Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

The CLAW project

C**LAW provides high-**L**evel** A**bstractions for** W**eather and climate models**

Valentin Clément, Xavier Lapillonne

C2SM
Center for Climate
Systems Modeling

Summary

- Approaches to performance portability
- CLAW language definition
- Reference compiler

Different optimization requirements in the physics on CPUs and GPUs

Profiler information in the physics:

- **CPU :** compute bound
  - Compiler auto-vectorization : easier with small loop construct
  - Pre-computation

- **GPU :** memory bound limited
  - Benefit from large kernels : reduce kernel launch overhead, better computation/memory access overlap
  - Loop re-ordering and scalar replacement
  - On the fly computation

Xavier Lapillonne xavier.lapillonne@meteoswiss.ch

Example *(pseudo code)*

Original code

```
do k=2,nk
 do i=1,ni
    temp(i) = SomeFunct(a(i,k-1))
 end do
 do i=1,ni
    c(i) = D*exp(temp(i))
 end do
 do i=1,ni
    a(i,k)=c(i)*a(i,k)
 end do
end do
```

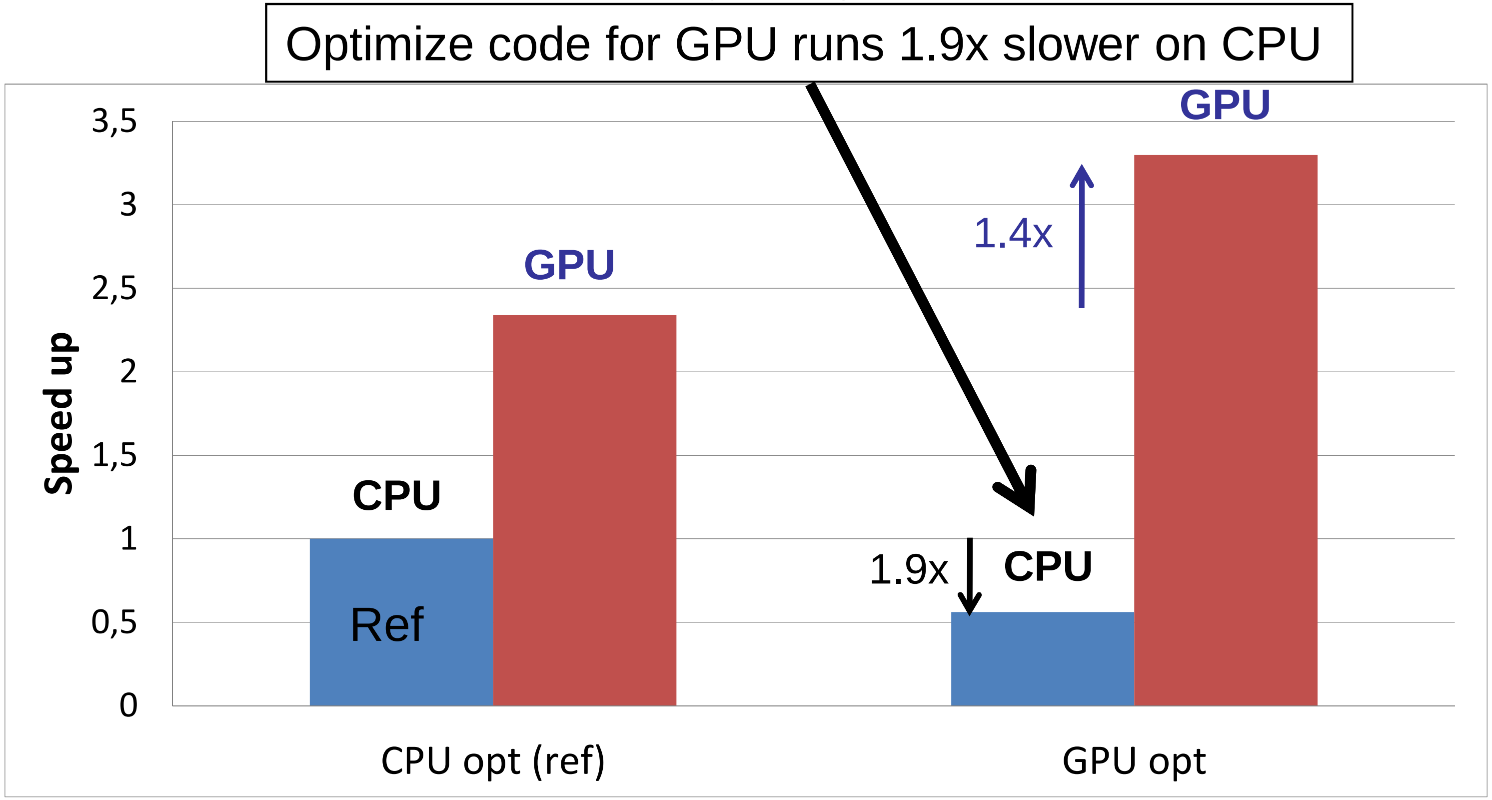
Optimize for GPU

```
!$acc parallel
!$acc loop gang vector
do k=2,nk
 do i=1,ni
    temp = SomeFunct(a(i,k-1))
    c = D*exp(temp)
    a(i,k)=c*a(i,k)
 end do
end do
!$acc end parallel
```

