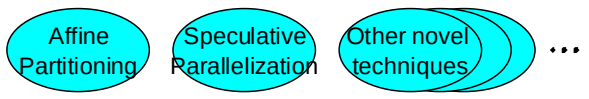
⑥ Interprocedual Techniques for Inducing Parallelization

Hidetoshi Iwashita[1,2], Eiji Yamanaka[1,2], Kiyofumi Suzuki[1,2], Masanori Kaneko[1,2], and Kohichiro Hotta[1,2]
([1] APC Technology Group [2]Fujitsu Limited)

Introduction

- Advanced Parallelizing Compiler (APC) is supported by a number of parallelization techniques:

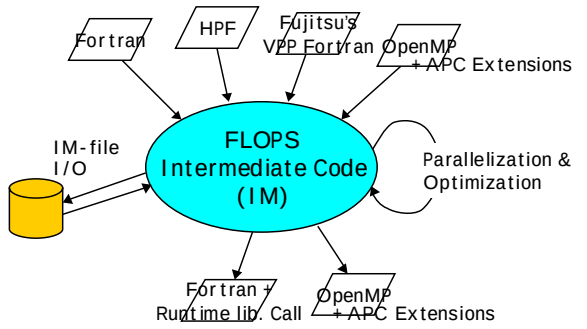


- In order to *induce* those techniques, fine dependency analysis and fundamental code optimization is needed previously.
- The key is: Interprocedural.

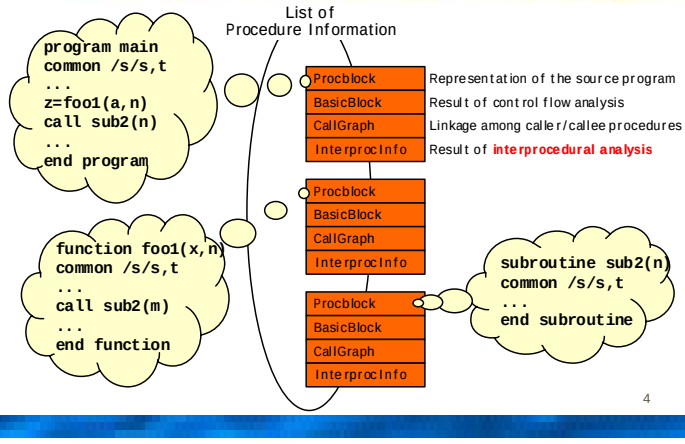
Contents

- Introduction of the compiler framework
- Interprocedural Techniques
  - Interprocedural data flow analysis
  - Interprocedural fundamental optimizations
    (1) Constant propagation & folding
    (2) Induction variable recognition
    (3) Scalar expansion
  - Inline expansion using interprocedural analysis
  - Compiler driver managing interprocedural information
- Summary

