C2SM Center for Climate Systems Modeling

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CLAW FORTRAN Compiler

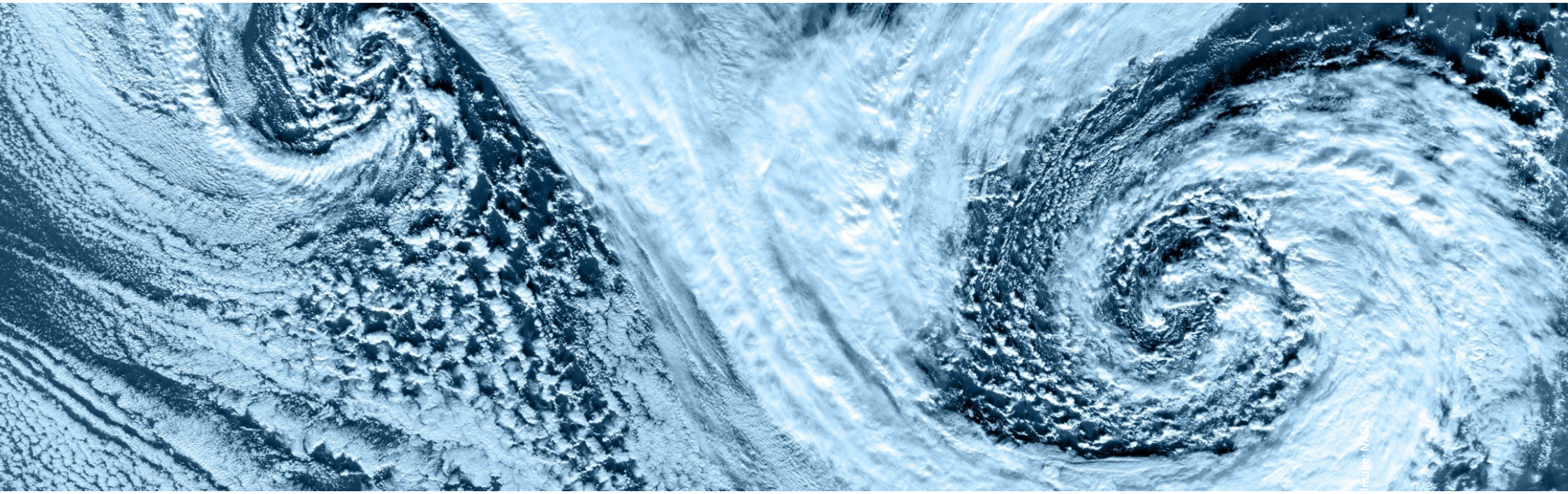
Abstractions for Weather and Climate Models

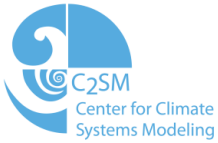
PASC'17

June 27, 2017

Valentin Clement, Jon Rood, Sylvaine Ferrachat, Will Sawyer, Oliver Fuhrer, Xavier Lapillonne

