Cache Optimization for Coarse Grain Task Parallel Processing Using Inter-Array Padding

K. Ishizaka, M. Obata, H. Kasahara
Waseda University, Tokyo, Japan

Research Background

- Wide use of SMP machine
  - From chip multiprocessors to high performance computers
- Increasing the number of processors
  - Gap between peak and effective performance is getting larger
- Automatic parallelizing compiler is important
  - To increase *effective performance* further
    - Multilevel parallel processing
      - Beyond limitation of loop parallelization
    - Locality Optimization
      - Cope with the speed gap between processor and memory
  - To improve *cost performance* and *usability* of SMP

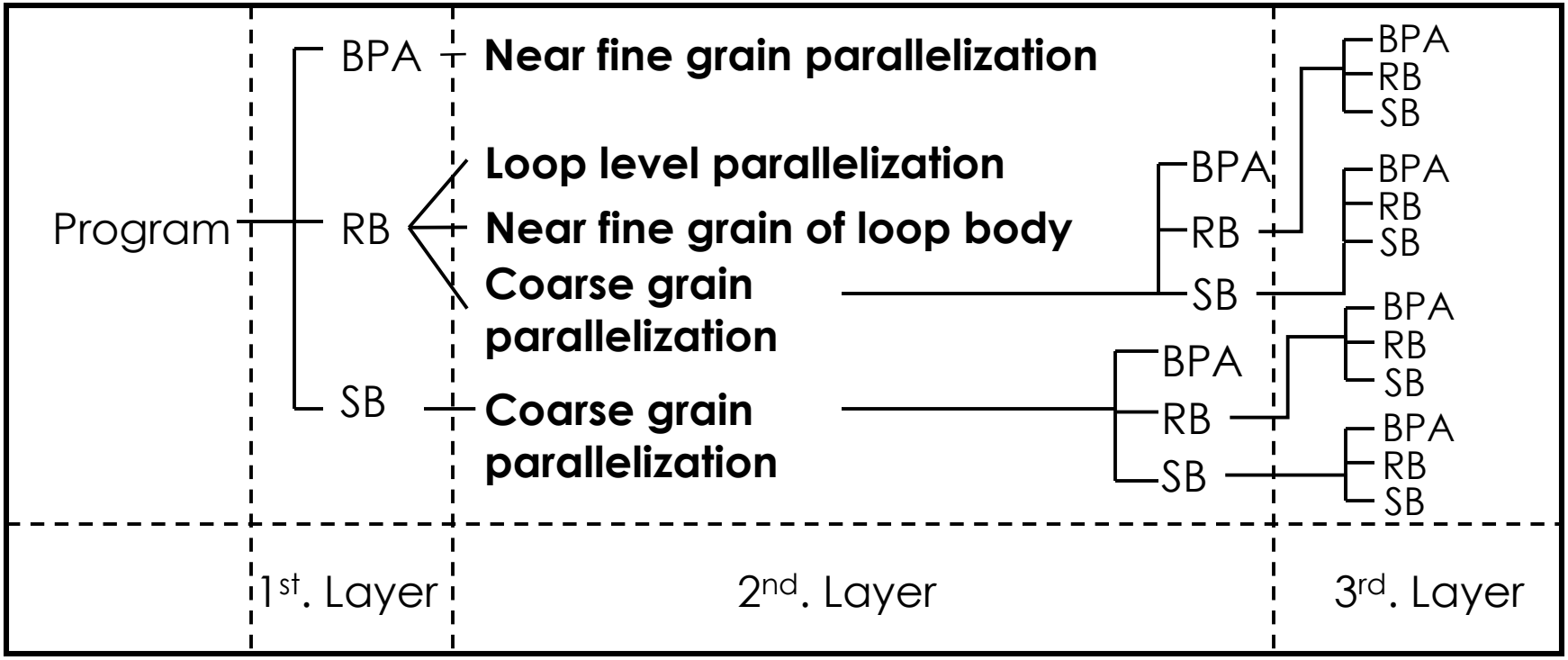
Multigrain Parallelization

- Improvement of effective performance and scalability
- In addition to the loop parallelism
  - Coarse grain task parallelism:
    - subroutines, loops, basic blocks
  - Near fine grain parallelism:
    - statements

➢OSCAR multigrain parallelizing compiler

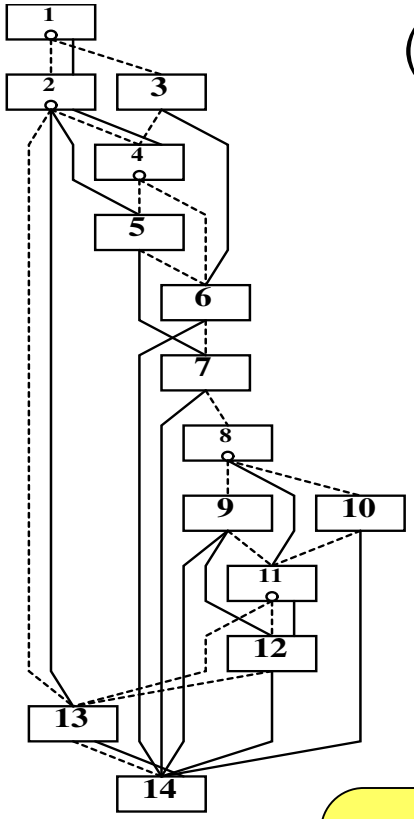
Generation of Coarse Grain Tasks

- Program is decomposed into macro-tasks(MTs)
  - Block of Pseudo Assignments (BPA): Basic Block (BB)
  - Repetition Block (RB) : natural loop
  - Subroutine Block (SB): subroutine



