Medium Grain Level Parallelization and A New Array Contraction Technique

Akira Hosoi and Toshihiro Ozawa

Advanced Parallelizing Compiler Project
Fujitsu LTD.

Overview

1. Extracting Loop Level Pipeline Parallelism
   and its Evaluation

2. Partial Array Contraction
   and Contraction-Oriented Loop Fusion

   Not implemented yet. But the methods are proposed to improve the performance of applu, BT, and SP drastically

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 <u>as many fully permutable loops as possible</u>

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
       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
```

fully permutable

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
```

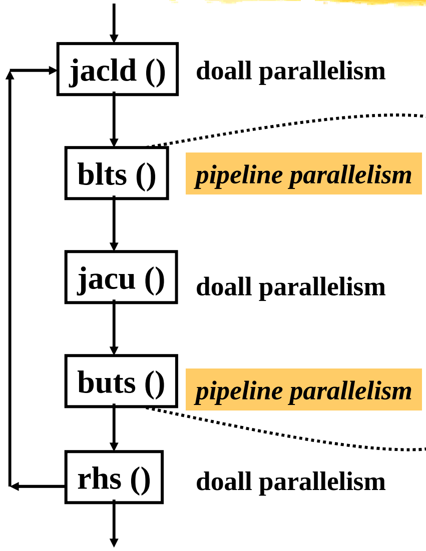
→ 1 <= *rank A* <= 3

Refined Algorithm (con't)

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

do i = i_0, i_1 *ignore*
   do j = $LB_j(i)$, $UB_j(i)$
      A(f(i, j)) = …
   enddo
   do k = $LB_k(i)$, $UB_k(i)$
         … = 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
         v(i,j,k)=f(v(i-1,j,k),v(i,j-1,k),
                      v(i,j,k-1))
      enddo
   enddo
enddo
```

pipelineable & fully permutable

How to generate pipelined code in openMP

1. **Interchange**

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

fully permutable

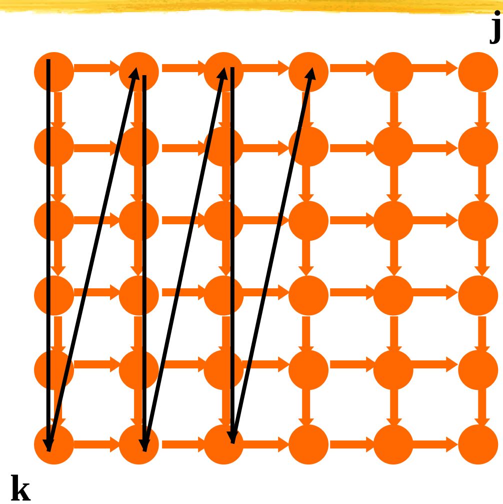
interchange

```
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

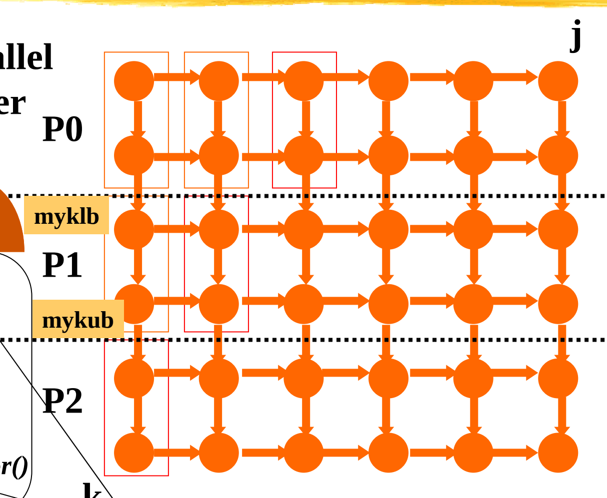
```
do  j = 1, ny-2
    do  k = 1, nz-2
        ...
        ...
    enddo
enddo
```



