Satoshi Ohshima

Assistant Professor, Supercomputing Research Division, Information Technology Center, The University of Tokyo

Auto-tuning of "Modified/generated code by optimization and auto-tuning" problem

My main research targets

1. Numerical calculation on various parallel processors
   - main target calculations
     - sparse matrix solver (CG method and its variations)
     - sparse matrix-vector multiplication (SpMV)
   - main target hardware
     - manycore processor (Xeon Phi, Knights Corner/Landing)
     - GPU (NVIDIA with CUDA, NVIDIA&AMD with OpenACC)
2. AT & AT language
   - optimization of OpenMP/OpenACC directives
     - OpenMP loop scheduling
     - OpenACC execution form (gang, worker, vector)
   - latest work: expansion of ppOpen-AT for OpenACC

Today's topics

- Latest work
  - ppOpen-AT for OpenACC
- Current work (future work of the latest work)
  - Auto-tuning of "Modified/generated code by optimization and auto-tuning" problem

ppOpen-AT for OpenACC

- This research topic is published at iWAPT2016 (IPDPS workshop).
  - Utilization and Expansion of ppOpen-AT for OpenACC
    Satoshi OHSHIMA, Takahiro KATAGIRI, Masaharu MATSUMOTO
  - In this talk, only the outline is shown.

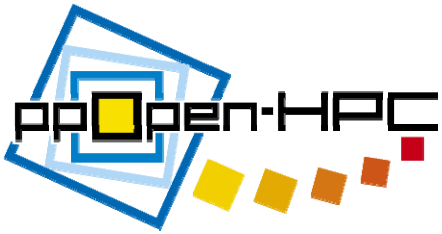
- short summary
  - In order to optimize OpenACC programs, we considered to utilize Auto-tuning techniques.
  - We show that some existing AT features of ppOpen-AT are useful, but ppOpen-AT cannot support some optimization strategies.
  - To support further optimization strategies, we added new specification to ppOpen-AT.
    - Variable Selection in Directive
    - Gang/Worker/Vector Optimization

OpenACC

- New parallel programing language (standard) which supports CPU, MIC, and GPU.
- Users can make parallel programs for CPU, MIC, and GPU by using pragmas (directives).
  - similar to OpenMP for GPU
- Developed by (mainly) NVIDIA and some compiler vendors.
- PGI, Cray, Pathscale compilers support OpenACC 2.0.
  - Probably, most of OpenACC users use PGI compiler.
- Some research compilers and open-source compilers support (a part of) OpenACC, too.
  - omni, OpenUH, OpenARC, …

