**

2005.06.01

Outline

- ✓ Motivation
- Tiling and Parallelism
- Tiling in concert with parallelization
- Experimental results
- Conclusion

Motivation

- Apply tiling in a parallelizing compiler (Polaris)
 - Polaris generates parallelized programs in OpenMP
 - Backend compilers generate executable



- Investigate performance on real benchmarks

Issues

- Tiling interacts with parallelization passes
 - Data dependence test, induction, reduction, ...
- Load balancing is necessary
- Parallelism and locality are compromised

Outline

- Motivation
- ✓ **Tiling and parallelism**
- Tiling in concert with parallelization
- Experimental results
- Conclusion

Tiling

- Loop strip-mining
 - L_i strip-mined into L_i' and L_i''
 - Cross-strip loops: L_i'
 - In-strip loops: L_i''
- Loop permutation

(a) Matrix Multiply

```
DO I = 1, N
  DO K = 1, N
    DO J = 1, N
      Z(J,I) = Z(J,I) + X(K,I) * Y(J,K)
```

(b) Tiled Matrix Multiply

```
DO K2 = 1, N, B
  DO J2 = 1, N, B
    DO I = 1, N
      DO K1 = K2, MIN(K2+B-1,N)
        DO J1 = J2, MIN(J2+B-1,N)
          Z(J1,I) = Z(J1,I) + X(K1,I) * Y(J1,K1)
```

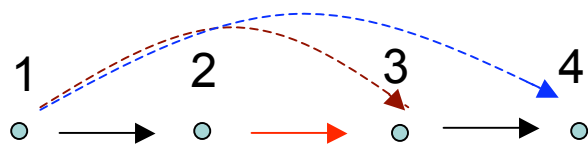
Possible Approaches

- Tiling before parallelization
 - Possible performance degradation
- Tiling after parallelization
 - Possible wrong result
- Our approach
 - *Tiling in concert with parallelization*

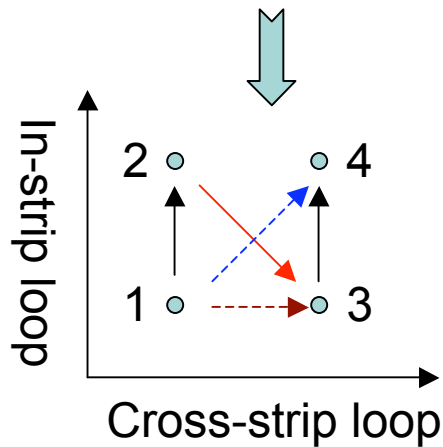
Direction Vector after Strip-mining

Lemma.

Strip-mining may create more direction vectors,
 i.e. $= \rightarrow ==$, $< \rightarrow =<$ or $<^*$, $> \rightarrow =>$ or $>^*$



“<“

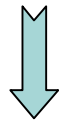



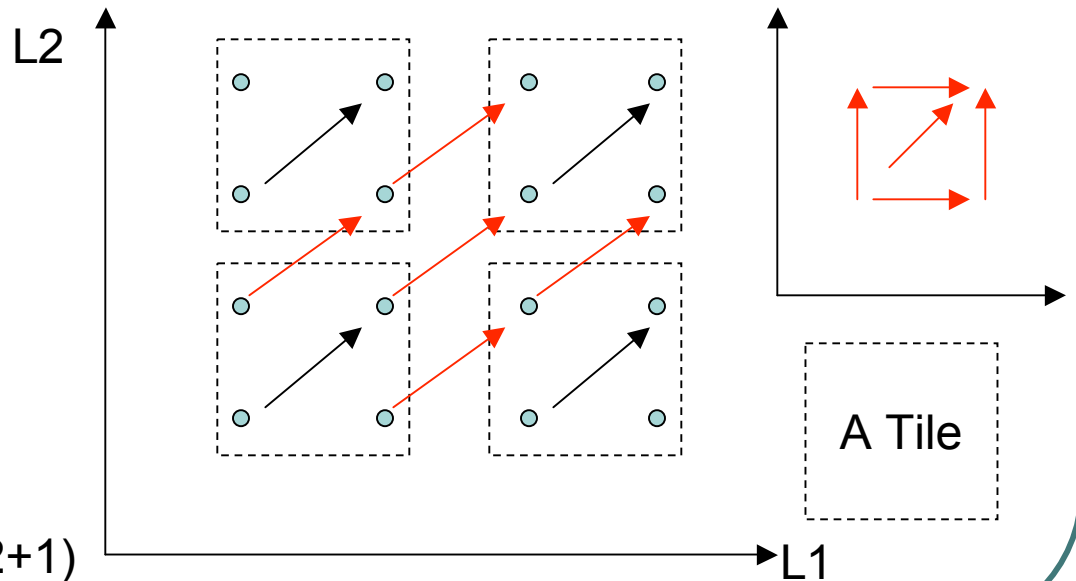
- in-strip dependence, “= $<$ ”
- cross-strip dependence, “<math><>”
- cross-strip dependence, “<math><=”
- cross-strip dependence, “<math><<”

Parallelism after Tiling

Theorem.