- Iteration space
- data dependence
- sequential execution order

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
```

implemented in OpenMP

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)

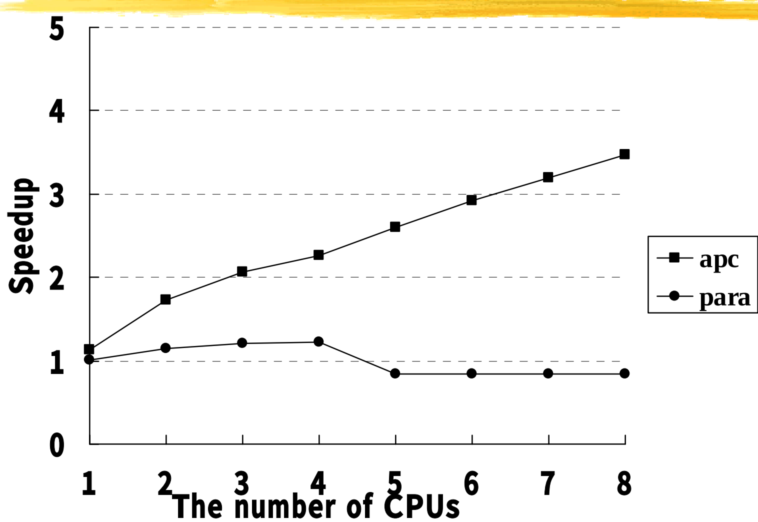
I-Cache	64KB
D-Cache	64KB(2-way)
L2-Cache (direct-map, off-chip)	4MB
- Memory	4GB

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

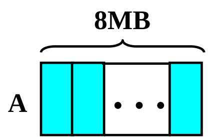
- The performance of SPEC CFP2000 / applu can be 2.5 times faster than that on Alpha Server

Partial Array Contraction and Contraction-Oriented Loop Fusion

1. Introduction
2. Problems of the previous array contraction and its solutions
3. Evaluation
4. Conclusion

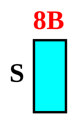
Array Contraction

shrinks the size of an array and improves data locality




```
PARAMETER(N=1000000)
REAL*8   A(N)
DO  I = 1, N
  A(I) = B(I)+C(I)
  D(I) = A(I)*E(I)
ENDDO
```

if A(1:N) are not used out of this loop nest,




```
DO  I = 1, N
  S = B(I)+C(I)
  D(I) = S*E(I)
ENDDO
```

usually combined with *loop fusion*

Problems of the previous array contraction(1)

Not so many opportunities of array contraction

```
DO  I = 1, N
  A(I) = . . .
ENDDO

DO  I = 1, N
  . . . = A(I)
ENDDO
```

fusion

```
DO  I = 1, N
  A(I) = . . .
    . . . = A(I)
ENDDO
```

contraction

```
DO  I = 1, N
  S1 = . . .
  . . . = S1
ENDDO
```

A(1) = . . .

```
DO  I = 2, N
  A(I) = . . .
ENDDO

DO  I = 1, N
  . . . = A(I)
ENDDO
```

fusion

```
DO  I = 1, N
  IF(I>=2) A(I) = . . .
  . . . = A(I)
ENDDO
```

contraction

```
DO  I = 1, N
  IF(I>=2) S2 = . . .
  . . . = S2
ENDDO
```

. . . =A(1)

Partial Array Contraction with Save & Restore

```
DO  I = 1, N
  A(I) = . . .
ENDDO

DO  I = 1, N
  . . . = A(I)
ENDDO
```

fusion

```
DO  I = 1, N
  A(I) = . . .
   . . . = A(I)
ENDDO
```

```
DO  I = 1, N
  S1 = . . .
  . . . = S1
  IF (I == 1) A(I) = S1
ENDDO
```

contracted

Compensation codes are inserted