valentin.clement@env.ethz.ch

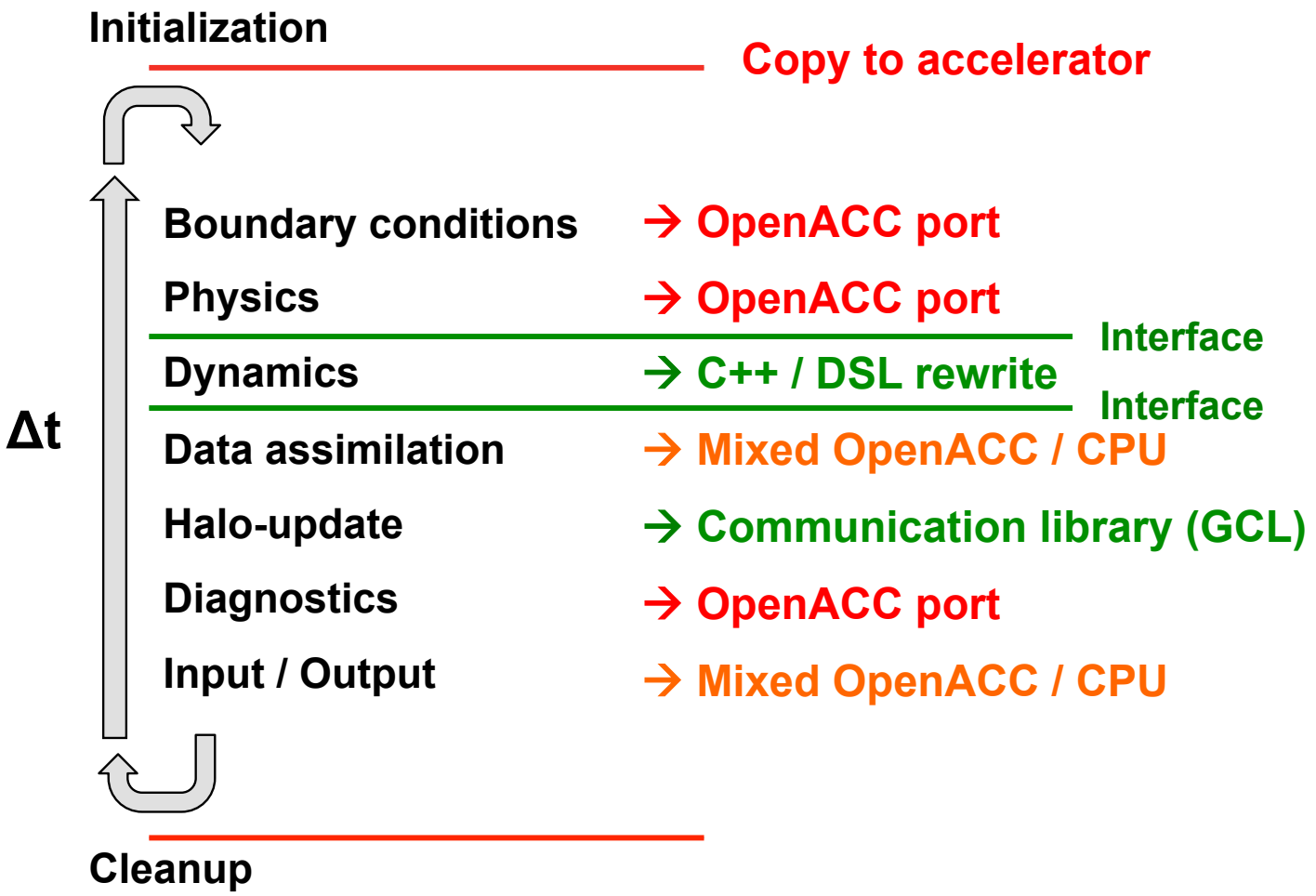




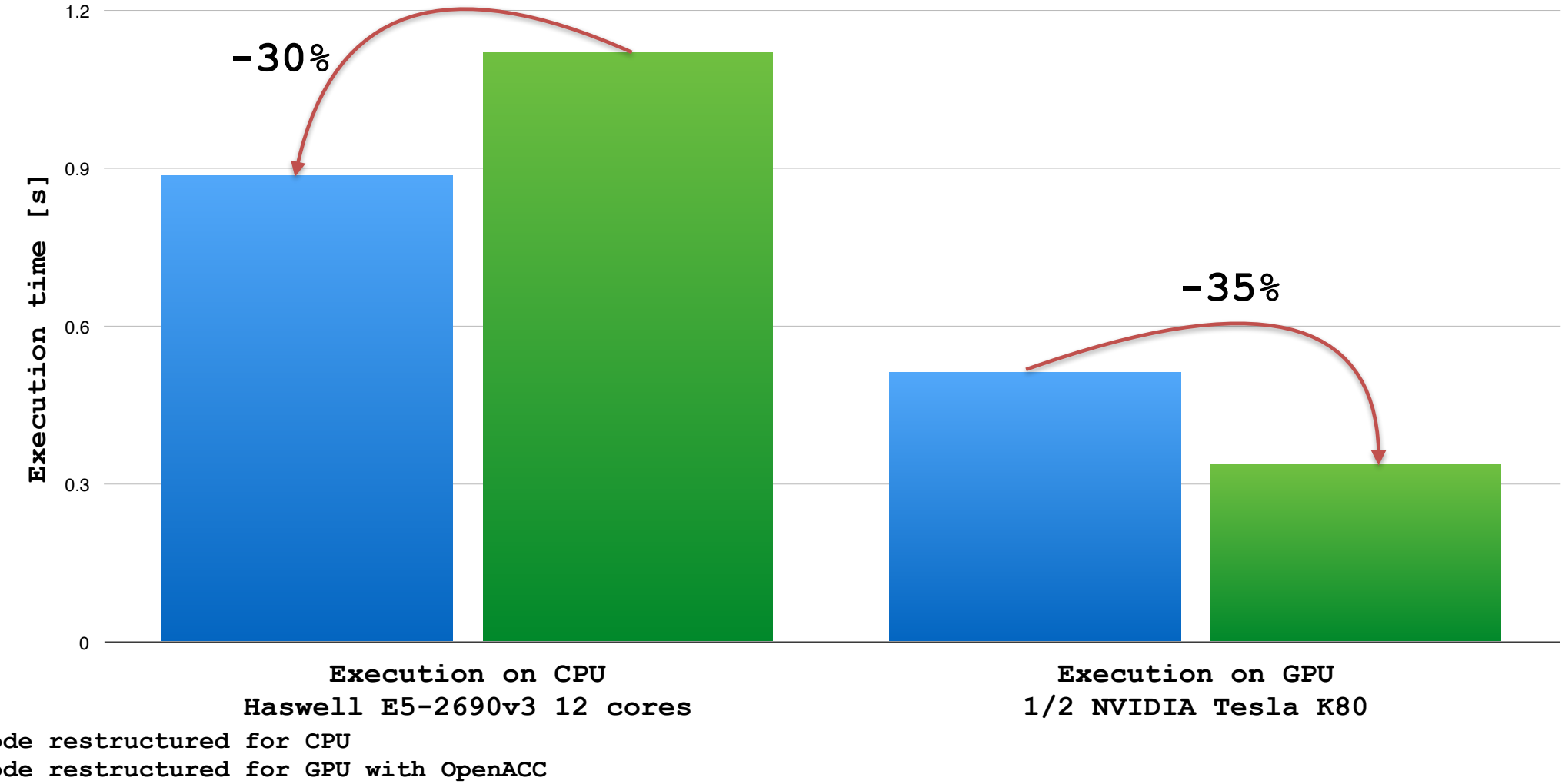
Summary

- Performance portability problem
- CLAW FORTRAN Compiler
  - CLAW low-level transformations
  - CLAW high-level abstraction

COSMO to GPU


Initialization
Copy to accelerator
Δt
Boundary conditions
→ OpenACC port
Physics
→ OpenACC port
Interface
Dynamics
→ C++ / DSL rewrite
Interface
Data assimilation
→ Mixed OpenACC / CPU
Halo-update
→ Communication library (GCL)
Diagnostics
→ OpenACC port
Input / Output
→ Mixed OpenACC / CPU
Cleanup

Performance portability problem - COSMO Radiation



Performance portability

In some cases CPU and GPU have different optimization requirements

CPU:

- Auto-vectorization: small loops
- Pre-computation

GPU:

- Benefit from large kernels (loop fusion): reduce kernel launch overhead, better computation/memory access overlap
- Loop re-ordering and scalar replacement
- On the fly computation

Performance portability - COSMO Radiation

Iteration over 2 dimensions (j = horizontal, k = vertical)

```
DO k=1,nz
  ! contains j loop
  CALL fct()
  DO j=1,nproma
    ! 1st loop body
  END DO
  DO j=1,nproma
    ! 2nd loop body
  END DO
  DO j=1,nproma
    ! 3rd loop body
  END DO
END DO
```

CPU optimal

```
!$acc parallel loop
DO j=1,nproma
  !$acc loop
  DO k=1,nz
    CALL fct()
    ! 1st loop body
    ! 2nd loop body
    ! 3rd loop body
  END DO
END DO
!$acc end parallel
```

GPU optimal

Performance portability - solution 2 code paths w/ #ifdef

```
#ifndef _OPENACC
DO k=1,nz
  CALL fct()
  DO j=1,nproma
    ! 1st loop body
  END DO
  DO j=1,nproma
    ! 2nd loop body
  END DO
  DO j=1,nproma
    ! 3rd loop body
  END DO
END DO
#else
!$acc parallel loop
DO j=1,nproma
  !$acc loop
  DO k=1,nz
    CALL fct()
    ! 1st loop body
    ! 2nd loop body
    ! 3rd loop body
  END DO
END DO
!$acc end parallel
#endif
```

CPU

GPU

- Multiple code paths
- Hard maintenance
- Error prone
- Domain scientists have to know well each target architectures

Performance portability - portable between what?

- Between different CPUs?
- Between different compilers for the same architecture?
- Between different architectures CPU vs CPU/GPU vs MIC?

“Most people are resigned to having different sources for different platforms, with simple #ifdef or other mechanisms”

DOE workshop output on performance portability

Can we do it better for our FORTRAN code than #ifdef?

CLAW low-level directives - loop-fusion

```
!$claw loop-fusion [group(group_id)] [collapse(n)]
```

```
!$claw loop-fusion collapse(2)
DO j = 1, nproma
  DO k = 1, kend
    ! Body 1
  END DO
END DO

!$claw loop-fusion collapse(2)
DO j = 1, nproma
  DO k = 1, kend
    ! Body 2
  END DO
END DO
```

original code

```
DO j = 1, nproma
  DO k = 1, kend
    ! Body 1
    ! Body 2
  END DO
END DO
```

transformed code

CLAW low-level directives - loop-interchange

```
!$claw loop-interchange [(induction_var[, induction_var] ...)]
```

```
!$claw loop-interchange
DO j = 1, nproma
  DO k = 1, kend
    ! Body
  END DO
END DO
```

```
!$claw loop-interchange (k,i,j)
DO i = 1, iend
  DO j = 1, jend
    DO k = 1, kend
      ! Body
    END DO
  END DO
END DO
```

original code

```
DO k = 1, kend
  DO j = 1, nproma
    ! Body
  END DO
END DO
```

```
DO k = 1, kend
  DO i = 1, iend
    DO j = 1, jend
      ! Body
    END DO
  END DO
END DO
```

