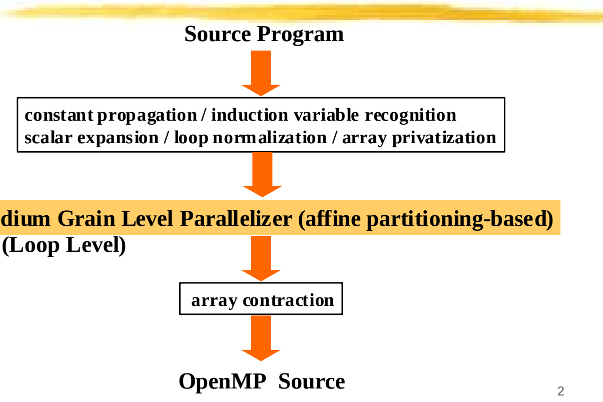
③ Extracting Loop Level Pipeline Parallelism and its Evaluation

Akira HOSOI[1,2], Masaki ARAI[1,2], Toshihiro OZAWA[1,2]
([1] APC Technology Group [2] Fujitsu Limited)

Extracting Loop Level Pipeline Parallelism and its Evaluation

1. The structure of the medium grain parallelizer
2. Affine Partitioning
   2.1. How to extract pipeline parallelism and its problem
   2.2. Refined Algorithm
3. How to implement pipeline parallelism in OpenMP
4. Evaluation

Structure of Medium Grain Level Parallelizer



Source Program

constant propagation / induction variable recognition
scalar expansion / loop normalization / array privatization

Medium Grain Level Parallelizer (affine partitioning-based)
(Loop Level)

array contraction

OpenMP Source

Affine Partitioning [Lim & Lam97]

- The followings can be done at the same time
  - parallelization
  - improve data locality
  - reduce synchronization overhead

- A lot of transformations can be done automatically

- Extract pipeline parallelism

Pipeline Parallelism extracted by Affine Partitioning

Any imperfectly nested loop nests are transformed as follows:

all the assignment statements are surrounded by as many fully permutable loops as possible

m loops

- *m*-1 dimensional pipeline parallel execution can be done
- *m* dimensional tiling can be done

Pipeline Parallelism extracted by Affine Partitioning (con't)

Example :

```
do  i = 1, N
   do  j = 1, i-1
       do  k = 1, j-1
           a(i, j) = a(i, j)-a(i, k)*a(j, k)
        enddo
        a(i, j) = a(i, j) / a(j, j)
    enddo
   do  k = 1, i-1
       a(i, i) = a(i, i)-a(i, k)*a(i, k)
   enddo
   a(i, i) = sqrt(a(i, i))
enddo
```

```
do  i = 1, N      }
   do  j = 1, i   }  fully permutable
      do  k = 1, i }
          if (j < i .and. k < j)
             a(i, j) = a(i, j)-a(i, k)*a(j, k)
          if  (j < i .and. k == j)
             a(i, j) = a(i, j) / a(j, j)
          if   (j == i .and. k < i)
            a(i, i) = a(i, i)-a(i, k)*a(i, k)
           if  (j == i .and. k == i)
             a(i, i) = sqrt(a(i, i))
       enddo
   enddo
enddo
```

How to extract pipeline parallelism

1. Construct an inequality system *Ax* >= 0 from array subscripts and loop bounds
2. Solve *Ax* >= 0 in such a way that *rank A* should be as large as possible (*rank A* = the number of fully permutable loops)

Problem of extracting pipeline parallelism

As the number of assignment statements in a loop nest increases a little, the solution space becomes very large

It takes a large amount of memory and compile time to solve the inequality system directly

Refined Algorithm

1. Assume the number of fully permutable loops in the transformed loop nest

the number of the common surrounding loops in the original loop nest <= *rank A* <= the maximum depth of the original loop nest

```
do  i = 1, N
   do  j = 1, i-1
       do  k = 1, j-1
           a(i, j) = a(i, j)-a(i, k)*a(j, k)
        enddo
        a(i, j) = a(i, j) / a(j, j)
    enddo
   do  k = 1, i-1
       a(i, i) = a(i, i)-a(i, k)*a(i, k)
   enddo
   a(i, i) = sqrt(a(i, i))
enddo
```

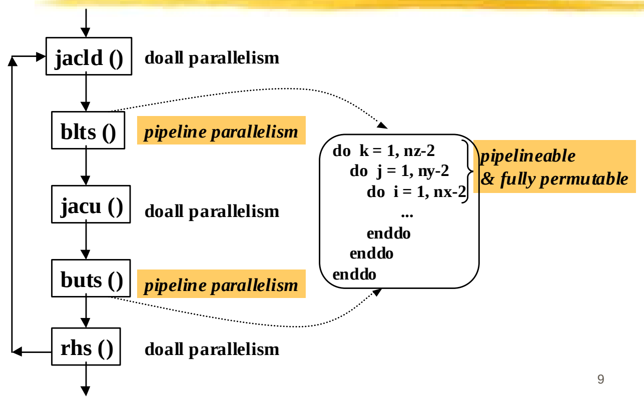
1 <= *rank A* <= 3

Refined Algorithm (con't)