Tiling after parallelization is unsafe

$$\begin{array}{l}
 \text{S} \left[\begin{array}{l} \text{DO L1} = 1, 4 \\ \text{P} \left[\begin{array}{l} \text{DO L2} = 1, 4 \\ \text{A(L1,L2)} = \text{A(L1+1,L2+1)} \end{array} \right. \end{array} \right.
 \end{array}$$


$$\begin{array}{l}
 \text{S} \left[\begin{array}{l} \text{DO LC1} = 1, 4, 2 \\ \text{S} \left[\begin{array}{l} \text{DO LC2} = 1, 4, 2 \\ \text{S} \left[\begin{array}{l} \text{DO LI1} = \text{LC1}, \text{LC1}+1, 1 \\ \text{P} \left[\begin{array}{l} \text{DO LI2} = \text{LC2}, \text{LC2}+1, 1 \\ \text{A(LI1,LI2)} = \text{A(LI1+1,LI2+1)} \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.
 \end{array}$$


Outline

- Motivation
- Tiling and Parallelism
- ✓ **Tiling in concert with parallelization**
- Experimental results
- Conclusion

Trading off Parallelism and Locality

- Enhancing locality may reduce parallelism

```
DO J=1,N
```

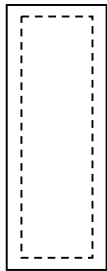
```
DO I=1,N
```

```
A(I,J) = A(I,J+1)
```

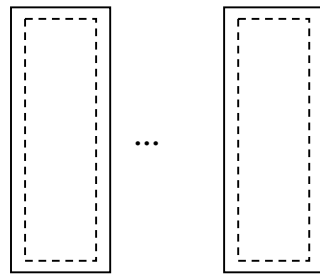
- Tiling may change fork-join overhead
 - [SP] → [SSP], increase fork-join overhead.
 - [SP] → [PSP], decrease fork-join overhead.
 - [PS] → [SPS], increase fork-join overhead.
 - [SS] → [SSS], no change of fork-join overhead.
 - [PP] → [PPP], no change of fork-join overhead.

Tile Size Selection

- Data references in a tile should be close to the cache size.

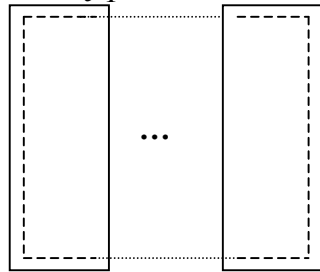


(a) Loop is sequential.
 $Ref_T = CS$



(b) Cross-strip loop is parallel.
Machine has distributed caches.

$$Ref_T = CS$$



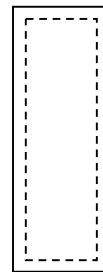
(c) In-strip loop is parallel.
Machine has distributed caches.

$$Ref_T = CS * P$$



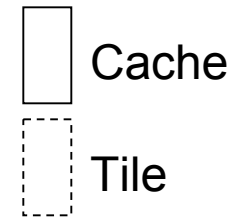
(d) Cross-strip loop is parallel.
Machine has a shared cache.

$$Ref_T = CS / P$$



(e) In-strip loop is parallel.
Machine has a shared cache.

$$Ref_T = CS$$



Ref_T : Mem ref.
in a tile

CS : Cache size

P : # of Proc.

Load Balancing

- Balance the parallel cross-strip loop

(a) Before tiling (balanced)

```
DO I = 1, 512
  DO J = 1, 512
```

(b) After tiling (not balanced)

```
DO J1 = 1, 512, 80
  DO I = 1, 512
    DO J = 1, MIN(J1+79,512)
```

T = 80

P = 4 → S = 64

I = 512

- Balanced tile size

$$S = \frac{I}{\lceil I / (P * T) \rceil * P}$$

S: Balanced tile size

T: Tile size by LRW

P: Number of processors

I: Number of iterations

Impact on parallelization passes

- Tiling does not change the loop body
- Limited effect on parallelization passes
 - Induction variable substitution
 - Privatization
 - Reduction variable recognition