transformed code

CLAW low-level directive transformations

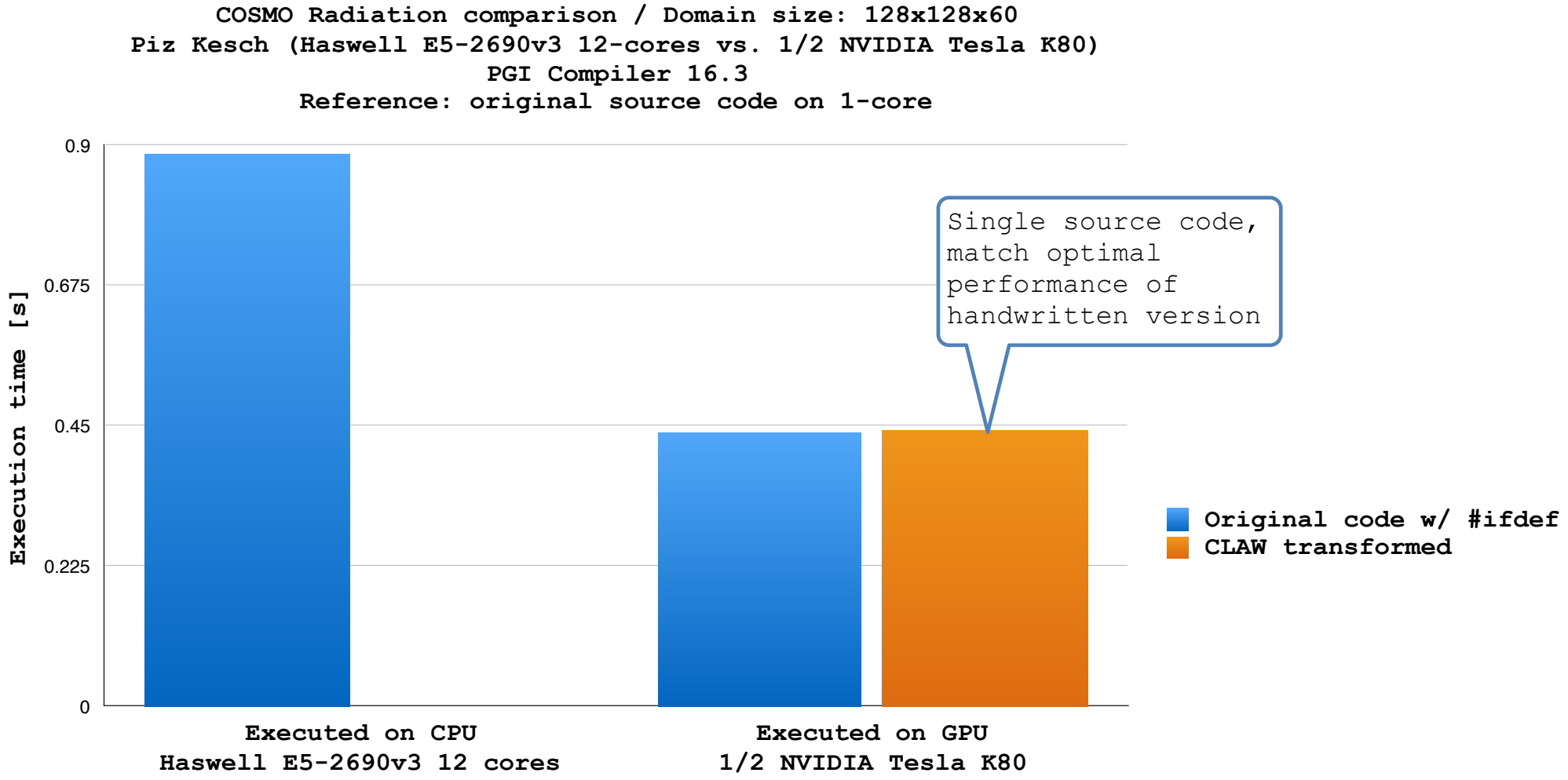
```
!$acc parallel loop
!$claw loop-interchange
DO k=1,nz
  !$claw loop-extract fusion
  CALL fct()
  !$claw loop-fusion group(j)
  !$acc loop
  DO j=1,nproma
    ! 1st loop body
  END DO
  !$claw loop-fusion group(j)
  !$acc loop
  DO j=1,nproma
    ! 2nd loop body
  END DO
  !$claw loop-fusion group(j)
  !$acc loop
  DO j=1,nproma
    ! 3rd loop body
  END DO
END DO
!$acc end parallel
```

clawfc

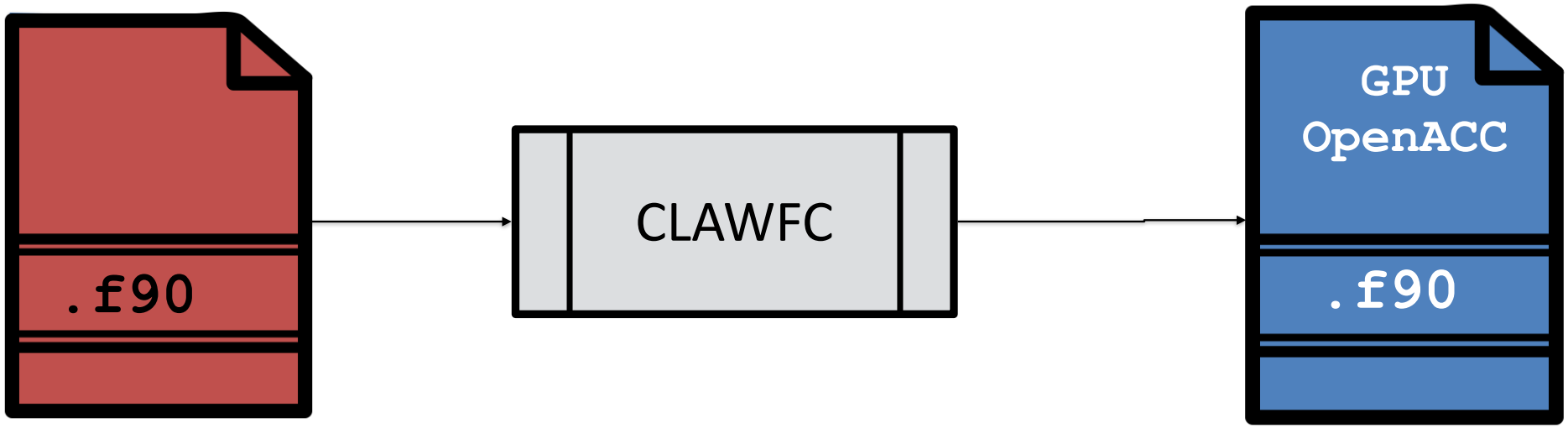
CPU to GPU

```
!$acc parallel loop
DO j=1,nproma
  !$acc loop
  DO k=1,nz
    CALL fct()
    ! 1st loop body
    ! 2nd loop body
    ! 3rd loop body
  END DO
END DO
!$acc end parallel
```