May not vectorize on CPU

Example in the radiation

- Considering inv_th/so routine (90% of radiation time)
- Test domain 128x128x60, 1 Sandy Bridge CPU vs 1 K20x CPU



Speed up with respect to reference code on CPU

Radiation is the strongest example, on average in the physics, code optimized for GPU is 1.3x times slower when run on CPU

Current approach

- Different code paths with preprocessor macros

```
SUBROUTINE inv_th(pclc,pca1, …)
  INTEGER(KIND=iintegers), INTENT (IN) :: ki1sd
  ! More declarations
  ! Shared code between CPU/GPU

#ifdef _OPENACC
  ! GPU optimized code
#else
  ! CPU optimized code
#endif
 ! Shared code between CPU/GPU
END SUBROUTINE inv_th
```

Write efficient code for each targeted architecture
-Hard to maintain for developers
-Can be compile with standard compilers
-Fine tuning possible

CLAW approach

- Directives with code transformation

```
SUBROUTINE inv_th(pclc,pca1, …)
  INTEGER(KIND=iintegers), INTENT(IN) :: ki1sd

  !$acc parallel
  !$acc loop collapse(3)
  !$claw loop-interchange (k,i,j)
  DO i=istart,iend
    DO j=jstart,jend
      DO k=kstart,kend
        ! Computation is done here
      END DO
    END DO
  END DO
  !$acc end parallel

END SUBROUTINE inv_th
```

CLAW (iteration 1)
-Code manipulation with AST
-Can compile with standard compiler
-Fortran standard not changed

Other tools:
-**F2C-ACC** (Specific for NIM, was not designed for external use. Want to move to std compiler)

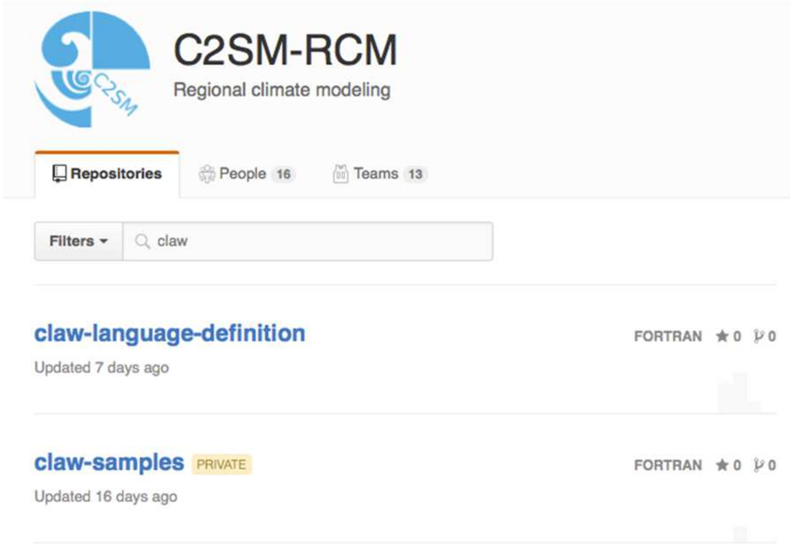
-**OpenACC** (PGI Compiler > 15.10, tries to use OpenACC also for CPU parallelism)

CLAW language definition

- Available on GitHub:
  - https://github.com/C2SM-RCM/claw-language-definition
- Currently defined as a directive language

```
!$claw directive options
```

- **Iteration 1**: Definition of low-level code transformation
- **Iteration 2 … N**: Refining the code transformation and **evolving** to higher abstraction for climate system and weather models



CLAW language definition

```
!$claw loop-interchange (k,i,j)
DO i=1, iend      ! loop at depth 0
  DO j=1, jend    ! loop at depth 1
    DO k=1, kend ! loop at depth 2
      ! loop body here
    END DO
  END DO
END DO
```