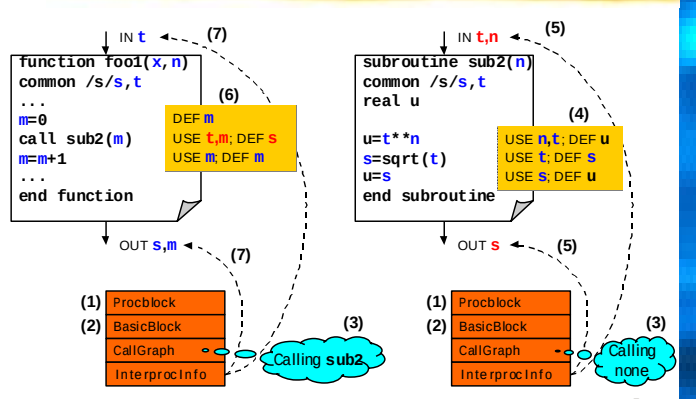
FLOPS Compiler Framework



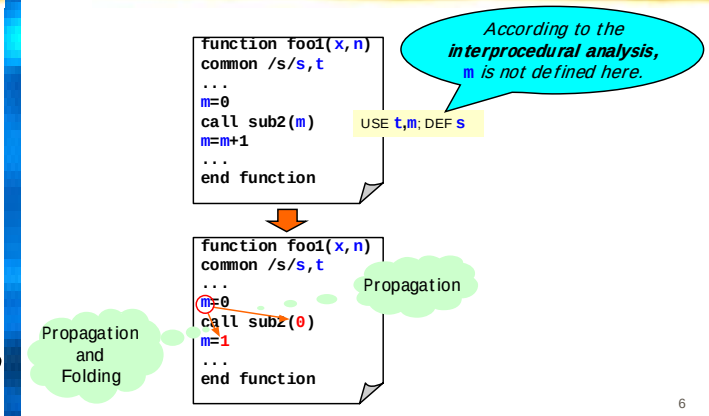
Interface between Compiler Modules



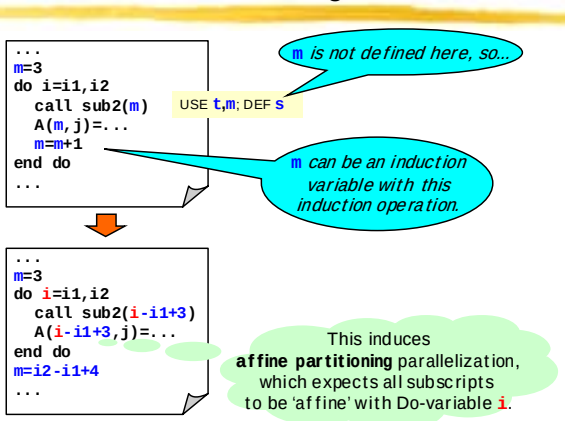
Interprocedural Data Flow Analysis



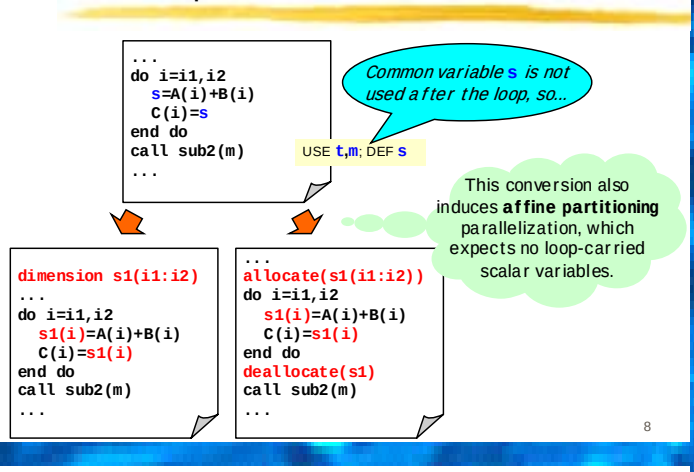
Interprocedural Optimizations (1) Constant Propagation & Folding



Interprocedural Optimizations (2) Induction Variable Recognition



Interprocedural Optimizations (3) Scalar Expansion



Inline Expansion using Interprocedural Analysis

Feature 1

- Supported by the interprocedural analysis and fundamental optimizations.
- Example: SPEC2000 applu

Caller

```
parameter(iar = 60)
dimension rsd(5,iar,iar,iar)
call blts(iar,iar,iar,…,rsd,…)
```

shape: [5,60,60,60]

Subroutine

```
subroutine blts(ldmx,ldmy,ldmz,…,v,…)
dimension v(5,ldmx,ldmy,*)
  v(x,y,z,u)
```

shape: [5,60,60,*]

→ rsd(x,y,z,u)

Because the actual and dummy arguments match in shape, ...

Identical conversion is allowed.

Inline Expansion using Interprocedural Analysis (cntd.)