CLAW low-level transformations - COSMO radiation



CLAW FORTRAN Compiler - low-level transformations

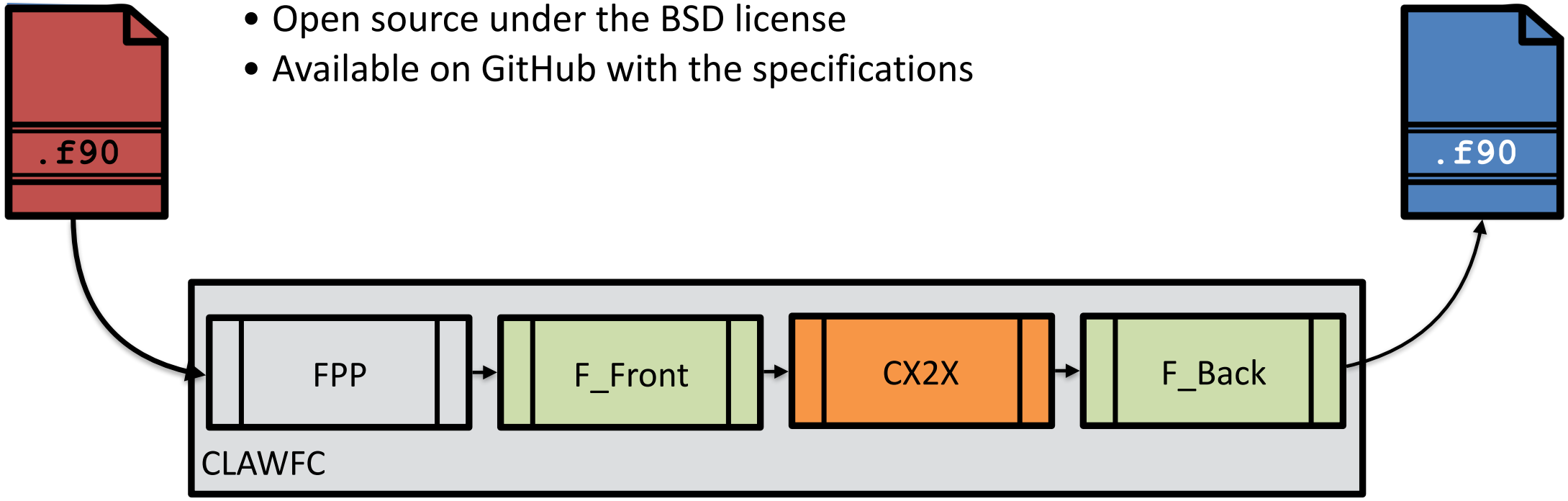


Original code with
`!$claw` directives

Transformed code

CLAW FORTRAN Compiler overview

- Based on the OMNI Compiler FORTRAN front-end & back-end
- Source-to-source translator
- Open source under the BSD license
- Available on GitHub with the specifications



OMNI Compiler

Sets of programs/libraries to build source-to-source compilers for C and FORTRAN via an XcodeML high-level intermediate representation

- Under LGPL v3 license
- Used to implement
  - XcalableMP 1.0 / 2.0, XcalableACC (XcalableMP + OpenACC), OpenMP, OpenACC

Development team

- Programming Environments Research Team from the RIKEN Advanced Institute for Computational Sciences, Kobe, Japan (K Computer, 7th)
- High Performance Computing System Lab, University of Tsukuba, Tsukuba, Japan (Oakforest-PACS, 6th)

http://www.omni-compiler.org
http://www.xcalablemp.org
https://github.com/omni-compiler

CLAW low-level directive transformations

- **Loop transformations**
  - loop fusion (`!$claw loop-fusion`)
  - loop reordering (`!$claw loop-interchange`)
  - loop hoisting (`!$claw loop-hoist`)
  - loop extraction (`!$claw loop-extract`)
  - if-else extraction (`!$claw if-extract`)
- **Specific transformation**
  - kcaching (`!$claw kcache`)
  - on-the-fly computation (`!$claw call`)
- **Utility transformation**
  - vector notation to do statements (`!$claw array-transform`)
  - code removal (`!$claw remove`)
  - ignore section (`!$claw ignore`)
  - conditional primitive directive / code (`!$claw acc/omp`)

Too many directives? Too complicated?

```
!$acc parallel loop
!$claw loop-interchange
DO k=1,nz
  !$claw loop-fusion
  !$acc loop
  !$omp parallel do
  DO j=1,nproma
    ! 1st loop body
  END DO
  !$omp end parallel do
  !$claw loop-fusion
  !$acc loop
  !$omp parallel do
  DO j=1,nproma
    ! 2nd loop body
  END DO
  !$omp end parallel do
  !$claw loop-fusion group(j)
  !$acc loop
  !$omp parallel do
  DO j=1,nproma
    ! 3rd loop body
  END DO
  !$omp end parallel do
END DO
!$acc end parallel
```

Typical code could includes the following compiler directives:

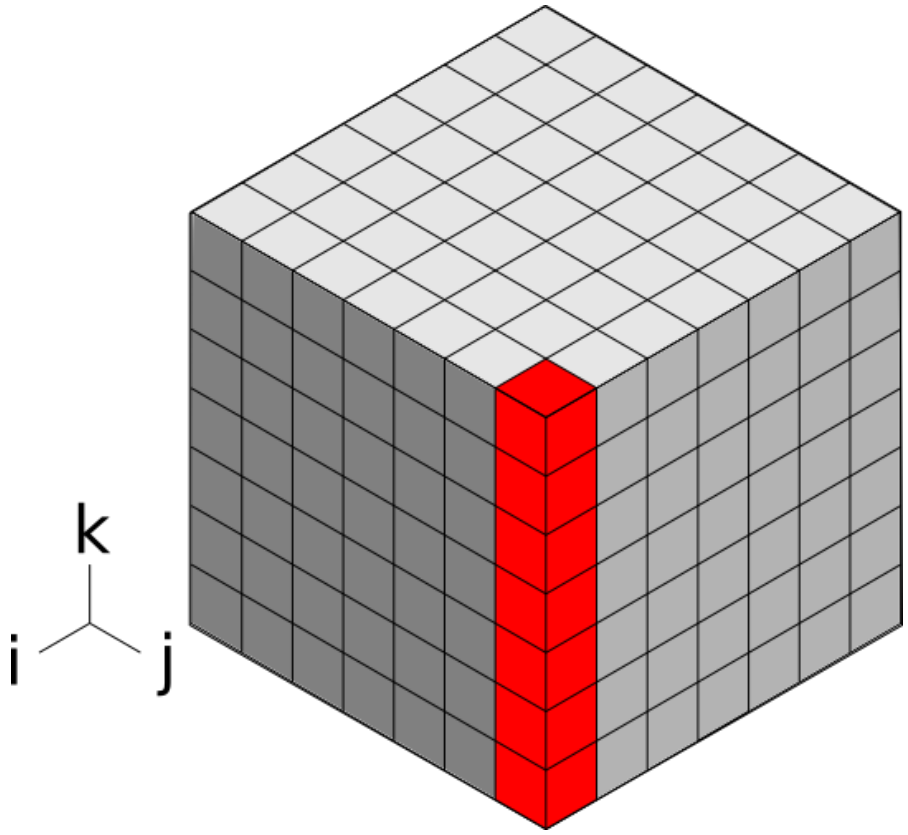
- OpenMP
- OpenACC
- CLAW
- …

Still targeted for performance aware developers.

Doesn't help the domain scientists.

Can we do it in a simpler way?