Xavier Lapillonne xavier.lapillonne@meteoswiss.ch

CLAW language definition

```
!$claw loop-interchange (k,i,j)
DO k=1, kend      ! loop at depth 2
  DO i=1, iend    ! loop at depth 0
    DO j=1, jend ! loop at depth 1
      ! loop body here
    END DO
  END DO
END DO
```

CLAW language definition

```
DO k=1, iend
  !$claw loop-fusion group(g1)
  DO i=1, iend
    ! loop #1 body here
  END DO

  !$claw loop-fusion group(g1)
  DO i=1, iend
    ! loop #2 body here
  END DO

  !$claw loop-fusion group(g2)
  DO i=1, jend
    ! loop #3 body here
  END DO

  !$claw loop-fusion group(g2)
  DO i=1, jend
    ! loop #4 body here
  END DO
END DO
```

CLAW language definition

```
DO k=1, iend




  !claw loop-fusion group(g1)
  DO i=1, iend
    ! loop #1 body here
    ! loop #2 body here
  END DO

  !claw loop-fusion group(g2)
  DO i=1, jend
    ! loop #3 body here
    ! loop #4 body here
  END DO




END DO
```

OMNI Compiler

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

-Used to implement XcalableMP (abstract inter-node communication), XcalableACC (XMP + OpenACC), OpenMP (implementation for C and Fortran), OpenACC (C implementation only)

•**Development team**

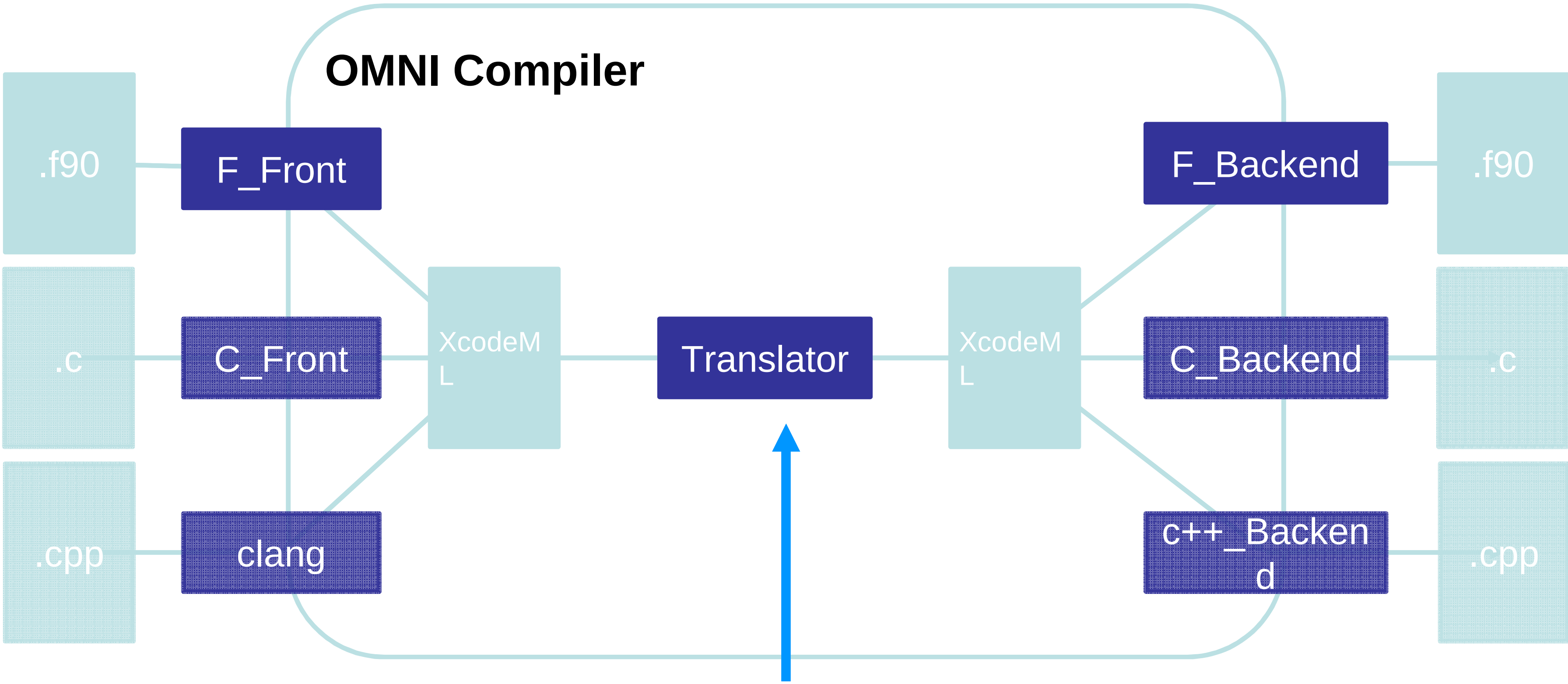
- Programming Environments Research Team from the RIKEN Advanced Institute for Computational Sciences, Kobe, Japan
- High Performance Computing System Lab, University of Tsukuba, Tsukuba