Earliest Executable Condition Analysis

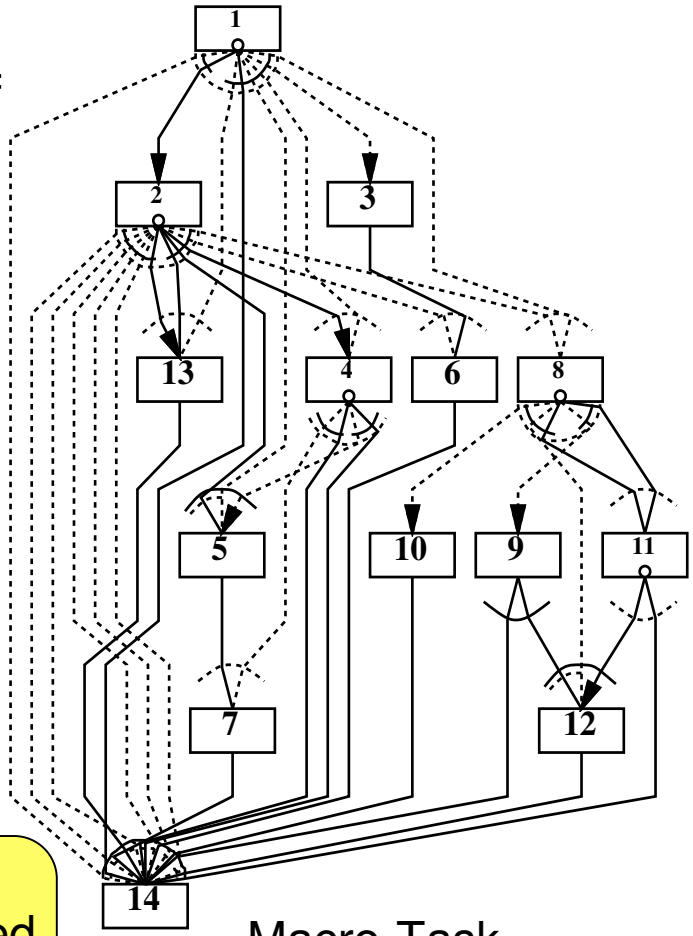
- Conditions on which macro-task may begin its execution earliest



Macro-Flow Graph

(Condition for determination of MT Execution)
AND
(Condition for Data access)

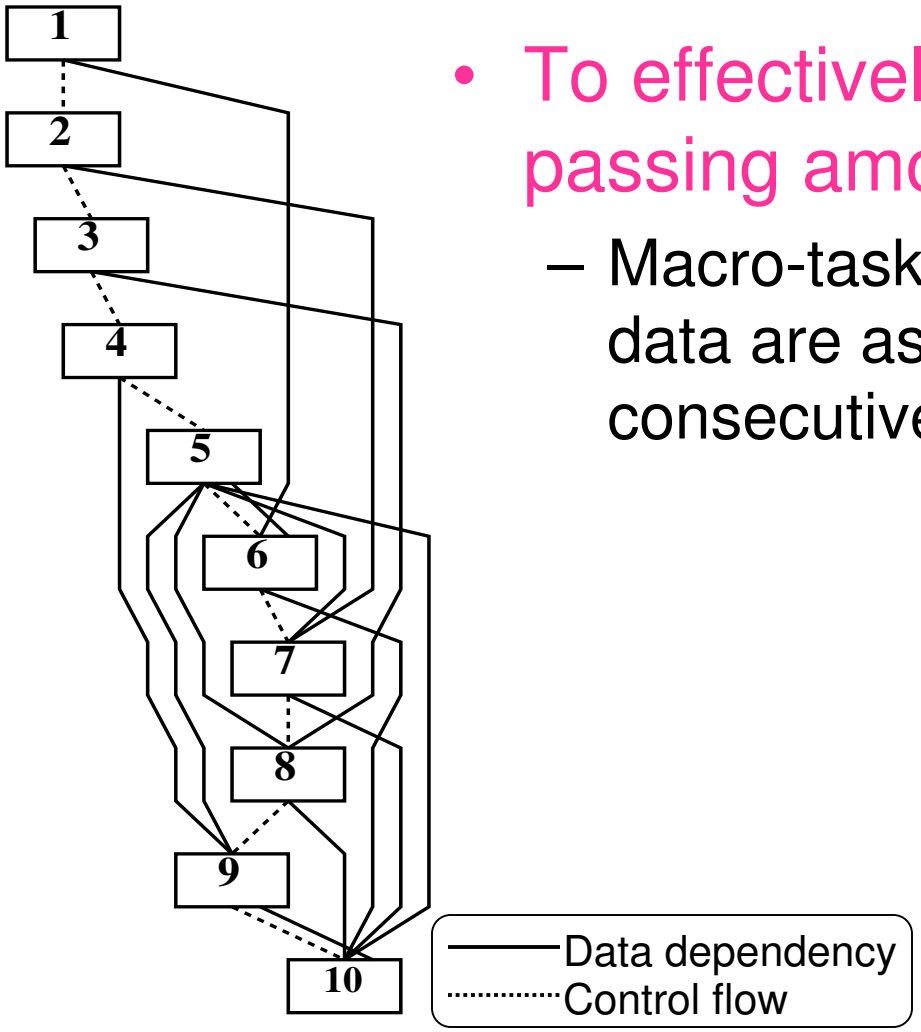
Ex. Earliest Executable Condition of MT6