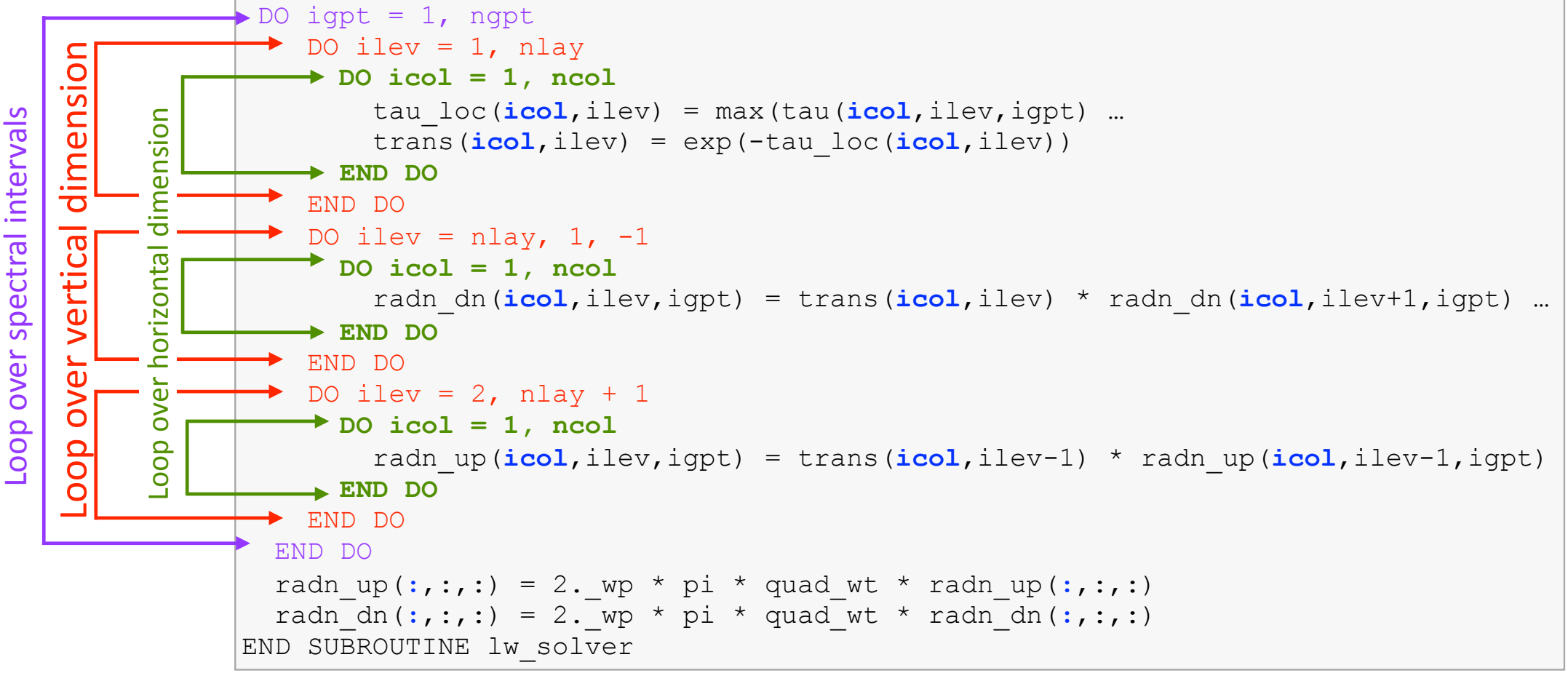
CLAW One column abstraction



Separation of concerns

- Domain scientists focus on their problem (1 column, 1 box)
- CLAW compiler produce code for each target and directive languages

RRTMGP Example - original code - CPU structured




```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
 ! DECLARATION PART OMITTED
 DO igpt = 1, ngpt
   DO ilev = 1, nlay
     DO icol = 1, ncol
         tau_loc(icol,ilev) = max(tau(icol,ilev,igpt) …
         trans(icol,ilev) = exp(-tau_loc(icol,ilev))
     END DO
   END DO
   DO ilev = nlay, 1, -1
     DO icol = 1, ncol
         radn_dn(icol,ilev,igpt) = trans(icol,ilev) * radn_dn(icol,ilev+1,igpt) …
     END DO
   END DO
   DO ilev = 2, nlay + 1
     DO icol = 1, ncol
         radn_up(icol,ilev,igpt) = trans(icol,ilev-1) * radn_up(icol,ilev-1,igpt)
     END DO
   END DO
  END DO
  radn_up(:,:,:) = 2._wp * pi * quad_wt * radn_up(:,:,:)
  radn_dn(:,:,:) = 2._wp * pi * quad_wt * radn_dn(:,:,:)
END SUBROUTINE lw_solver
```

RRTMGP Example - one column only

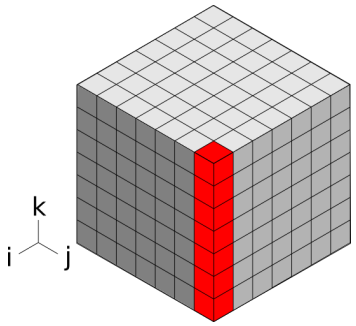
Loop over vertical dimension

```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
  ! DECLARATION PART OMITTED
  DO igpt = 1, ngpt
    DO ilev = 1, nlay
      tau_loc(ilev) = max(tau(ilev,igpt) …
      trans(ilev) = exp(-tau_loc(ilev))
    END DO
    DO ilev = nlay, 1, -1
      radn_dn(ilev,igpt) = trans(ilev) * radn_dn(ilev+1,igpt) …
    END DO
    DO ilev = 2, nlay + 1
      radn_up(ilev,igpt) = trans(ilev-1) * radn_up(ilev-1,igpt)
    END DO
  END DO
  radn_up(:,:) = 2._wp * pi * quad_wt * radn_up(:,:)
  radn_dn(:,:) = 2._wp * pi * quad_wt * radn_dn(:,:)
END SUBROUTINE lw_solver
```

RRTMGP Example - CLAW code

Algorithm for one column only

```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
 !$claw define dimension icol(1:ncol) &
 !$claw parallelize
 DO igpt = 1, ngpt
   DO ilev = 1, nlay
      tau_loc(ilev) = max(tau(ilev,igpt) …
      trans(ilev) = exp(-tau_loc(ilev))
    END DO
    DO ilev = nlay, 1, -1
      radn_dn(ilev,igpt) = trans(ilev) * radn_dn(ilev+1,igpt) …
    END DO
    DO ilev = 2, nlay + 1
      radn_up(ilev,igpt) = trans(ilev-1) * radn_up(ilev-1,igpt)
    END DO
  END DO
  radn_up(:,:) = 2._wp * pi * quad_wt * radn_up(:,:)
  radn_dn(:,:) = 2._wp * pi * quad_wt * radn_dn(:,:)
END SUBROUTINE lw_solver
```