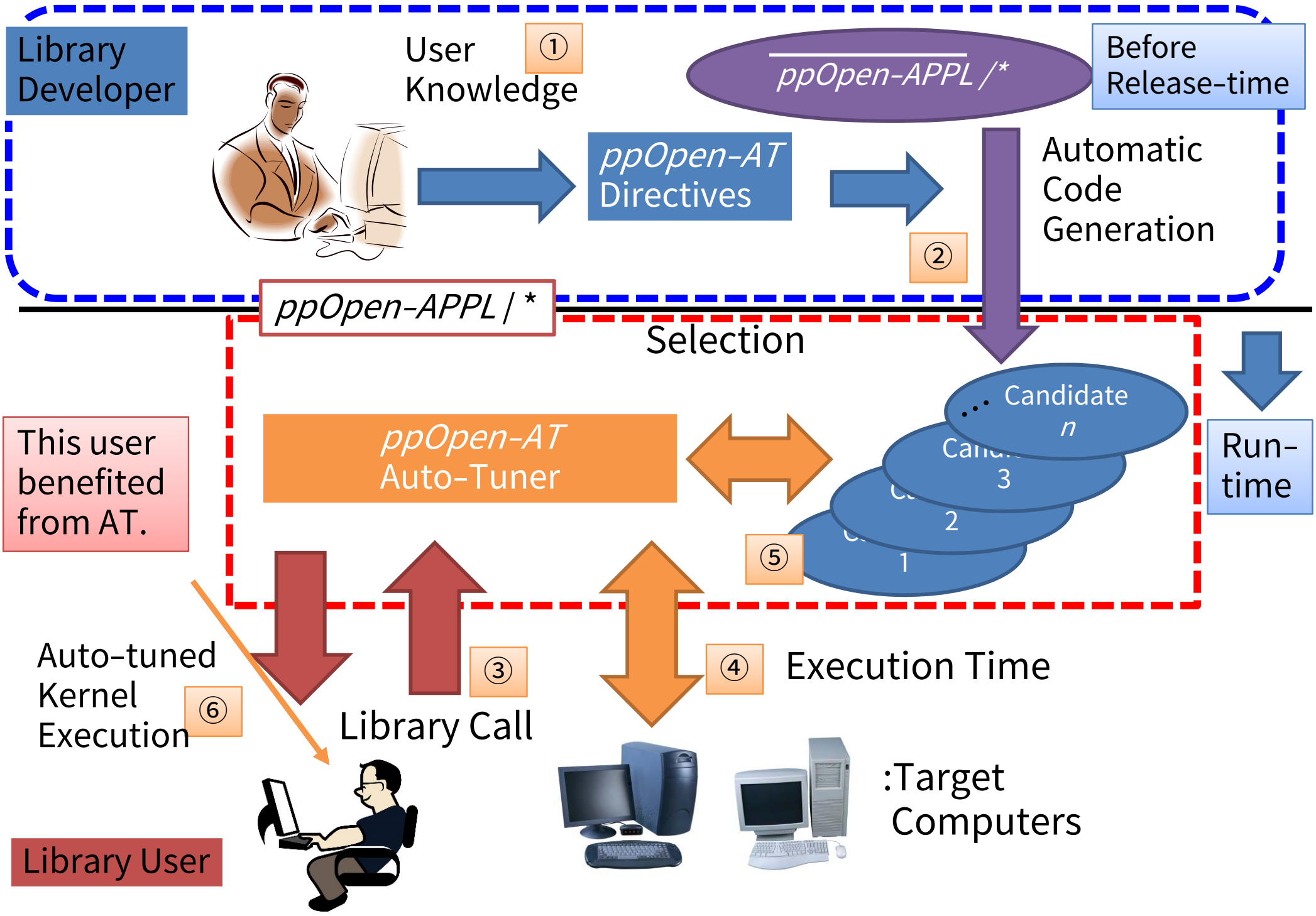
ppOpen-AT

http://ppopenhpc.cc.u-tokyo.ac.jp/



- directive-based AT language
  - part of ppOpen-HPC led by Prof.Nakajima
  - led by Prof.Katagiri
  - provides source-to-source program modification/generation tool and runtime library
  - supports Fortran and C languages
  - generates candidate codes before compile time, and chooses the best one at 1st runtime, then uses only the best code
    - optimal code is chosen based on depending parameter (e.g. problem size)
  - provides parameter estimation features
    - d-Spline (-> previous talk)
  - NOT a compiler, DO NOT generate codes at runtime
    - useful for traditional supercomputers in which software and compilers are limited

ppOpen-AT System



Questions

- Present situation (before this research)
  - ppOpen-AT is used on MPI/OpenMP environment.
  - While OpenMP is directive-based language, ppOpen-AT treats OpenMP directives as comments.

- Questions
  1. Is ppOpen-AT (treats directives as comments) useful for OpenACC optimization?
  2. If ppOpen-AT can modify OpenACC directives, can ppOpen-AT support any other good optimization techniques?

Major AT features of ppOpen-AT

- Loop Unrolling
  - chooses the best unrolling level
- Variable Selection
  - chooses the best value from the given values
- Loop Transformation
  - chooses loop variation from collapse, split, rotation, and their combinations
- Kernel Selection
  - chooses the best sub kernel by given sub kernels

Q. Are these features useful for OpenACC?

Useful in OpenACC ?

- Loop Unrolling
  - chooses the best unrolling
- Variable Selection
  - chooses the best value fro
- Loop Transformation
  - chooses loop variati fo combinations
- Kernel Selection
  - chooses the best sub n

not a major optimization strategy, but reduces the number of branch instructions

very general feature, useful for selecting block size or so

makes nested loops simple, modifies amount of registers, modifies memory access pattern, etc.

very general feature, useful for comparing and choosing target processor or so

- basically, useful -> ppOpen-AT is useful for OpenACC
- However, there are some difficult cases.