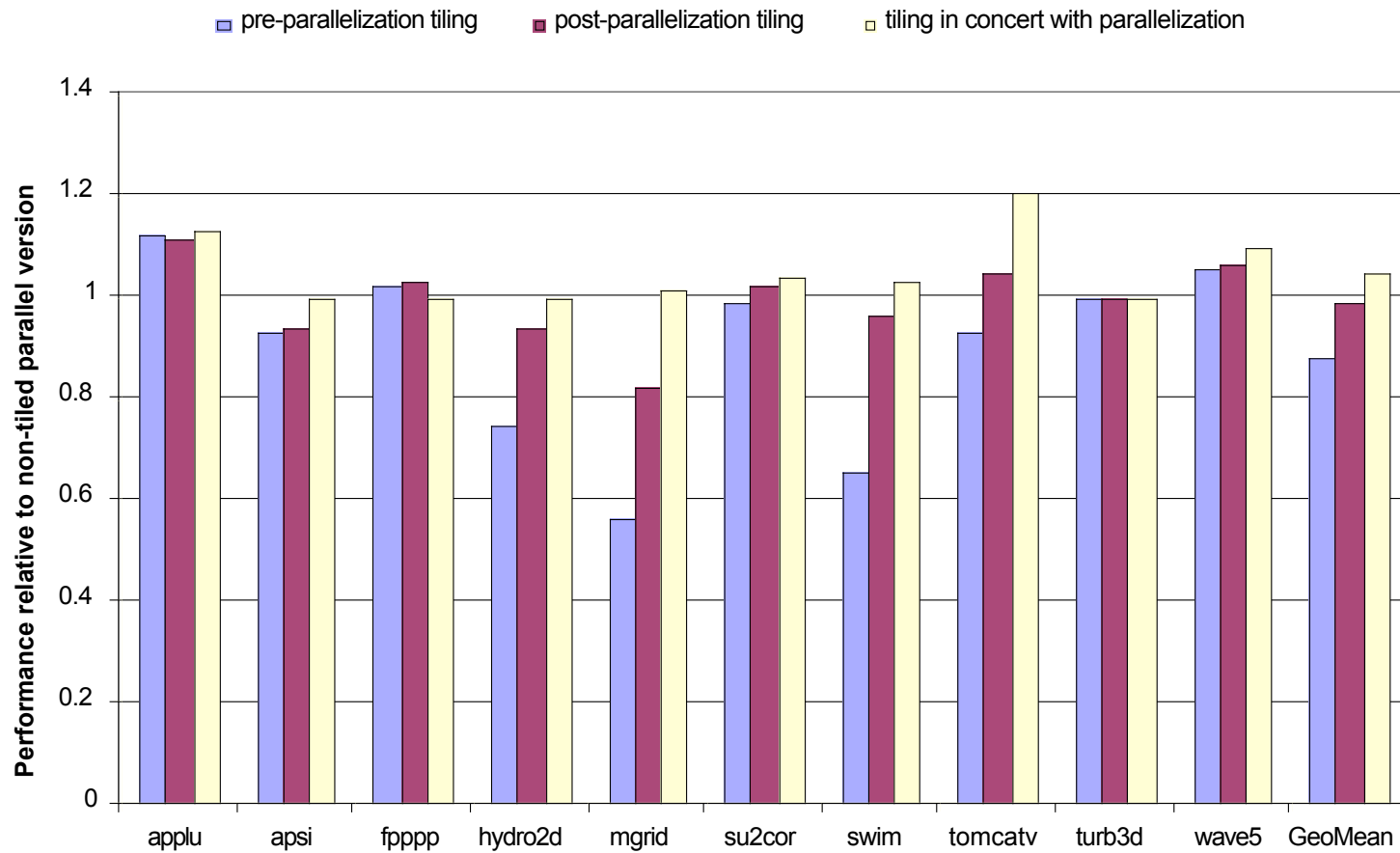
Tiling *in Concert* with Parallelization

- Find the best tiled version in favor of
 - parallelism first and then locality
- Compute tile size based on
 - parallelism and cache configuration
- Tune the tile size to balance load
- Update reduction/private variable attribute
- Generate two versions if iteration number I unknown:
 - Original parallel version is used when I is small
 - Otherwise, tiled version is used