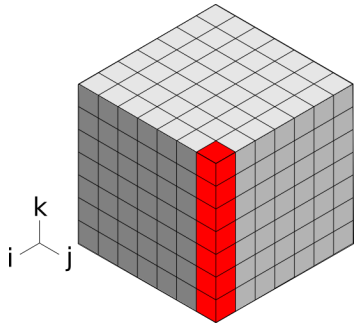


Dependency on the vertical dimension only

RRTMGP Example - GPU w/ OpenACC

```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
! DECLARATION W/ PROMOTED VARIABLES
!$acc data present(…)
!$acc parallel
!$acc loop gang vector private(…)
DO icol = 1 , ncol , 1
  !$acc loop seq
    DO igpt = 1 , ngpt , 1
    !$acc loop seq
    DO ilev = 1 , nlay , 1
      tau_loc(ilev) = max(tau(icol,ilev,igpt)
      trans(ilev) = exp(-tau_loc(ilev))
    END DO
    !$acc loop seq
    DO ilev = nlay , 1 , (-1)
      radn_dn(icol,ilev,igpt) = trans(ilev) * radn_dn(icol,ilev+1,igpt)
    END DO
   !$acc loop seq
   DO ilev = 2 , nlay + 1 , 1
      radn_up(icol,ilev,igpt) = trans(ilev-1)*radn_up(icol,ilev-1,igpt)
    END DO
  END DO
  !$acc loop seq
  DO igpt = 1 , ngpt , 1
    !$acc loop seq
    DO ilev = 1 , nlay + 1 , 1
      radn_up(icol,igpt,ilev) = 2._wp * pi * quad_wt * radn_up(icol,igpt,ilev)
      radn_dn(icol,igpt,ilev) = 2._wp * pi * quad_wt * radn_dn(icol,igpt,ilev)
    END DO
  END DO
END DO
!$acc end parallel
!$acc end data
END SUBROUTINE lw_solver
```

CLAW - One column - transformation rules



Automatic promotions

- Inside the parallelized subroutine
- Along the call graph

Iteration over the horizontal dimensions

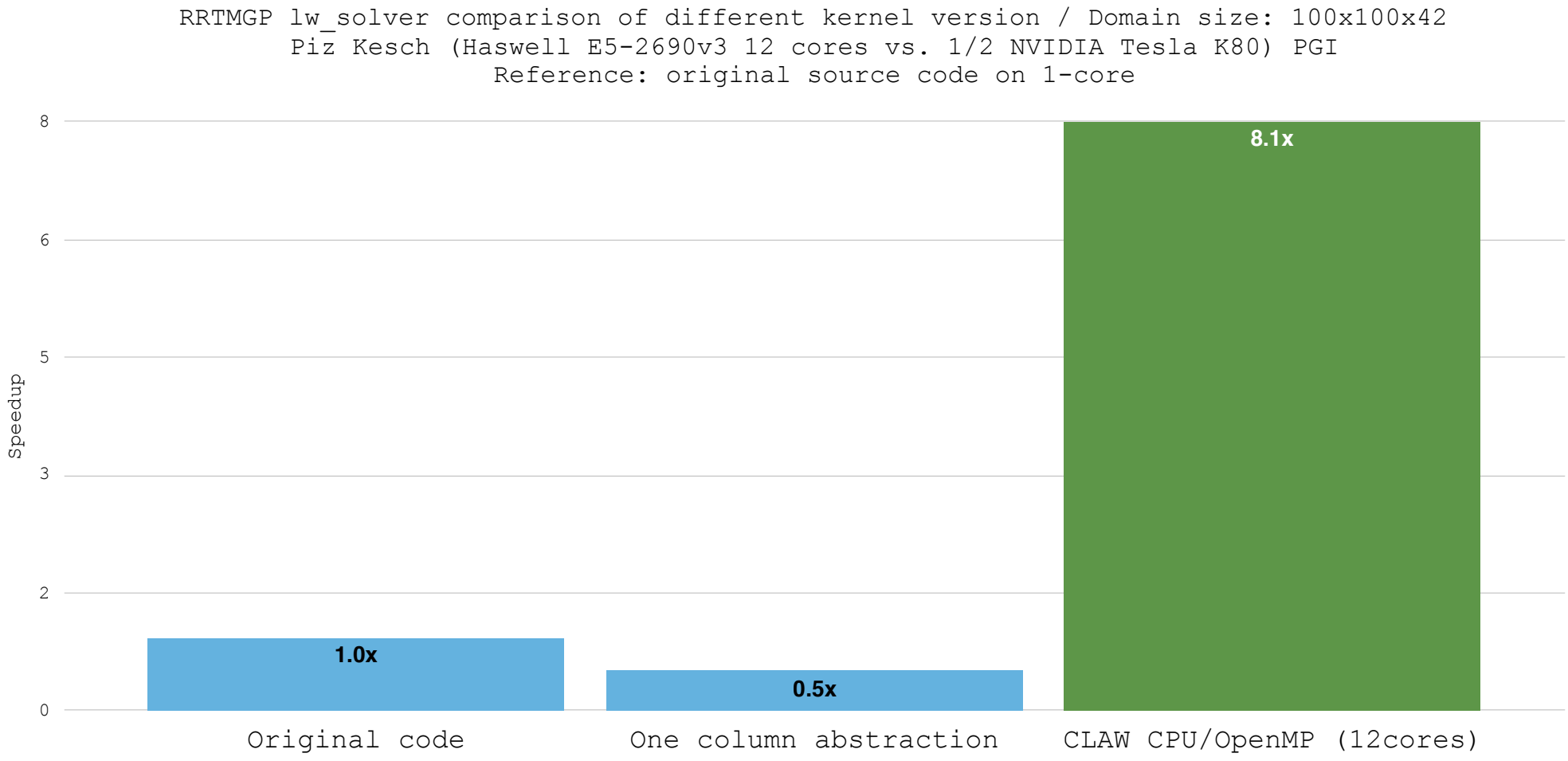
- Specific generation of do statements according to the architecture and data layout