Macro-Task Graph

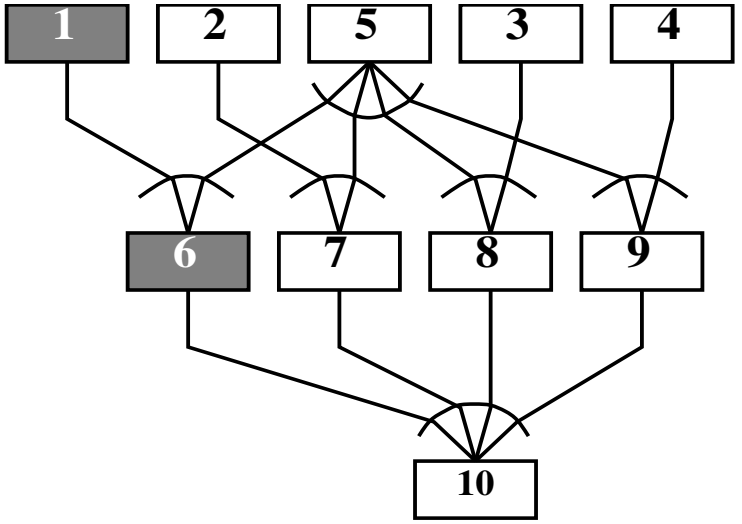
MT2 takes a branch
that guarantees MT4 will be executed
OR
MT3 completes execution

Cache Optimization among Macro-Tasks



Macro Flow Graph

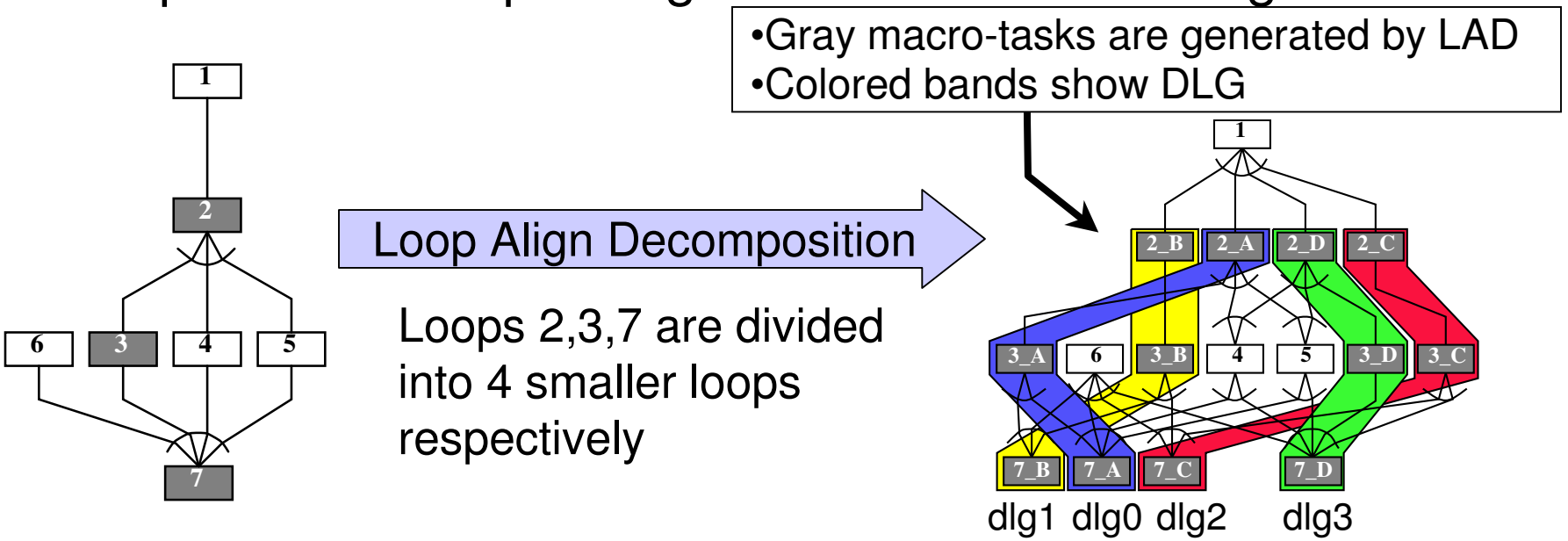
- To effectively use cache for data passing among macro-tasks
  - Macro-tasks accessing the same shared data are assigned to the same processor consecutively



Macro Task Graph

Loop Align Decomposition (LAD)

- If loops access larger data than cache size
  - Divide the loops to smaller loops considering data dependencies among loops
  - Define Data Localizable Group (DLG)
    - DLG: group of macro-tasks to be assigned to the same processor for passing the shared data through cache