Feature 2

- Leaves an array argument multi-dimensional.
  - Even if the actual and dummy arguments have different shapes.
  - Collapsing the multi-dimensional array makes the successive optimization difficult in the source-to-source compiler.
- Example: NPB3.0 BT

Caller

```
call binvcrhs(lhs(1,1,bb,0),lhs(1,1,cc,0),rhs(1,0,j,k))
```

Subprogram — Expansion code

```
subroutine binvcrhs (lhs,c,r)
dimension lhs(5,5),c(5,5),r(5)
```

lhs(x,y) → lhs(x,y,bb,0)
c(x,y) → lhs(x,y,cc,0)
r(x) → rhs(x,0,j,k)

Compiler Driver

- Requirement
  - Ease of reconfiguration of the compiler
  - Managing two types of functional modules, which handle each procedure one by one and all procedures together
  - Easy and fine control of every functional module
- Our solution, FLOPS compiler framework:
  - Reconfigurable without recompilation of the compiler itself
  - Programable behavior of the compiler with a script language

An Example of Driver Script

```
parse                 # input a source file including Fortran procedures
repeat                # For each procedure:
 pass mbb             #   making basic blocks (control flow analysis)
end repeat
pass ip_ana           # interprocedural data dependency analysis
pass inline           # selective inline expansion
repeat                # For each procedure:
  pass c_prop         #   constant propagation/folding
  pass i_var          #   induction variable elimination
  pass s_exp          #   scalar expansion
end repeat
echo ***AFFINE STARTS!***      # message output
repeat                # For each procedure:
  pass affine         #   affine partitioning parallelization
end repeat
codegen -k7s          # Fortran code generation with options specifying
end                   # F77 style and fixed format
```


An Illustration of the Execution of FLOPS Compiler Module

Source Program

```
% cat test1.f
      program main
      real a(100)
      a=1.0
      n=2
      call sub1(a,n)
      write(*,*) a,n
      end
      subroutine sub1(x,n)
      real x(100)
      integer n
      do i=1,100
        x(i)=n*i
      enddo
      return
      end
%
```

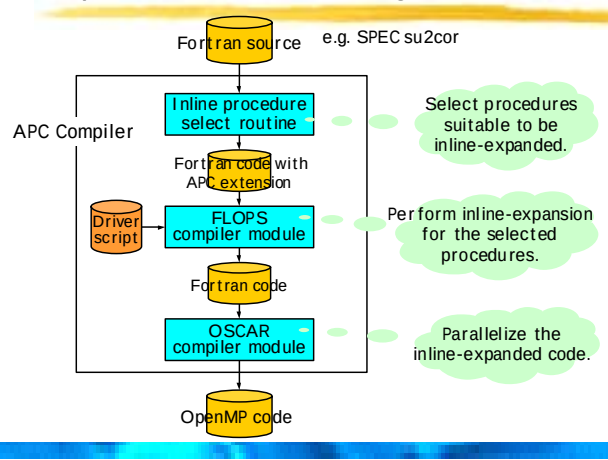
Driver Script

```
% cat script1
parse
pass inline sub1
codegen -k7s
end
%
```

Output

```
% apc_d <script1 test1.f
      PROGRAM main
      REAL a(1:100)
      INTEGER*4 n
      EXTERNAL sub1
      INTEGER*4 iX0
      a = 1.0e0
      n = 2
      DO iX0=1,100,1
        a(iX0) = n*iX0
      ENDDO
      CONTINUE
      WRITE (UNIT=*, FMT=*) a,n
      END
      SUBROUTINE sub1(x,n)
      REAL x(1:100)
      INTEGER*4 n
      INTEGER*4 i
      DO i=1,100,1
        x(i) = n*i
      ENDDO
      RETURN
      END
%
```


An Example of the Combination of Compiler Modules Integrated in APC



Summary

- Intermediate expression was extended for handling the result of interprocedural analysis.
- Fundamental optimizations induce parallelization techniques such as Affine Partitioning.
- Inline expansion, supported by the fundamental optimizations, supports the successive optimizations.
- Compiler driver is reconfigurable and programable with the driver script.