Generation of directives for inserted and existing do statements

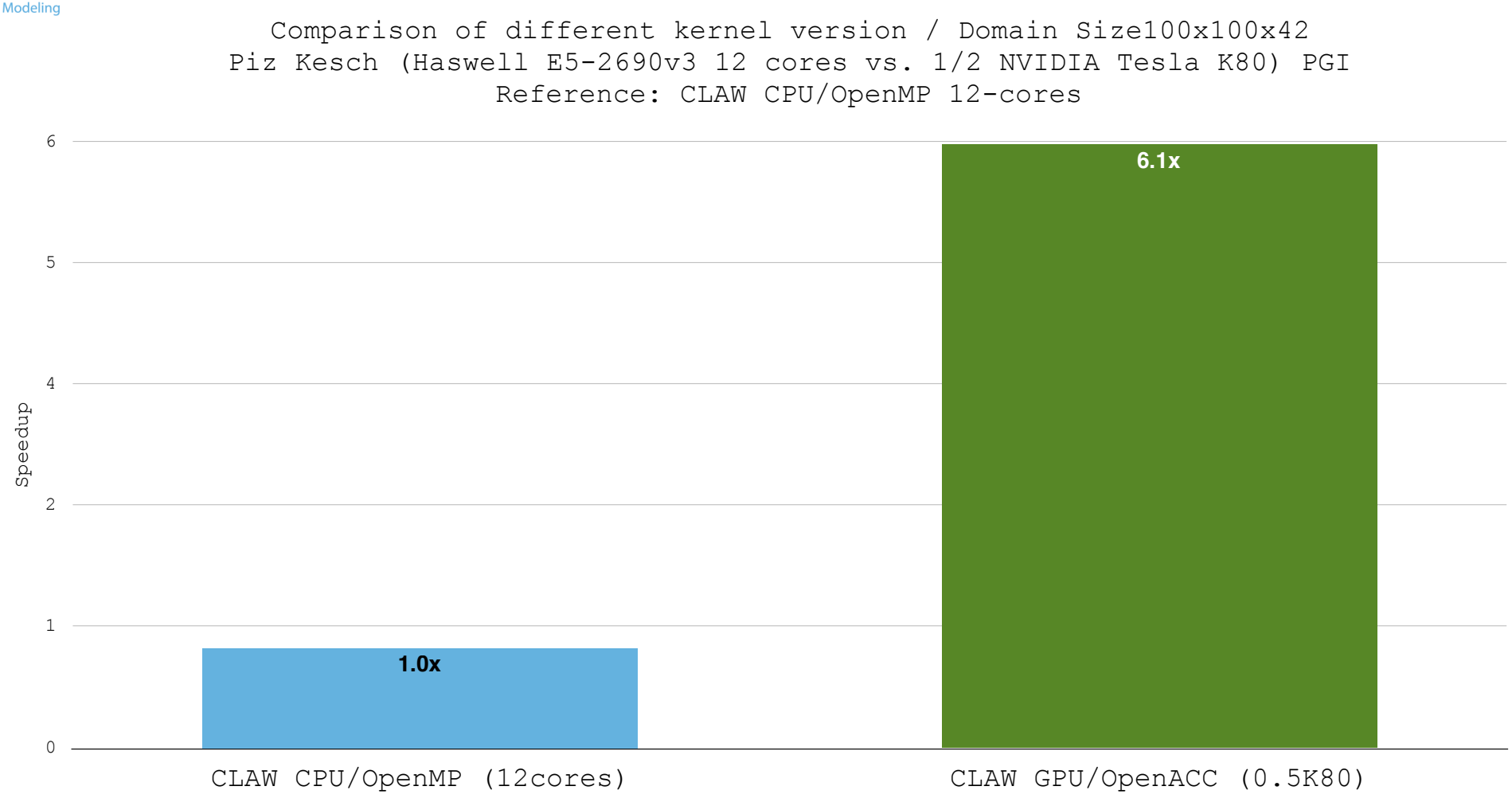
- OpenMP
- OpenACC

Currently, transformation rules are based on observation made in COSMO, HAMMOZ or ICON. Goal would be to add intelligence here or to couple it with tools that can drives the transformations

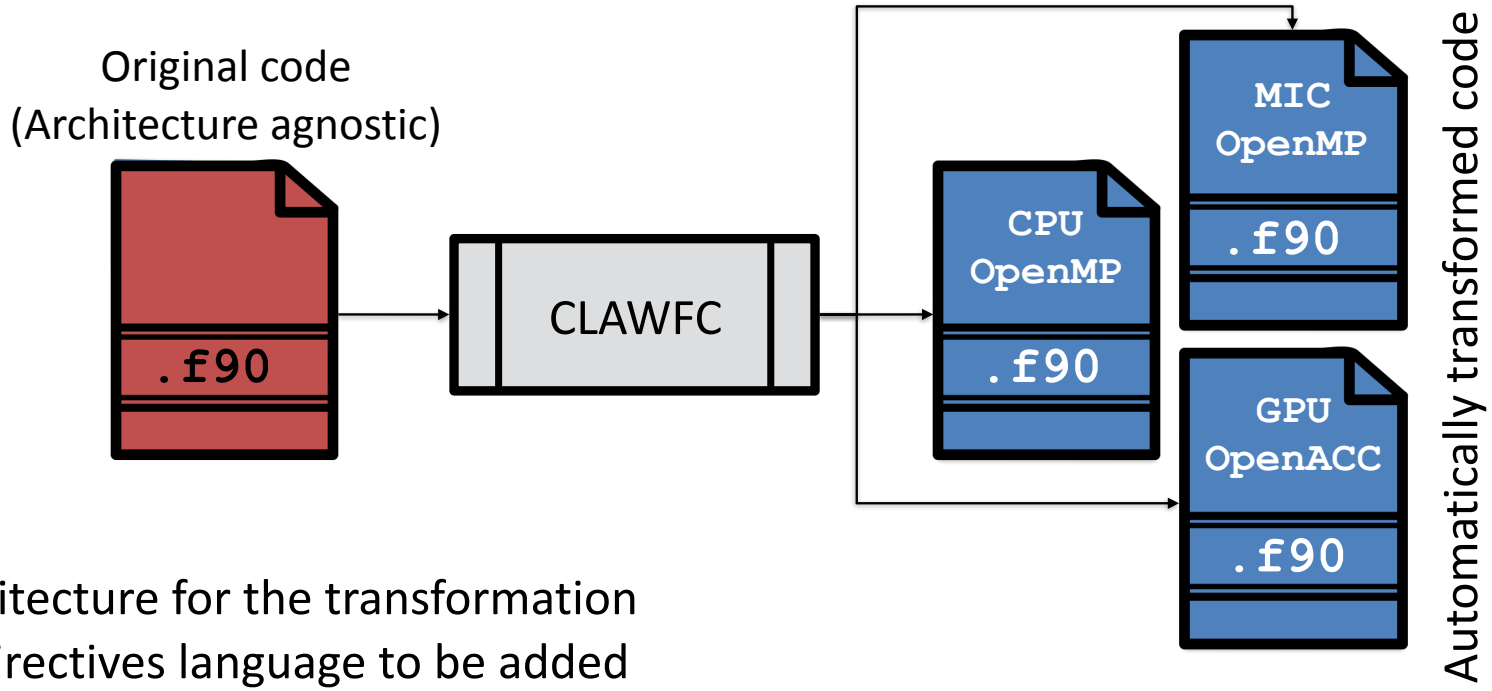
RRTMGP lw_solver - Original vs. CLAW CPU/OpenMP



RRTMGP lw_solver - CLAW CPU vs. CLAW GPU



CLAW - Apply transformation



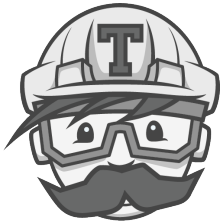
- A single source code
- Specify a target architecture for the transformation
- Specify a compiler directives language to be added

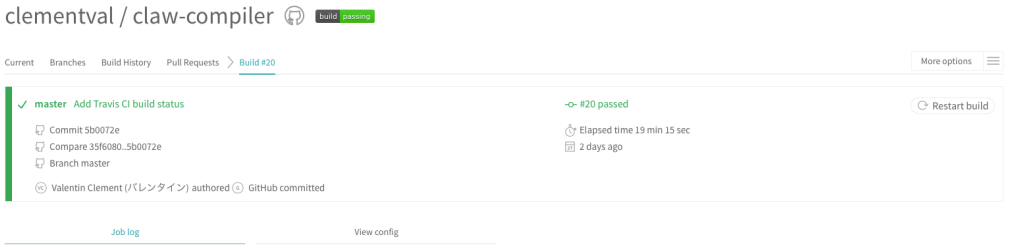
```
clawfc --directive=openacc --target=gpu -o mo_lw_solver.acc.f90 mo_lw_solver.f90
clawfc --directive=openmp --target=cpu -o mo_lw_solver.omp.f90 mo_lw_solver.f90
clawfc --directive=openmp --target=mic -o mo_lw_solver.mic.f90 mo_lw_solver.f90
```

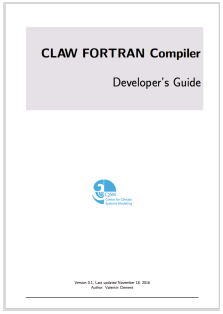
CLAW FOTRAN Compiler - Resources

https://github.com/C2SM-RCM/claw-compiler

https://github.com/omni-compiler







CLAW FORTRAN Compiler developer's guide

CLAW - Future

- **One column abstraction / IJK abstraction**
  - Continue investigation with RRTMGP
  - Refinement of directive specifications, transformations and directives generation
- **ENIAC: Enabling the ICON on heterogenous architectures**
  - PASC Project from July 2017
  - Work packages:
    - Consolidation and extension of CLAW with inputs from GPU and XeonPhi port (specific XeonPhi transformation + generation)
    - Getting CLAWFC in a production stage
    - IJK and one column abstraction
    - Possible link between CLAWFC and GridTools

C2SM Center for Climate Systems Modeling

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

valentin.clement@env.ethz.ch

Feedbacks or collaborations much welcome!