```
DO  I = 2, N
  A(I) = . . .
ENDDO

DO  I = 1, N
  . . . = A(I)
ENDDO
```

fusion

```
DO  I = 1, N
  IF(I>=2) A(I) = . . .
  . . . = A(I)
ENDDO
```

```
DO  I = 1, N
  IF (I==1) S2 = A(I)
  IF(I>=2) S2 = . . .
  . . . = S2
ENDDO
```

contracted

Problems of the previous array contraction(2)

It is difficult to decide which loop nests should be fused

```
DO  J = 1, N2
  DO  I = 1, N1
    A(I, J) = . . .
  ENDDO
ENDDO
.....................................
DO  J = 1, N2
  DO  I = 1, N1
    . . . = f(A(I, J), A(N1-I+1, J))
  ENDDO
ENDDO
.....................................
DO  J = 1, N2
  DO  I = 1, N1
    A(I, J) = . . .
  ENDDO
ENDDO
.....................................
DO  J = 1, N2
  DO  I = 1, N1
    . . . = g(A(I, J), A(I, N2-J+1))
  ENDDO
ENDDO
```

How we should split a loop nest sequence for better contraction ?

Definition: definition / use-dominated

- An array *A* is *definition-dominated* in a loop nest *L*: if no element is used before defined in *L*.

ex.

```
DO  I = 1, N
  A(I) = . . .
ENDDO
```

```
DO  I = 1, N
  A(I) = . . .
  . . .  =A(I)
ENDDO
```

- An array *A* is *use-dominated* in a loop nest *L*: if no element is defined before used in *L*.

ex.

```
DO  I = 1, N
  . . . = A(I)
ENDDO
```

```
DO  I = 1, N
  . . .  =  A(I)
  A(I) = . . .
ENDDO
```


How to split a loop nest sequence

If *A* is definition-dominated in *L*, *L* becomes the first loop nest of a new group

```
DO  J = 1, N2
  DO  I = 1, N1
    A(I, J) = . . .
  ENDDO
ENDDO

DO  J = 1, N2
  DO  I = 1, N1
     . . . = f(A(I, J), A(N1-I+1, J))
  ENDDO
ENDDO
DO  J = 1, N2
  DO  I = 1, N1
    A(I, J) = . . .
  ENDDO
ENDDO

DO  J = 1, N2
  DO  I = 1, N1
     . . . = g(A(I, J), A(I, N2-J+1))
  ENDDO
ENDDO
```

Benchmark Programs

- SPEC CFP2000/applu

- NPB2.3-serial/SP CLASS A

- NPB2.3-serial/BT CLASS A

How to apply Our Method to SP

1. split

```
      do  100  k = 1, n3-2
       do  100  j = 1, n2-2
        do  100  i = 1, n1-2
100        lhs(i, j, k, 1:15) = . . .

      do  110  k = 1, n3-2
       do  110  j = 1, n2-2
        do  110  i = 1, n1-2
110       lhs(i, j, k, 1:15)=lhs(i, j, k, 1:15) + . . .

      do  120  k = 1, n3-2
       do  120  j = 1, n2-2
        do  120  i = 0, n1-3
120      lhs(i, j, k, 1:15)=f0(lhs(i, j, k, 1:15),
          lhs(i+1, j, k, 1:15), lhs(i+2, j, k, 1:15))
              . . .
       do  130  k = 1, n3-1
        do  130  j = 1, n2-1
         do  130  i = n1-3, 0, -1
130        . . . = g0(lhs(i, j, k, 1:15),
                 lhs(i+1, j, k, 1:15),
                 lhs(i+2, j, k, 1:15))
```

2. renaming
save & restore code are added

