Fortranの
メタプログラミングツール
Metax

理研 R-CCS
村井 均

Introduction (1) — Background & Motivation

- Program tuning is needed for diverse HPC environments.
- Compilers have a limited capability of automatic optimization.
- Users have to perform various kinds of complicated optimization of their application programs.

➡ Lower productivity of HPC programs

Introduction (2) — What's "Metaprogramming"?

- *Metaprogram* ≡ Program that processes programs
- *Metaprogramming* ≡ Programming technique using metaprograms
- Benefits:
  - Performance improvement by compile-time evaluation
  - Productivity improvement by abstraction and reuse of boilerplate codes
  - Code transformation excluded from user's programming work

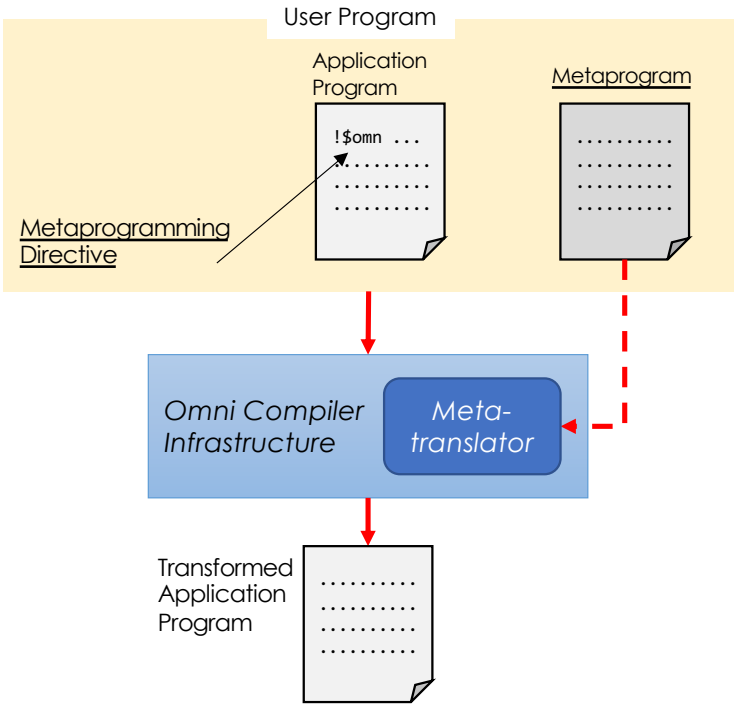
➡ Goal: Developing a new metaprogramming framework based on the *Omni* compiler infrastructure.

Proposed Framework (1) — Approach

- Directive-based
  - Existing HPC languages including Fortran can be supported.
  - It has been widely accepted in HPC.
  - The amount of modifications to existing code can be smaller.
- AST-based
  - Any kind of source-level transformations can be supported.

Proposed Framework (2) — Basic Design

- User Program
  = Application Program + *Metaprogram*
- *Metaprogramming Directive*
  - specifies the kind of transformation to be applied and its target region of code.
- Metaprogram
  - defines the transformation to be applied.
- *Meta-translator*
  - transforms application programs according to metaprograms.



AST-based であることのメリット

- たとえば以下のような場合に、あるコンパイラはインライン展開を適用できない。
  - 仮引数が OPTIONAL 属性をもつ。
  - 仮引数または実引数が形状引継ぎ配列である。
- AST (ソース) レベルの変形を行う Metax では可能。

```
!$omn inline (sub)
call sub(a)
```

```
subroutine sub(a, b)
optional b
a = 100
if (present(b)) b = 200
...
```

```
a = 100
if (.false.) b =200
...
```

```
real a(10,10)
!$omn inline (sub)
call sub(a(6:10,7))
```

```
subroutine sub(x)
real x(:)
x(3) = ...
...
```

```
a(8,7) = ...
...
```

コンパイラ基盤を用いるメリット

- C++ 等のテンプレートメタプログラミングと異なり、Metax はコンパイラの解析情報を活用できる。
- たとえば、変数/指示文の有効範囲を認識できるので、以下のような「メタプログラミング宣言指示文」を実現できる。

```
module mmm
 real x(10,10)
!$omd linearize (x)
...
```

モジュール mmm 内で、変数 x に対してメタプログラミング宣言指示文 linearize を指定。

```
subroutine s1
use mmm

x(3,3) =
```

```
subroutine s2
use mmm

... = x(5,1)
```

モジュール mmm を参照するプログラム単位のすべての x の出現に対し、変換を適用できる。

Case Study (1) — Loop Unrolling

```
!$omn unroll (5)
do i = 1, n
  a (i) = ...
end do
```

```
DO i = 1 , n , 5
  a ( i ) = ...
  a ( i + 1 ) = ...
  a ( i + 2 ) = ...
  a ( i + 3 ) = ...
  a ( i + 4 ) = ...
END DO
DO i = 1 + 5 * ( ( n - 1 + 1 ) / 5 ) , n , 1
  a ( i ) = ...
END DO
```

- Metaprogramming Directive
- Metaprogram
  - modifies the iteration space,
  - replicates the loop body, and
  - creates the remainder loop.