Difficult cases (Issues)

- These optimization techniques are difficult for ppOpen-AT
  - optimizing execution form (gang, worker, vector)
  - optimizing collapse level

Example
- `!$acc kernels loop gang(26) vector(256)`
- `!$acc kernels loop collapse(2)`

- To do them, we have to optimize variables in directive.
- Current ppOpen-AT cannot optimize the variables in directive because directives are regarded as comment.
  - directives are processed before compiling
    → cannot know and use the value in execution time
  - NOTE: policy of ppOpen-AT: keep target code with ppOpen-AT directives collect grammatically if ignoring the directives

Proposal: Variable Selection in Directive

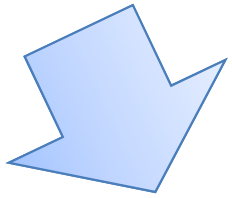
Simple idea: Variable Selection in directive

If OAT directives are ignored, OpenACC compiler (interpreter) causes compile error.

```
integer, parameter :: X = 2
!OAT$ static variableD (X) region start
!OAT$ name MyLoop
!OAT$ variedD (X) from 1 to 2
!$acc kernels loop gang vector collapse (X)
  do i=1, N
    do j=1, N
    ......
    enddo
  enddo
!$acc end kernels
!OAT$ static variableD region end
```

Solution: constant value

- const variables (C) and parameter variables (Fortran) can be used in directive



```
!$acc kernels loop gang vector collapse (1)
!$acc kernels loop gang vector collapse (2)
```

- Using constant value, ppOpen-AT began to support this feature.
  - Level of loop collapse can be AT target.
  - Similarly, execution form can be AT target.

Proposal: Gang/Worker/Vector Optimization

Original code

```
!OAT$ static GWV region start
!OAT$ name GWV-test
!OAT$ GWV-list L (0,0,256) (30,,128) (60,-,-)
!$acc kernels
!OAT$ GWV-target L
!$acc loop gang, worker, vector
  do i=1, N
    ......
  enddo
!$acc end kernels
!OAT$ static GWV region end
```

give a label and candidate list

loop just after the label is a target

- >0 : given value = #parallelism
- 0 : not use the form
- others : default (default = compiler dependent)

While Variable Selection in Directive can optimize execution form, but it is difficult to use complex cases include omission of the forms and numbers.

execution forms generated by above example

```
!$acc loop vector(256)
!$acc loop gang(30) worker vector(128)
!$acc loop gang(60) worker vector
```

Another version of Gang/Worker/Vector Optimization

```
!OAT$ static GWV region start
!OAT$ name GWV-test2
!OAT$ GWV-list [L1,L2] [(0,,),(0,,128)] [(,0,),(,,32)]
!OAT$ GWV-target L1
!$acc kernels loop
  do i=1, N
  ......
  enddo
!$acc end kernels
!OAT$ GWV-target L2
!$acc kernels loop
  do j=1, N
  ......
  enddo
!$acc end kernels
!OAT$ static GWV region end
```

can indicate the *combination*

```
case(1)
!$acc kernels loop worker vector
!$acc kernels loop worker vector(128)

case(2)
!$acc kernels loop gang vector
!$acc kernels loop gang worker vector(32)
```

- can find the best execution form by a little bit directives
- issue: if there are large number of combinations, long search time is needed (appropriate if the chosen value is used many times)

Summary of this research topic

- To optimize OpenACC program, we considered to utilize Auto-tuning techniques.
  - some existing AT features of ppOpen-AT are useful
  - ppOpen-AT cannot support some optimization strategies
- To support further optimization strategies, we added new specifications to ppOpen-AT.
  - Variable Selection in Directive
  - Gang/Worker/Vector Optimization

These features are available in current ppOpen-AT.

- ToDo: to evaluate performance and productivity at various applications
- There are some future work …

Modified/generated code by optimization and auto-tuning problem

- This is an advanced problem which I noticed when I'm developing and evaluating ppOpen-AT for OpenACC.
- In OpenACC…
  - optimizing each loop is important
    - sequential or parallel (loop directive, seq directive)
    - execution form (gang, worker, vector)
  - loop transformation has large effect
  - kernel selection is also important
- Question
  - How to use the combination of them?
  - Example
    - When AT feature modifies the loop structures, new loop structures are generated. I want to optimize the generated loops again.