```
do  k = 1, n3-2
 do  j = 1, n2-2
   lhs1(0, j, k, 1:15)=lhs(0, j, k, 1:15)
   lhs1(n1-1, j, k, 1:15)=lhs(n1-1, j, k, 1:15)
 enddo
enddo

do  k = 1, n3-2
 do  j = 1, n2-2
   lhs(0, j, k, 1:15)=lhs1(0, j, k, 1:15)
   lhs(n1-1, j, k, 1:15)=lhs1(n1-1, j, k, 1:15)
 enddo
enddo
```

How to apply Our Method to SP (con't)

3. fusion

&

4. partial contraction

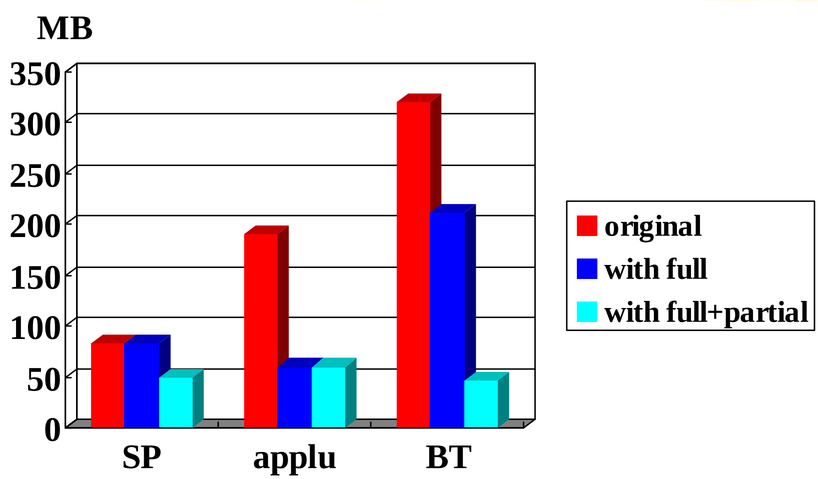
```
do  k = 1, n3-2
  do  j = 1, n2-2
    lhs1(0, 1:15)=lhs(0, j, k, 1:15)
    lhs1(n1-1, 1:15)=lhs(n1-1, j, k, 1:15)
    do  i = 1, n1-2
      lhs1(i, 1:15) = . . .
    enddo
    do  i = 1, n1-2
      lhs1(i, 1:15)=lhs1(i, 1:15) + . . .
    enddo
   do  i = 0, n1-3
      lhs1(i, 1:15)=f0(lhs1(i, 1:15),
       lhs1(i+1, 1:15), lhs1(i+2, 1:15))
   enddo
   do i = n1-3, 0, -1
      . . . = g0(lhs1(i, 1:15),
                 lhs1(i+1, 1:15),