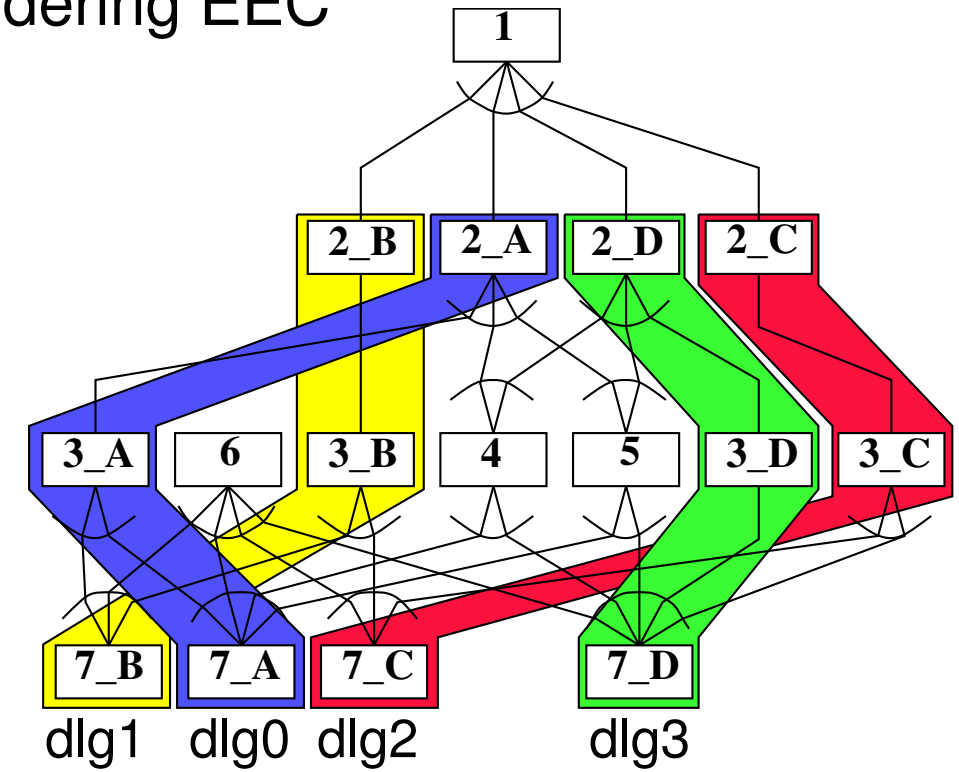


(a) Before loop decomposition

(b) after loop decomposition

Consecutive Execution

Macro-tasks in a DLG are assigned to the same processor as consecutively as possible considering EEC



time

1
2_A
2_B
2_C
2_D
3_A
3_B
3_C
3_D
4
5
6
7_A
7_B
7_C
7_D

Original execution order on single processor

time

1
6
2_A
3_A
2_B
3_B
7_B
2_C
3_C
7_C
2_D
3_D
4
5
7_A
7_D

Scheduling result on single processor

Effective cache usage

Data Layout on a Cache for a Part of SPEC95 Swim

loop1

```
      DO 200 J=1,N
      DO 200 I=1,M
         UNEW(I+1,J) = UOLD(I+1,J)+
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2      +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
         VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1      *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2      -TDTSDY*(H(I,J+1)-H(I,J))
         PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
     1      -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
```

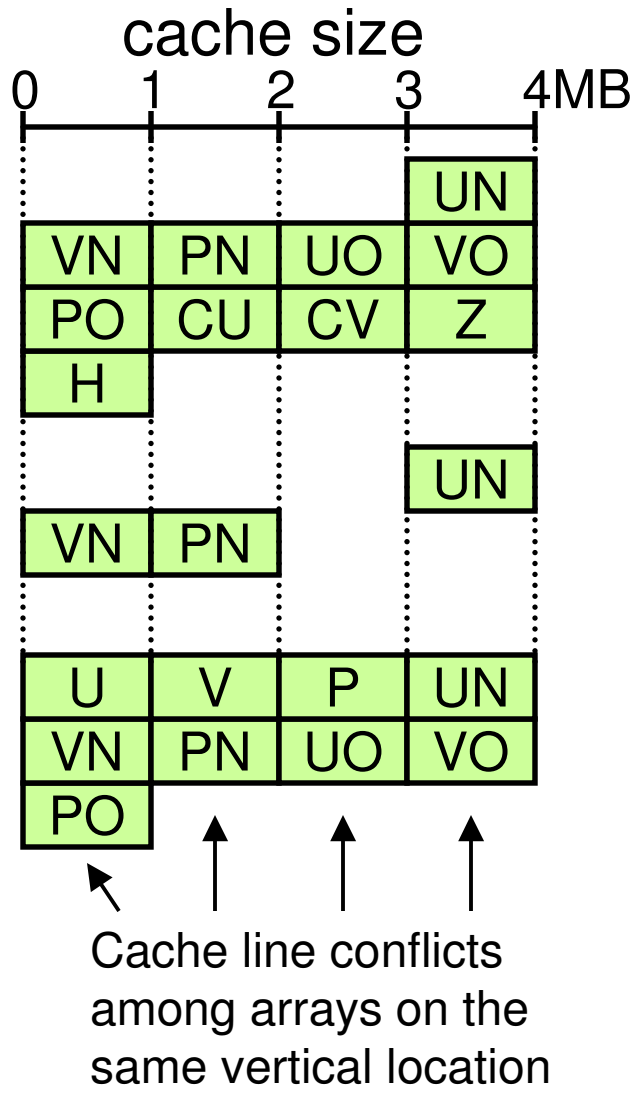
loop2

```
      DO 210 J=1,N
         UNEW(1,J) = UNEW(M+1,J)
         VNEW(M+1,J+1) = VNEW(1,J+1)
         PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
```