Example : Loop Transformation & Execution Form

```
!OAT$ static LoopFusionSplit region start
!OAT$ name MMtest
!$acc kernels
!$acc loop
do x=1, m
  do y=1, n
    do z=1, k
      c(x,y) = c(x,y) + a(x,z) * b(z,y)
    enddo
  enddo
enddo
!$acc end kernels
!OAT$ static LoopFusionSplit region end
```

program conversion & generation

```
!!OAT$ static LoopFusionSplit region start
!!OAT$ name MMtest
!$acc kernels
!$acc loop
  do X_y = 1 , m*n
  X = (X_y-1)/n + 1
  y = mod((X_y-1),n) + 1
! do x=1, m
!   do y=1, n
      do z=1, k
         c(x,y) = c(x,y) + a(x,z) * b(z,y)
      enddo
!   enddo
  enddo
!$acc end kernels
return
end
```

(this is one of the candidate codes)

New parallel loop
We want to optimize the execution form of this loop.

Example : Kernel Selection & Loop Transformation

```
!OAT$ static select region start
!OAT$ name Select_Kernels_with_Transform

!OAT$ select sub region start
!$acc kernels copyin(foo) copyout(bar)
!$acc loop
do i=1, N
  ..... ! calculate on GPU
enddo
!$acc end kernels
!OAT$ select sub region end

!OAT$ select sub region start
!$acc kernels copyin(foo) copyout(bar)
!$acc loop
do i=1,N
  ..... ! calculate on GPU
enddo
!$acc end kernels
!OAT$ select sub region end

!OAT$ static select region end
```

sub kernel 1

sub kernel 2

program conversion & generation

```
select case(iusw1)

  case(1)
!$acc kernels copyin(foo) copyout(bar)
!$acc loop
    do i=1, N
      ..... ! calculate on GPU
    enddo
!$acc end kernels

  case(2)
!$acc kernels copyin(foo) copyout(bar)
!$acc loop
    do i=1,N
      ..... ! calculate on GPU
    enddo
!$acc end kernels

end select
```