http://www.omni-compiler.org

RIKEN
Advanced Institute for
Computational Science

CLAW compiler

- Source-to-source compiler
- Based on the OMNI Compiler



14

Code manipulation

```
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (1)
  Body
    Loop 1 Body statements
    Loop 2 Body statements
```

Code generation →

```
DO i=istart,iend
  v(i) = i + 1
  v(i) = i + 2
END DO
```

Current status

- First low-level language definition
- Currently implementing the compiler for the 1st definition
- Apply code transformation to radiation standalone

Next steps

- Refine the language definition to higher level of abstraction

Any questions?

Code manipulation

```
!$claw loop-fusion
DO i=istart,iend
  v(i) = i + 1
END DO

!$claw loop-fusion
DO i=istart,iend
  v(i) = i + 2
END DO
```

AST generation →

```
PRAGMA STMT claw loop-fusion
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (1)
  Body
    Loop 1 Body statements
PRAGMA STMT claw loop-fusion
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (1)
  Body
    Loop 2 Body statements
```

Code manipulation

```
PRAGMA STMT claw loop-fusion
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (1)
  Body
    Loop 1 Body statements
PRAGMA STMT claw loop-fusion
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (2)
  Body
    Loop 2 Body statements
```

AST transform

```
DO STMT
  Induction Var
    Var (int, local, i)
  Index Range
    Lower Bound
      Var (int, local, istart)
    Upper Bound
      Var (int, local, iend)
    Step
      int const (2)
  Body
    Loop 1 Body statements
    Loop 2 Body statements
```

CLAW language definition

```
PROGRAM main
  !$claw loop-extract map(value1,value2:i) fusion group(g1)
  CALL compute(value1, value2)

  !$claw loop-fusion group(g1)
  DO i = istart, iend
    ! some computation here
    print*,'Inside loop', i
  END DO
END PROGRAM main

SUBROUTINE compute(value1, value2)
  REAL, INTENT (IN) :: value2(x:y), value2(x:y)

  DO i = istart, iend
    ! some computation with value1(i) here
  END DO
END SUBROUTINE xyz
```

CLAW language definition

```fortran
PROGRAM main
  !claw loop-extract map(value1,value2:i) fusion group(g1)
  DO i = istart, iend
    CALL compute(value1(i), value2(i))
    ! some computation here
    print*,'Inside loop', i
  END DO
END PROGRAM main
```

Possible approaches (2/4)

- Language specific annotation preprocessed by a dedicated tool

```
@domainDependant{attribute(autoDom)}
  a, b, c, d
@end domainDependant

@parallelRegion{domName(x,y), domSize(NX, NY)}
  do z=1,NZ
    c(z) = a(z) + b(z)
  end do
@end parallelRegion
```

Hybrid Fortran
Rewrite amounts of code with dedicated annotation
-Create code targeted for specific architecture
-Cannot be compiled with standard compilers
-Not standard Fortran sentinels but special annotation syntax
-Only one parallel region per subroutine

Possible approaches (3/4)

- Extending Fortran language

```
SUBROUTINE div_3D_dsl(vn, div, &
 div_coeffs, cells_subset)

<OnEdges_3D_double :: vn>
<OnCells_3D_double :: div>
<OnCellsToEdges_3D_double :: div_coeffs>
<Cells_3D_SubsetRange :: cells_subset>
<Cells_3D_Element :: cell>
<EdgesOfCells_3D_Element :: edge>

!

<cell .belongsTo. cells_subset>
<edge .belongsTo. cell>

<on cell:
  cell%div = SUM[on edge] edge%vn &
    * edge%div_coeffs;
end on cell>

END SUBROUTINE div_3D_dsl
```

ICON DSL Lite
Extend language with new types and features.
-Abstract the domain problem from the language
-Separate the concept of DSL and DPL (performance). Target DSL at the moment.
-Cannot be compiled by standard compilers