Case Study (1) — Loop Unrolling

- Metaprogram (`unroll` class)

```
public class unroll implements METAXexec {

  public void run(BlockList body, XobjList clauses, METAXblock metaxBlock){

    Xobject factor = clauses.getArg(0);

    ...

    FdoBlock do_block = (FdoBlock)body.getHead();
    FdoBlock reminder_do_block = do_block.copy();

    //
    // transform the iteration space
    //

    do_block.setStep(factor);

    //
    // replicate the loop body
    //

    ...
```

Case Study (1) — Loop Unrolling

- Metaprogram (`unroll` class) (contd.)

```
    // replicate it "factor" times
    for (int i = 0; i < factor.getInt(); i++){
      ...
      // replace the reference to the loop counter i with i, i+1, i+2, ...
      for (j.init(); !j.end(); j.next()) {
        Xobject x = j.getXobject();
        if (x != null && x.Opcode() == Xcode.VAR &&
            x.getSym().equals(ctl_var)){
          j.setXobject(Xcons.plusOp(x, Xcons.IntConstant(i)));
        }
      }
      newBody.add(Bcons.buildBlock(newObj));
    }

    do_block.setBody(newBody);

    //
    // create the remainder loop
    //

    Xobject reminder_lb = ...
    reminder_do_block.setLowerBound(reminder_lb);
    body.add(reminder_do_block);

  }
}
```

Case Study (2) — Data-Layout Optimization of Derived Types

```
type t0
  integer p
end type t0
type t1
  real x
  type (t0) y
end type t1
type (t1) a(100)
!$omd aos_to_soa (a)
a(5)%y%p = 0.
```

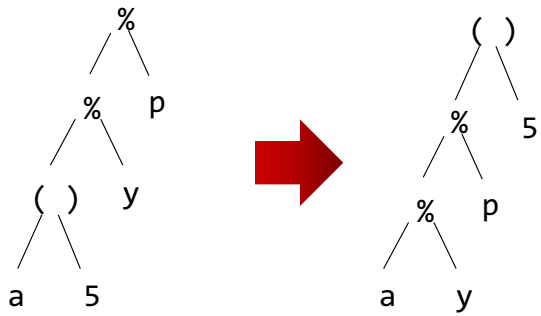
```
TYPE :: t0
  INTEGER :: p ( 1 : 100 )
END TYPE t0

TYPE :: t1
  REAL :: x ( 1 : 100 )
  TYPE ( t0 ) :: y
END TYPE t1

TYPE ( t1 ) :: a

a % y % p ( 5 ) = 0.
```

- Metaprogramming Declarative Directive
- Metaprogram
  - transforms its declaration,
  - transforms its appearance.



AoS Format
a(5)%y%p

SoA Format
a%y%p(5)

Case Study (3) — Inline Expansion for A64FX

- To fully vectorize *Gysela*† with the Fujitsu compiler, inline expansion must be applied manually.
- We used Metax as an *inliner*.

```
!$omn inline ( &
...
!$omn  spline2d_scoef_rblock_inplace, &
...
!$omn )
...
call spline2d_scoef_rblock_inplace(...)
...
!$omn end inline
```

This `inline` directive has the names of subroutines to be inlined recursively.

† Grandgirard, V. et al.: GYSELA, a full-f global gyrokinetic Semi-Lagrangian code for ITG turbulence simulations. AIP Conference Proceedings, 871(1), 100-111. doi:101063/12404543 (Nov 2006).

Case Study (3) — Inline Expansion for A64FX

- Target:
  - subroutine: `bsl_2d_newton_polar_advec_fast`
  - data: `smalltest`
- 12 threads x 4 processes x 2 nodes of Fugaku

	elapsed time	SIMD ratio	SVE ratio
original	4.05	1.41	26.33
(1)	3.56	2.79	35.15
(1)(2)	2.26	3.60	30.63
(1)(2)(3)	2.10	4.67	38.02
(1)(2)(4)	1.30	25.11	80.67
(1)(2)(3)(4)	1.17	29.96	86.02

(1) `-Knoalias=s` compile flag
(2) METAX for inline expansion
(3) Hoisting IF statements out of the loop
(4) Replacement of MODULO intrinsic functions

Summary

- Metax is a metaprogramming framework for existing HPC languages based on Omni.
- Metax allows users to define their own directives for any kind of source-level transformation.
- Three case studies prove its feasibility.
  - loop unrolling
  - data-layout optimization of derived types
  - inline expansion for A64FX
- Future works:
  - more well-defined and easier-to-use API to write metaprograms.
  - more case studies to evaluate its feasibility and effectiveness.