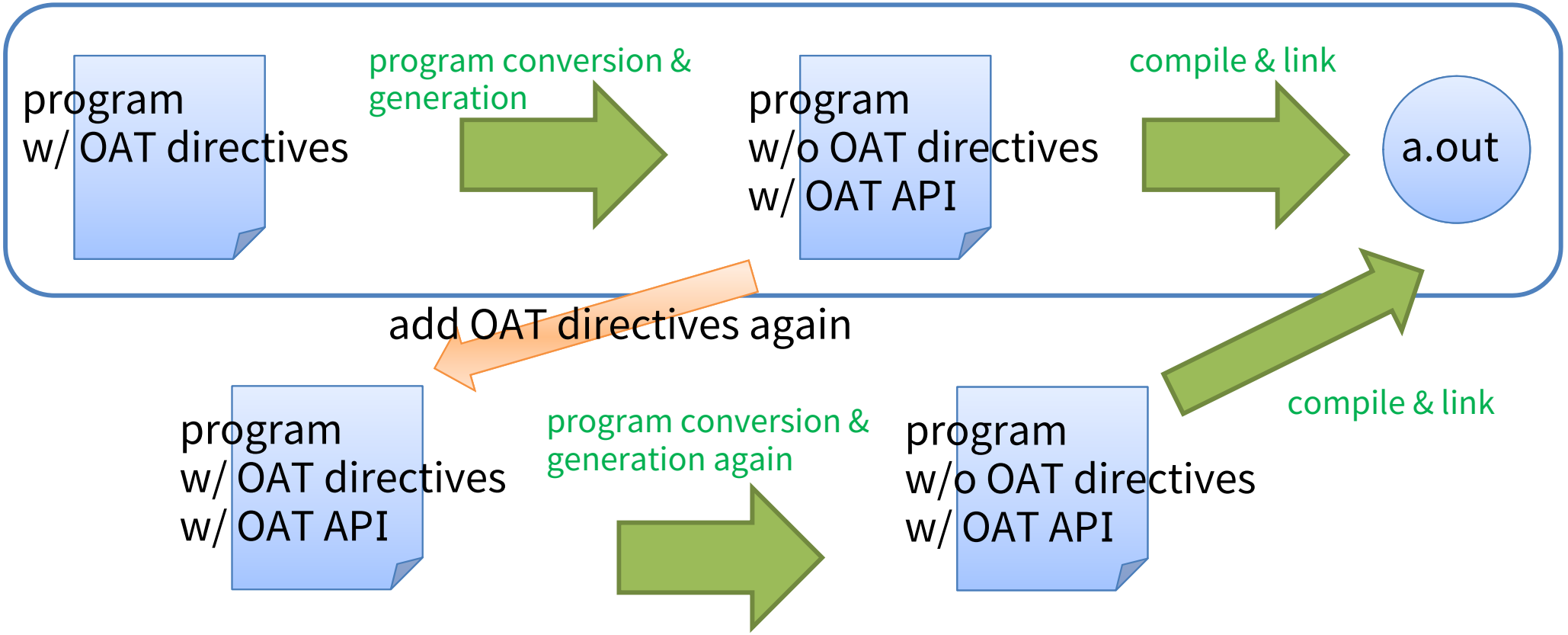
sub kernel 1

sub kernel 2

We want to transform this loop.

Solution?

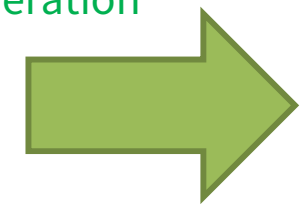
- Because ppOpen-AT writes human-readable code, user can add OAT directives to the AT applied code and transform it again.



Issue 1

```
!OAT$ ~ start
do …
enddo
!OAT$ ~ end
```

program conversion & generation



```
select case ~
case (1)
.....
case (2)
.....
case (3)
.....
case (4)
.....
case (5)
.....
case (6)
.....
case (7)
.....
case (8)
.....
```