2. Ignore the loop bounds of common surrounding loops to simplify the inequality system

```
do  i = i0, i1   ignore
   do  j = LBj(i), UBj(i)
      A(f(i, j)) = …
   enddo
   do  k = LBk(i), UBk(i)
             … = A(g(i, k))
   enddo
enddo
```

The main loop of SPEC CFP2000 / applu



```
do  k = 1, nz-2
   do  j = 1, ny-2
      do  i = 1, nx-2
           ...
      enddo
   enddo
enddo
```

How to generate pipelined code in openMP

1. Interchange

```
do  k = 1, nz-2      }
   do  j = 1, ny-2   }  fully permutable
      do  i = 1, nx-2 }
          ...
          ...
      enddo
   enddo
enddo
```

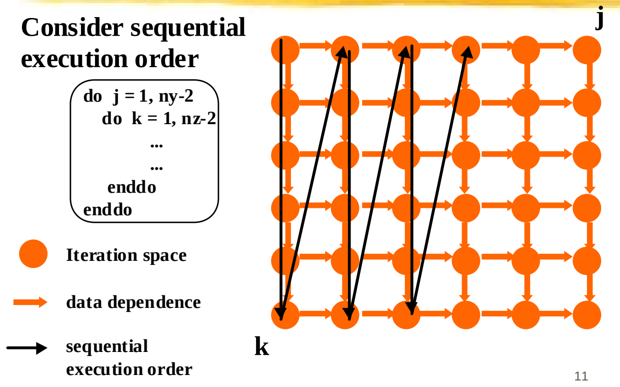
interchange

```
do  j = 1, ny-2
   do  k = 1, nz-2
      do  i = 1, nx-2
          ...
          ...
      enddo
   enddo
enddo
```

How to generate pipelined code in openMP (Cont.)

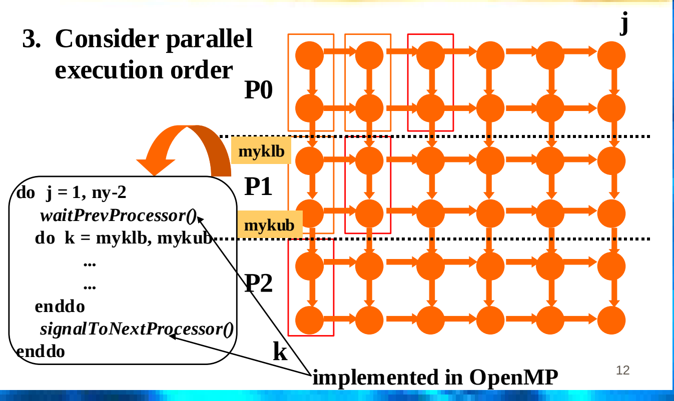
2. Consider sequential execution order

```
do  j = 1, ny-2
   do  k = 1, nz-2
        ...
        ...
   enddo
enddo
```



How to generate pipelined code in openMP (Cont.)

3. Consider parallel execution order



```
do  j = 1, ny-2
    waitPrevProcessor()
   do  k = myklb, mykub
          ...
          ...
   enddo
    signalToNextProcessor()
enddo
```

The Alpha Server

The Alpha Server GS160 Model 6/73

- Alpha 21264 (731MHz) ✕ 8
  (The cc-NUMA machine in which each unit has 4 processors)
- L1-Cache (on-chip)

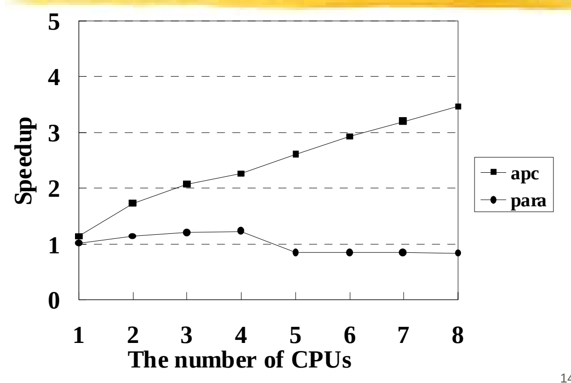
I-Cache	64KB
D-Cache	64KB(2-way)
L2-Cache (direct-map, off-chip)	4MB
- Memory	4GB

The Alpha Digital Fortran Compiler

- compile options:

parallelized code： -v -arch ev6 -O5 –fkapargs=' -conc -ur=1'
sequential code： -v -arch ev6 -O5 –fkapargs='-ur=1'

Speedup of applu on the the Alpha Server



Conclusion

- Pipeline parallelism are automatically extracted from the complicated imperfectly nested loop

- Pipelined code is implemented in OpenMP

- The performance of SPEC CFP2000 / applu can be 2.5 times faster than that on Alpha Server