                 lhs1(i+2, 1:15))
   enddo
   lhs(0, j, k, 1:15)=lhs1(0, 1:15)
   lhs(n1-1, j, k, 1:15)=lhs1(n1-1, 1:15)
  enddo
enddo
```

partially contracted

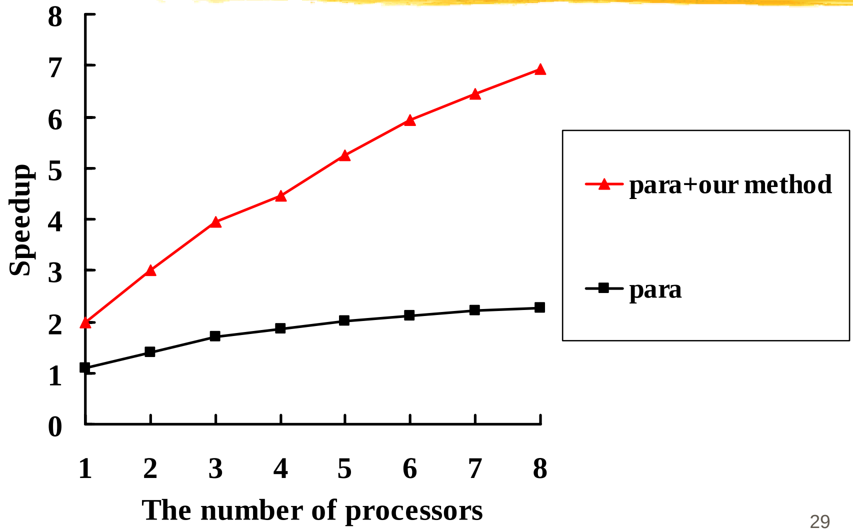
lhs(65, 65, 65, 15) / 33MB

lhs1(65, 15) / 8KB

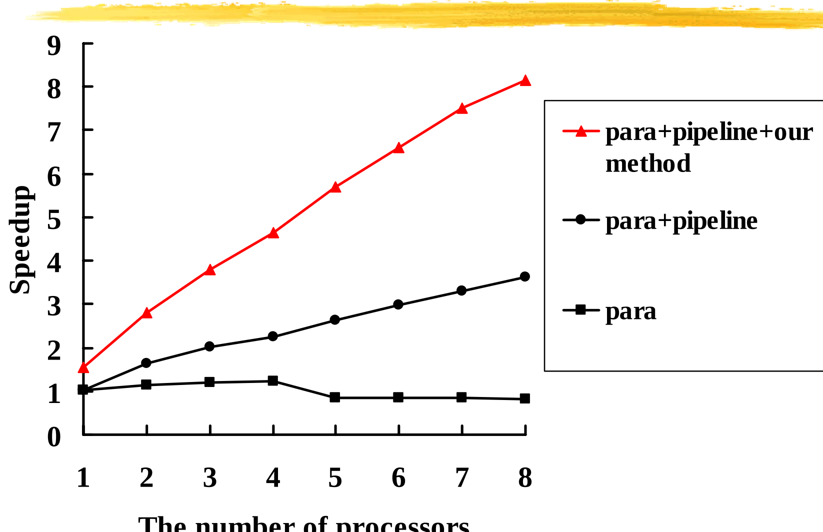
Array data size without/with contraction



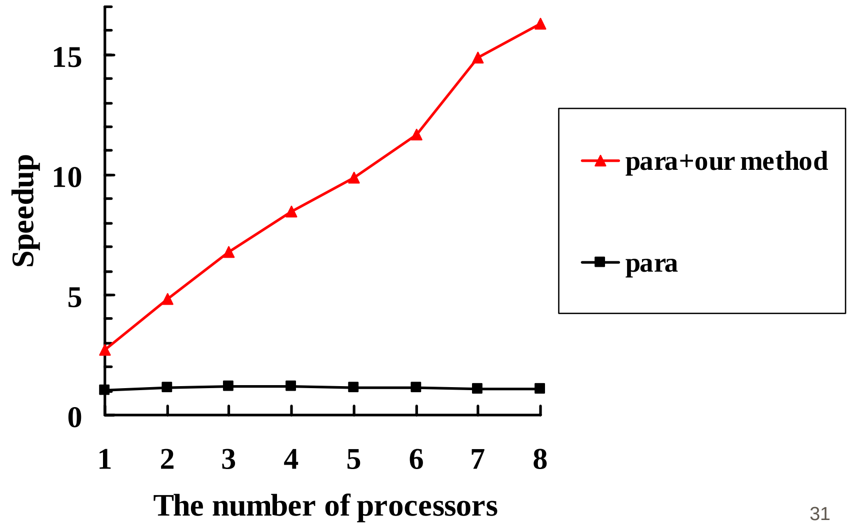
Speedup of SP on the Alpha Server



Speedup of applu on the Alpha Server



Speedup of BT on the Alpha Server



Conclusion

- We generalize array contraction and introduce *partial array contraction with save & restore*
  ➡ We have more opportunities of array contraction
- We split a loop nest sequence using *definition / use-dominated*
  ➡ Array contraction can be applied more effectively
- The performance of SP, applu, and BT are drastically improved