add OAT directive again

add OAT directive again

add OAT directive again

add OAT directive again

add OAT directive again

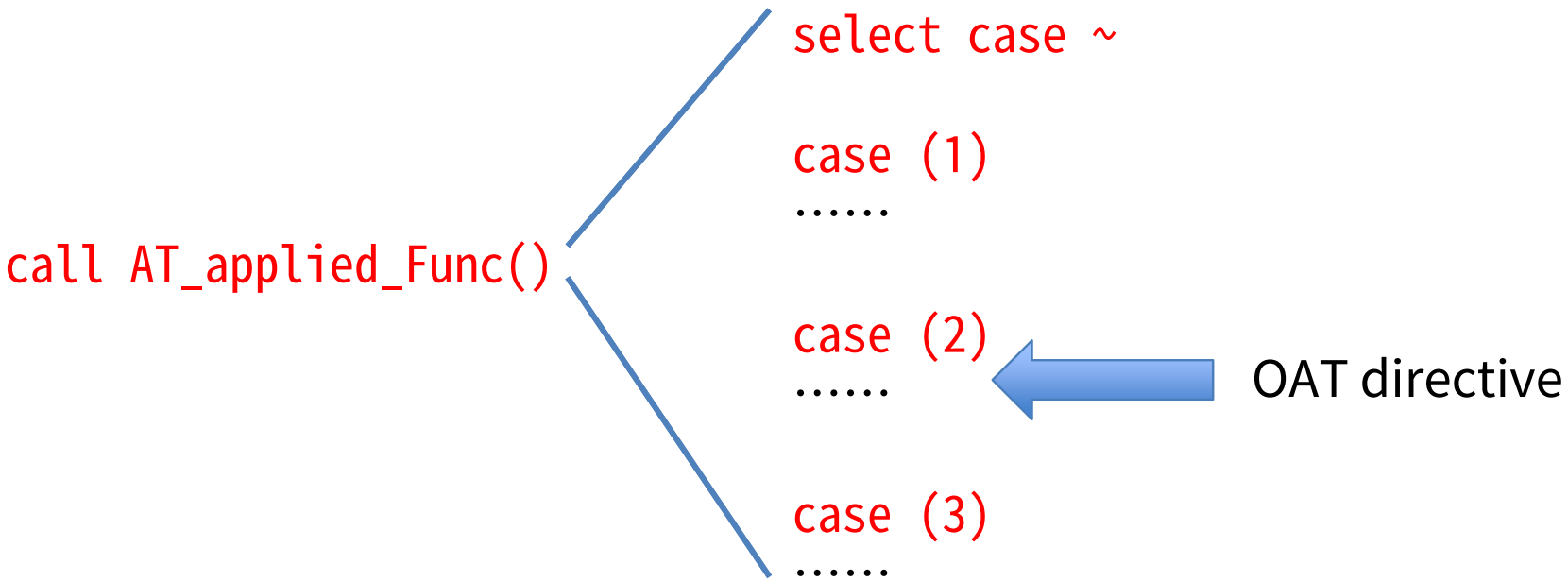
add OAT directive again

add OAT directive again

add OAT directive again

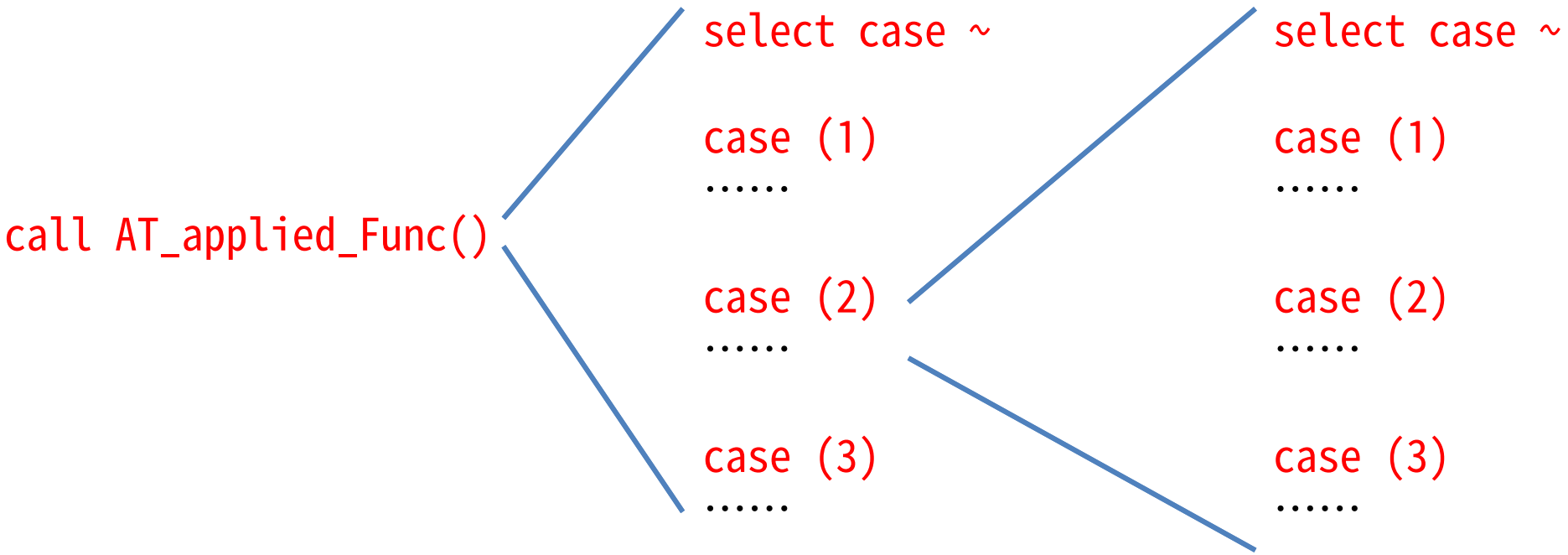
If there are many candidate codes, users will have to add many directives again.
This is not an impossible work, but bother work for users.

Issue 2



Runtime has the performance information of all AT variations and chooses the best one.
If new OAT directive is added to the candidate code, ……

Issue 2



Runtime has the performance information of all AT variations and chooses the best one.
If new OAT directive is added to the candidate code, there are candidate codes in the candidate code.
To choose the best code, we have to design runtime program again.

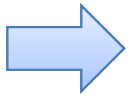
- Because it is difficult to solve above issues completely, I consider very simple case.