loop3

```
      DO 300 J=1,N
      DO 300 I=1,M
         UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
         VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
         POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
  300  CONTINUE
```

(a) An example of target loops for data localization



(b) Image of alignment of arrays on cache accessed by target loops

Partial Arrays Accessed inside a DLG0 by Data Localization

loop1

```
      DO 200 J=1,N
      DO 200 I=1,M
         UNEW(I+1,J) = UOLD(I+1,J)+
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2      +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
         VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1      *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2      -TDTSDY*(H(I,J+1)-H(I,J))
         PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
     1      -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
```

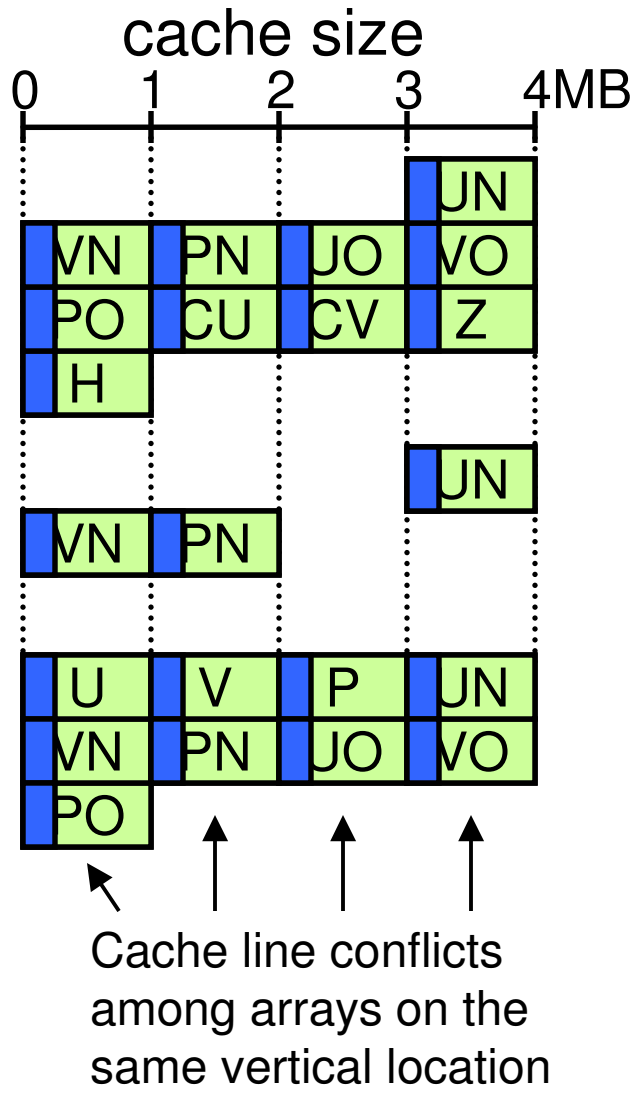
loop2

```
      DO 210 J=1,N
         UNEW(1,J) = UNEW(M+1,J)
         VNEW(M+1,J+1) = VNEW(1,J+1)
         PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
```

loop3

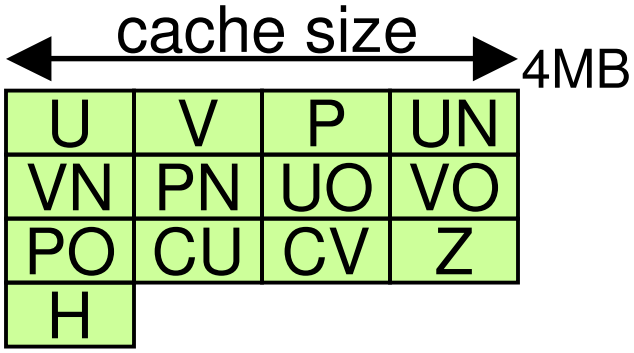
```
      DO 300 J=1,N
      DO 300 I=1,M
         UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
         VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
         POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
  300  CONTINUE
```

(a) An example of target loops for data localization



(b) Image of alignment of arrays on cache accessed by target loops

Padding to Remove Conflict Miss



Example:

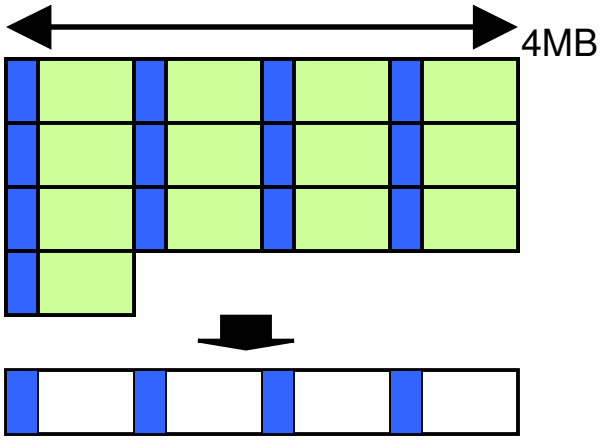
spec95 swim: 1MB x 13 Arrays

Cache: 4MB direct map

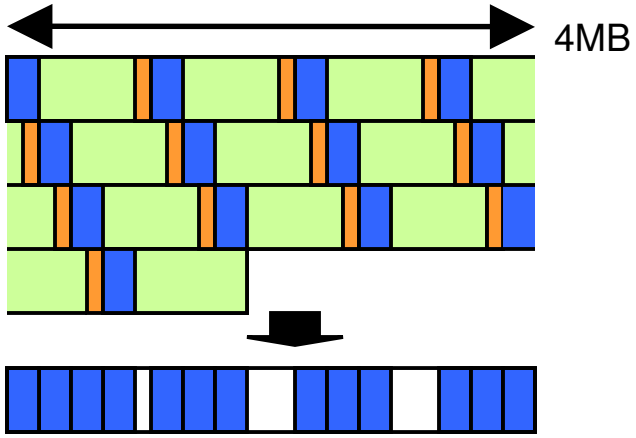
: accessed inside a DLG0

: padding

Loop division



padding by increasing array size



partial arrays are allocated to the limited part of cache

arrays are allocated to the whole cache

many conflict misses

Page Replacement Policy

- Operating System maps virtual address to physical address
  - Compiler works on virtual address
  - Cache uses physical address (ex. L2 of Ultra SPARC II)

    Cache performance and efficiency of compiler data layout transformation depend on OS policy
- Sequential addresses on virtual address are
  - Hashed VA of Solaris8, Page Coloring of AIX4.3
    - sequential on physical address
  - Bin Hopping
    - not sequential on physical address