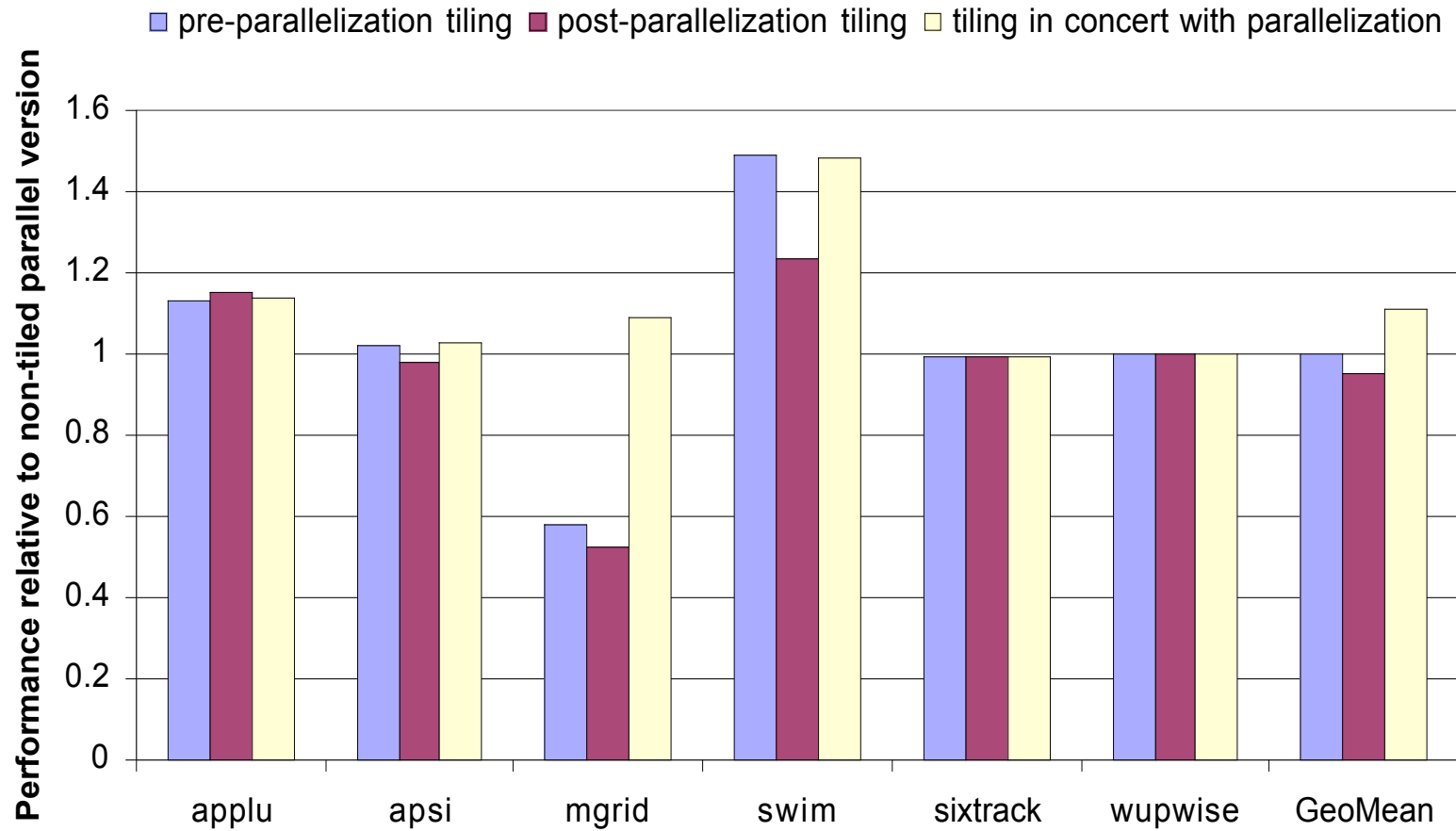
Outline

- Motivation
- Tiling and Parallelism
- Tiling in concert with parallelization
- ✓ **Experimental results**
- Conclusion

Result on SPEC CPU 95



Result on SPEC CPU 2000



On the performance bound

Percentage of tilable loops based on reuse

Benchmark	Total	Reuse	Nested	w/o Call	
APPLU	149	125	55	54	(97.60%)
APSI	388	310	111	59	(19.50%)
FPPPP	49	37	15	8	(5.80%)
HYDRO2D	170	117	21	21	(53.70%)
MGRID	38	24	8	8	(86.40%)
SU2COR	208	177	37	22	(14.90%)
SWIM	24	15	3	3	(60.10%)
TOMCATV	16	14	5	5	(95.90%)
TURB3D	64	43	12	11	(22.20%)
WAVE5	362	274	59	57	(19.70%)

Conclusion

- Tiling and parallelism
- Tiling in concert with parallelization
- Comprehensive evaluation