- Optimization of execution form (gang, worker, vector) has large impact to the performance of OpenACC program.
- This optimization can be used with other AT features such as loop transformations.
- Can we choose the best execution form (semi–) automatically?

(semi-) automatic optimization of execution form

- limitation: target loop is single (not nested) loop

- generally speaking, empirically, in CUDA GPU (Kepler),
  - worker : we don't have to use worker level
  - vector : should be 32 × n (32, 64, 96, 128, ⋯, 256)
  - gang : > #SMX
    - K40 has 15 SMX units: 15, 30, 45, ⋯ will be good
    - vector × gang < #loop_length
  - example:
    - vector : 32*(1,2,⋯,8) : 8variations
    - gang : 15,30,45,60 : 4variations



total 32 variations

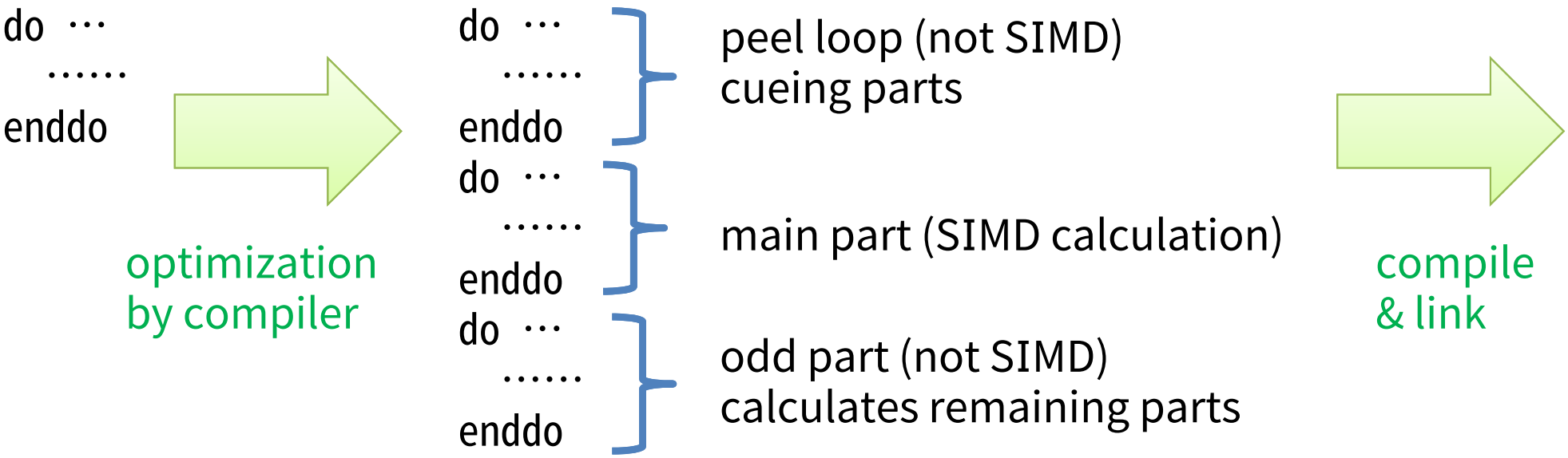
(cont.)

- It is expected that the best performance will be obtained by comparing all 32 variations.
- If the target kernel is not very heavy, 32 variations will be acceptable.
  - We can measure the performance of all 32 variations and choose the best one.
  - Because number of variations is small, performance model is not required.
  - Of course, if performance model will be useful for reducing the tuning time.
- If this tuning is used with other parameters (such as Loop Transformation), number of variations increases and performance model will be important.
  - This issue will be happen in the case of OpenMP scheduling settings.

ex. schedule(dynamic, 4)

On the other hand…

- In OpenACC, OpenACC compiler convert source code written by users directly.
- However in intel compiler, compiler modifies users' code before compiling.
  - Example of SIMD : Compiler generates peel loop and odd loop codes.
  - When making and tuning performance model, we have to consider that target code may be changed from users' codes.



Conclusion

- In this talk, ppOpen-AT for OpenACC and current advanced issues are shown.
- Some of the issues will be solved by hierarchical AT (talked by Prof.Katagiri). We have to discuss them and propose some solutions.