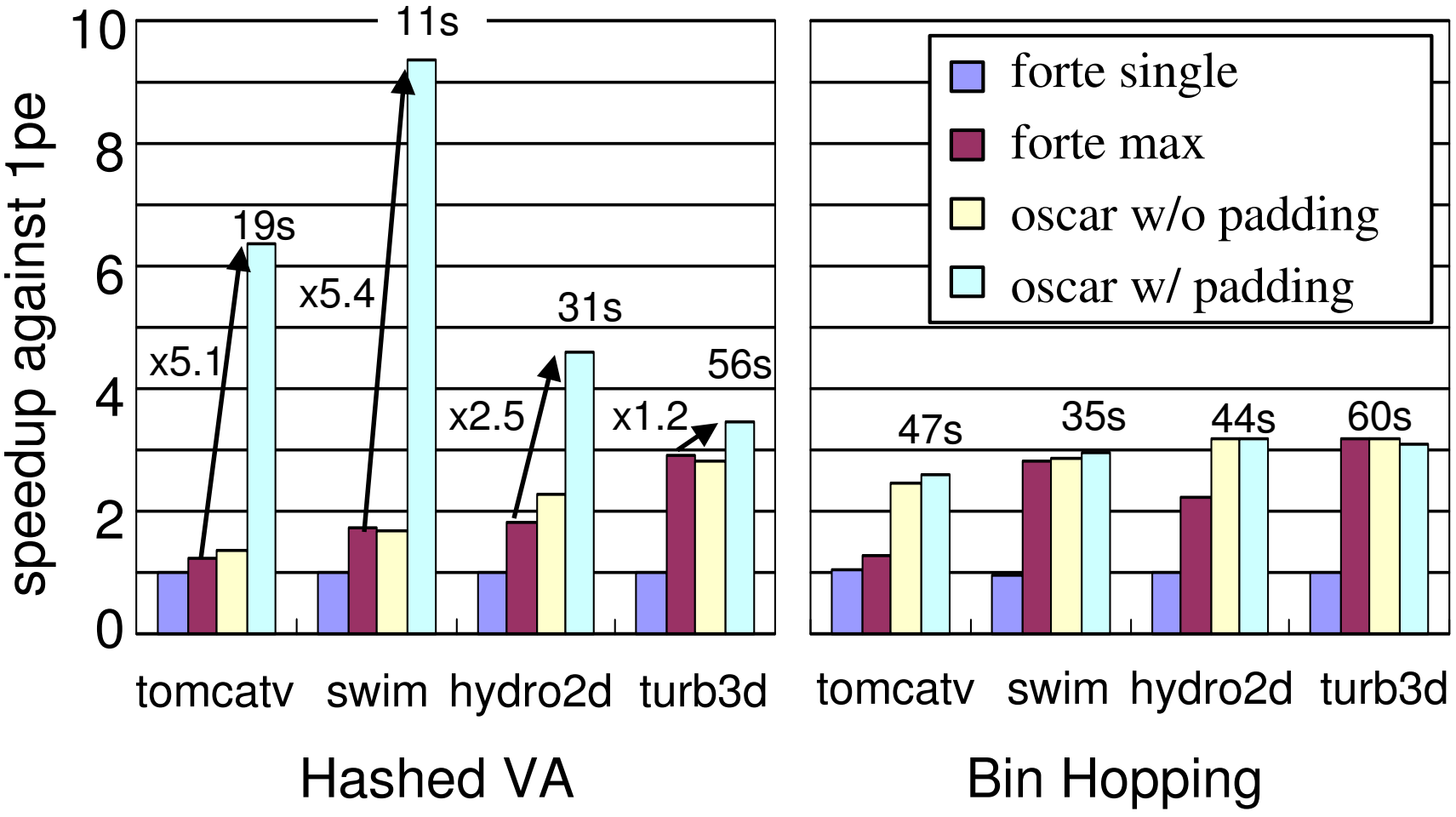
SMP Machines for Performance Evaluation

- Sun Ultra 80
  - Four 450MHz Ultra SPARC IIs
  - 4MB direct map L2 cache
  - Solaris8 (Hashed VA and Bin Hopping)
  - Sun Forte 6 update 2
- IBM RS/6000 44p-270
  - Four 375MHz Power3s
  - 4MB 4-way set associative L2 cache(LRU)
  - AIX 4.3 (Page Coloring and Bin Hopping)
  - XL FORTRAN compiler 7.1

(underline: default page replacement policy)

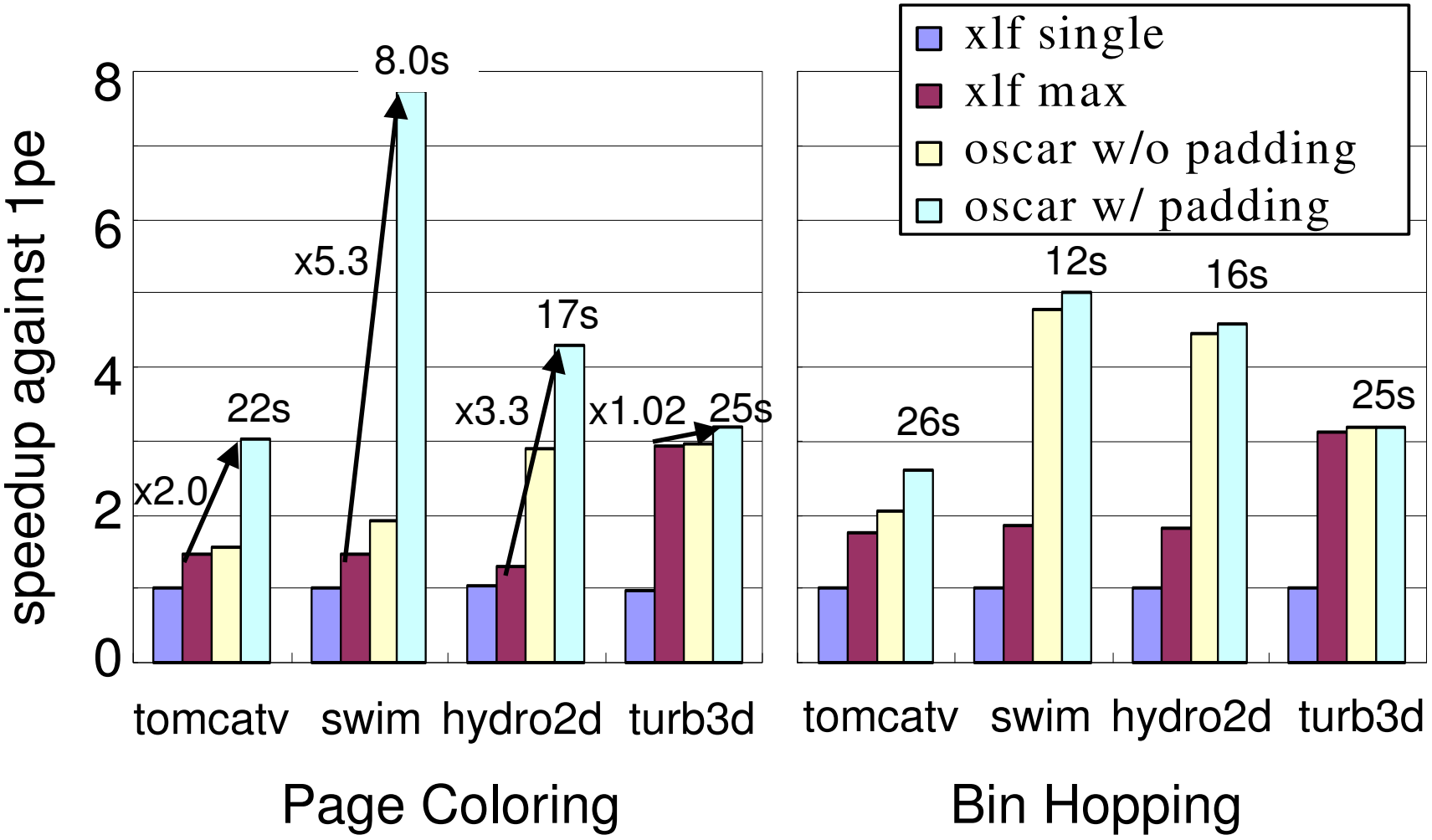
Performance of Data Localization with Inter-Array Padding on Ultra 80