`https://github.com/C2SM-RCM/claw-compiler`

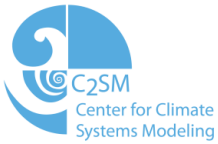
`https://github.com/omni-compiler`

C2SM
Center for Climate
Systems Modeling

Backup slides

https://github.com/C2SM-RCM/claw-compiler

https://github.com/omni-compiler



XcodeML/F 2008 Specifications

- Describe all FORTRAN 2008 via high-level intermediate representation based on XML
  - For each type of node in the AST
    - Description of required and optional children
    - Description of required and optional attributes
- XcodeML/F 2008 is currently a draft
  - XcodeML/F 95 is available currently
  - I have access to the draft if you need it
  - Collaboration with RIKEN to finalize the draft (review)

XcodeML/F 2008 Specifications

日本語を話せますか？

They are working on an english version as well…

6.5 FdoStatement 要素

FdoStatement 要素は、DO 文を表現する。

文番号 DO 文は、文番号を削除した等価な DO 文に置き換えて指定しなければならない。

内容モデル

```
(Var?, indexRange?, body?)
```

子要素

要素	説明	指定
Var	DO 変数を指定する。	O
indexRange	DO 変数の値範囲を指定する。	O
body	DO 文に含まれる文を指定する。	O

属性

属性	型	説明	指定
共通属性	—	『9.2 定義／宣言／文要素の共通属性』を参照。	—

XcodeML/F Example - FORTRAN original code

```
PROGRAM test
  CALL my_kernel(10)
END PROGRAM test

SUBROUTINE my_kernel(nz)
  INTEGER, INTENT(IN) :: nz
  INTEGER :: i
  REAL(KIND=8) :: a(nz)

  DO i = 1, nz
    a(i) = i
  END DO
END SUBROUTINE my_kernel
```

XcodeML/F Example - XcodeProgram

```
<XcodeProgram source="sample.f90" language="Fortran"
              time="2016-11-24 14:20:22"
              compiler-info="XcodeML/Fortran-FrontEnd" version="1.0">
  <typeTable>
      <!-- omitted code here -->
  </typeTable>
  <globalSymbols>
    <id type="F7fa2c9c067d0" sclass="ffunc">
      <name>test</name>
    </id>
    <id type="F7fa2c9c06d30" sclass="ffunc">
      <name>my_kernel</name>
    </id>
  </globalSymbols>
  <globalDeclarations>
    <FfunctionDefinition lineno="1" file="sample.f90">
      <!-- omitted code here -->
    </FfunctionDefinition>
    <FfunctionDefinition lineno="5" file="sample.f90">
      <!-- omitted code here -->
    </FfunctionDefinition>
  </globalDeclarations>
</XcodeProgram>
```

XcodeML/F Example - typeTable

```
<XcodeProgram>
 <typeTable>
  <FbasicType type="I7fa8ac505ef0" intent="in" ref="Fint"/>
  <FbasicType type="R7fa8ac506db0" ref="Freal">
   <kind><FintConstant type="Fint">8</FintConstant></kind>
  </FbasicType>
  <FbasicType type="R7fa8ac506f50" ref="R7fa8ac506db0"/>
  <FbasicType type="A7fa8ac507020" ref="R7fa8ac506f50">
   <indexRange>
    <lowerBound><FintConstant type="Fint">1</FintConstant></lowerBound>
    <upperBound><Var type="I7fa8ac505ef0" scope="local">nz</Var></upperBound>
   </indexRange>
  </FbasicType>
  <FfunctionType type="F7fa8ac5041e0" return_type="Fvoid" is_program="true"/>
  <FfunctionType type="F7fa8ac5047b0" return_type="Fvoid">
   <params>
    <name type="I7fa8ac505ef0">nz</name>
   </params>
  </FfunctionType>
 </typeTable>
 <globalSymbols></globalSymbols>
 <globalDeclarations></globalDeclarations>
</XcodeProgram>
```

XcodeML/F Example - FfunctionDefinition (1/2)

```
<globalDeclarations>
   <FfunctionDefinition lineno="1" file="sample.f90">
 </FfunctionDefinition>
   <FfunctionDefinition lineno="5" file="sample.f90">
     <name type="F7fa2c9c06d30">my_kernel</name>
     <symbols>
       <id type="F7fa2c9c06d30" sclass="ffunc"><name>my_kernel</name></id>
       <id type="I7fa2c9c08430" sclass="fparam"><name>nz</name></id>
       <id type="Fint" sclass="flocal"><name>i</name></id>
       <id type="A7fa2c9c09310" sclass="flocal"><name>a</name></id>
     </symbols>
     <declarations>
       <varDecl lineno="6" file="sample.f90">
         <name type="I7fa2c9c08430">nz</name>
       </varDecl>
       <varDecl lineno="7" file="sample.f90">
         <name type="Fint">i</name>
       </varDecl>
       <varDecl lineno="8" file="sample.f90">
         <name type="A7fa2c9c09310">a</name>
       </varDecl>
     </declarations>
```

XcodeML/F Example - FfunctionDefinition (2/2)

```
<body>
 <FdoStatement lineno="10" file="sample.f90">
   <Var type="Fint" scope="local">i</Var>
    <indexRange>
      <lowerBound><FintConstant type="Fint">1</FintConstant></lowerBound>
      <upperBound><Var type="I7fa2c9c08430" scope="local">nz</Var></upperBound>
      <step><FintConstant type="Fint">1</FintConstant></step>
    </indexRange>
    <body>
     <FassignStatement lineno="11" file="sample.f90">
       <FarrayRef type="Freal">
         <varRef type="A7fa2c9c09310">
           <Var type="A7fa2c9c09310" scope="local">a</Var>
          </varRef>
          <arrayIndex>
           <Var type="Fint" scope="local">i</Var>
          </arrayIndex>
        </FarrayRef>
        <Var type="Fint" scope="local">i</Var>
      </FassignStatement>
    </body>
  </FdoStatement>
</body></FfunctionDefinition></globalDeclarations></XcodeProgram>
```

Transformation class

```
public class Parallelize extends Transformation
  public boolean analyze(XcodeProgram xcodeml, Transformer t){
    // Check if the transformation is possible
    return canBeTransformed;
  }

 public void transform(XcodeProgram xcodeml, Transformer t){
   if(_claw.getTarget() == Target.GPU){
     // Apply specific GPU transformation
   } else if(_claw.getTarget() == Target.CPU){
     // Apply specific CPU transformation
   } else if (_claw.getTarget == Target.MIC){
     // Apply specific MIC transformation
   }
 }
}
```

- Each transformation is an object with an analysis and a transformation step.
- Transformation objects have access to various information such as: target architecture, desired directives language …