4pe, L2 4MB, direct map



Performance of Data Localization with Inter-Array Padding on RS/6000 44p-270

4pe, L2 4MB, 4-way set associative(LRU)

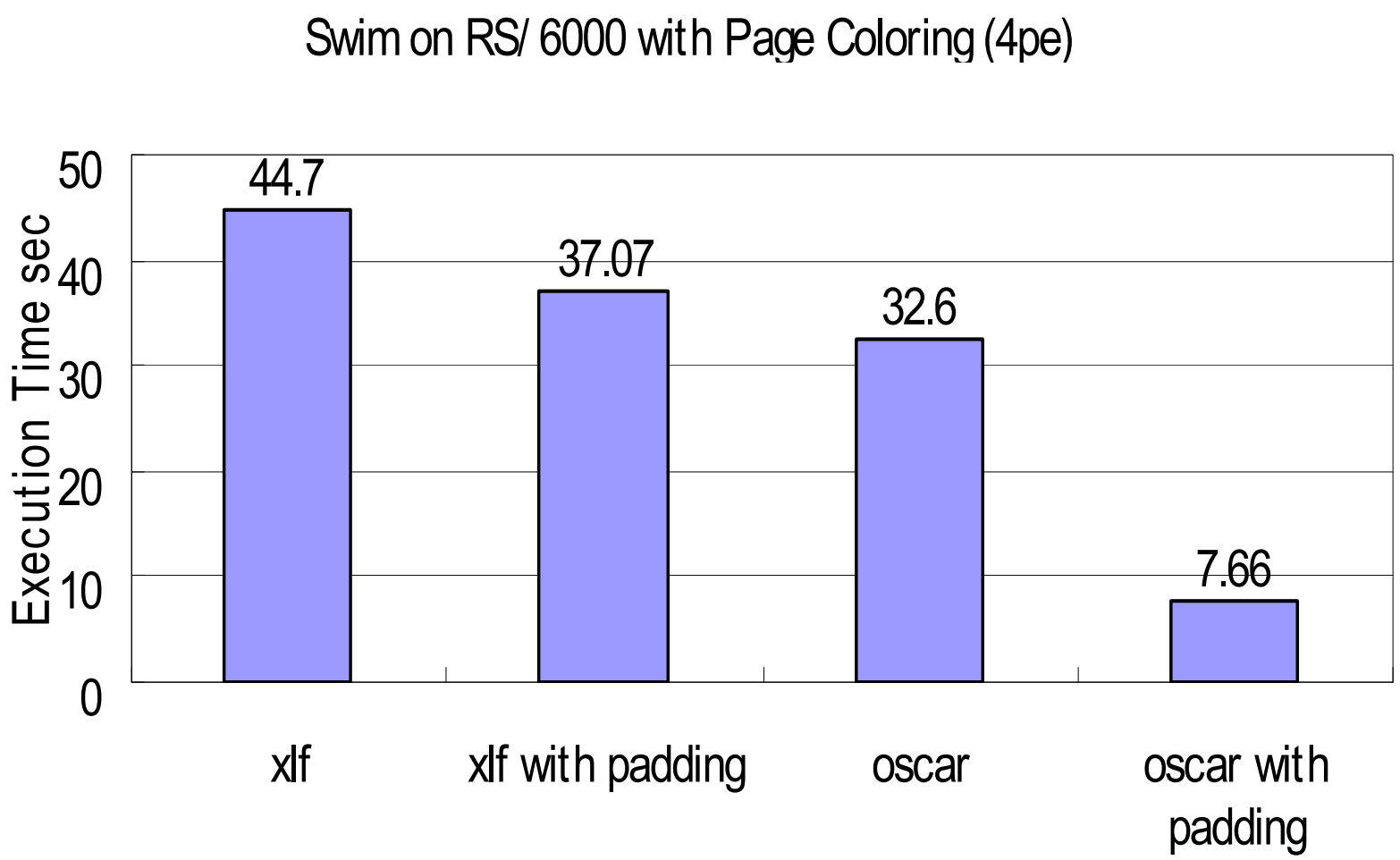


Related Works

- Loop Fusion and Cache Partitioning
  - N.Manjikian and T.Abdelrahman, Univ. of Toronto
- Padding heuristics
  - G.Rivera and C.-W.Tseng, Univ. of Maryland
- Compiler-directed page coloring
  - E.Bugnion and M.Lam et al, Stanford Univ.

Conclusions

- Cache optimization among coarse grain task using inter-array padding with Data Localization
  - Improve data locality over loops by Data Localization composed of *Loop Division* and *Consecutive Execution*
  - Decrease cache misses by *Inter-Array Padding*
- In the evaluation using SPEC95 tomcatv, swim, hydro2d and turb3d
  - on Sun Ultra 80 4processors workstation
    - 5.1 times speedup for tomcatv, 5.4 times speedup for swim compared with Sun Forte 6u2 on Hashed VA
  - on IBM RS/6000 4processors workstation
    - 5.3 times speedup for swim, 3.3 times speedup for hydro2d compared with IBM XLF 7.1 on page coloring


Swim on RS/ 6000 with Page Coloring (4pe)
Execution Time sec
50
40
30
20
10
0
44.7
37.07
32.6
7.66
xlf
xlf with padding
oscar
oscar with padding