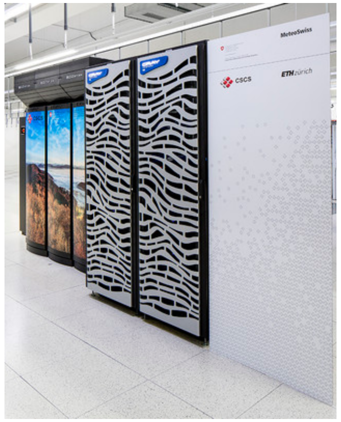
CLAW Compiler - Abstraction for Weather and Climate Models

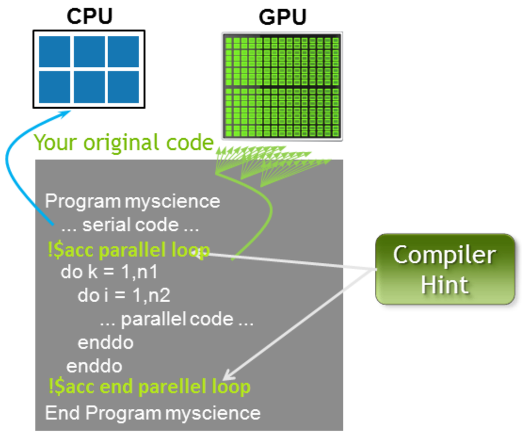
5th ENES HPC Workshop, Lecce, Italy
May 17-18, 2018
Valentin Clement
valentin.clement@env.ethz.ch

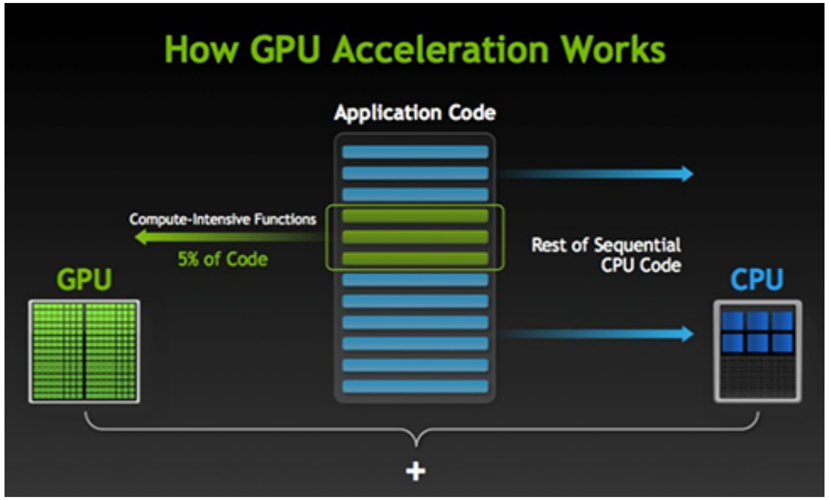
Summary

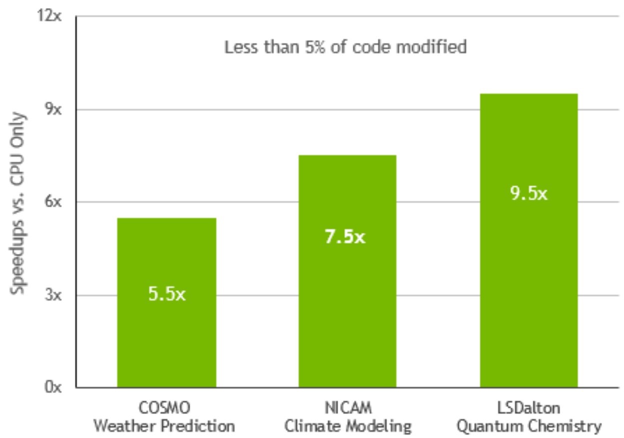
- Performance portability problem
- CLAW Single Column Abstraction
- CLAW Compiler
- Performance results

Porting COSMO to hybrid architecture with directives

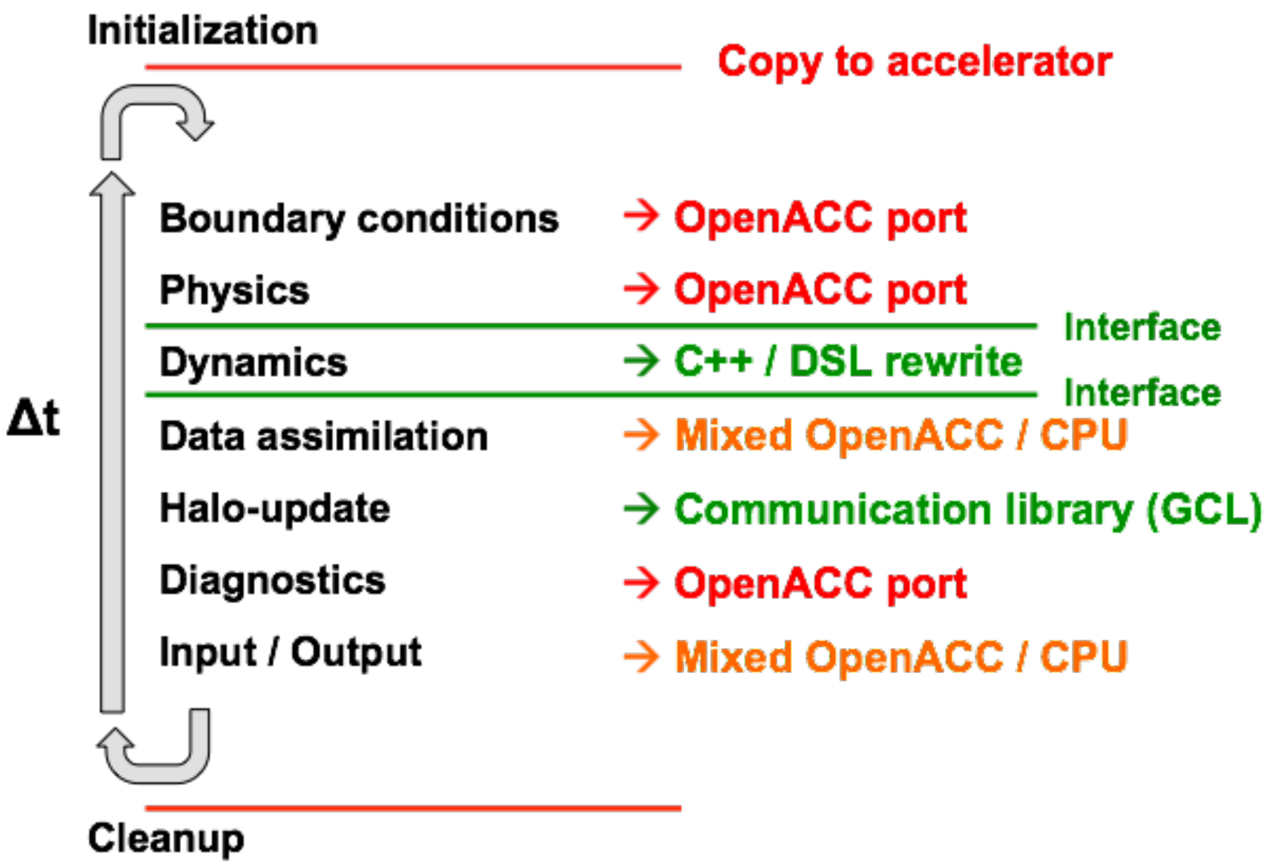




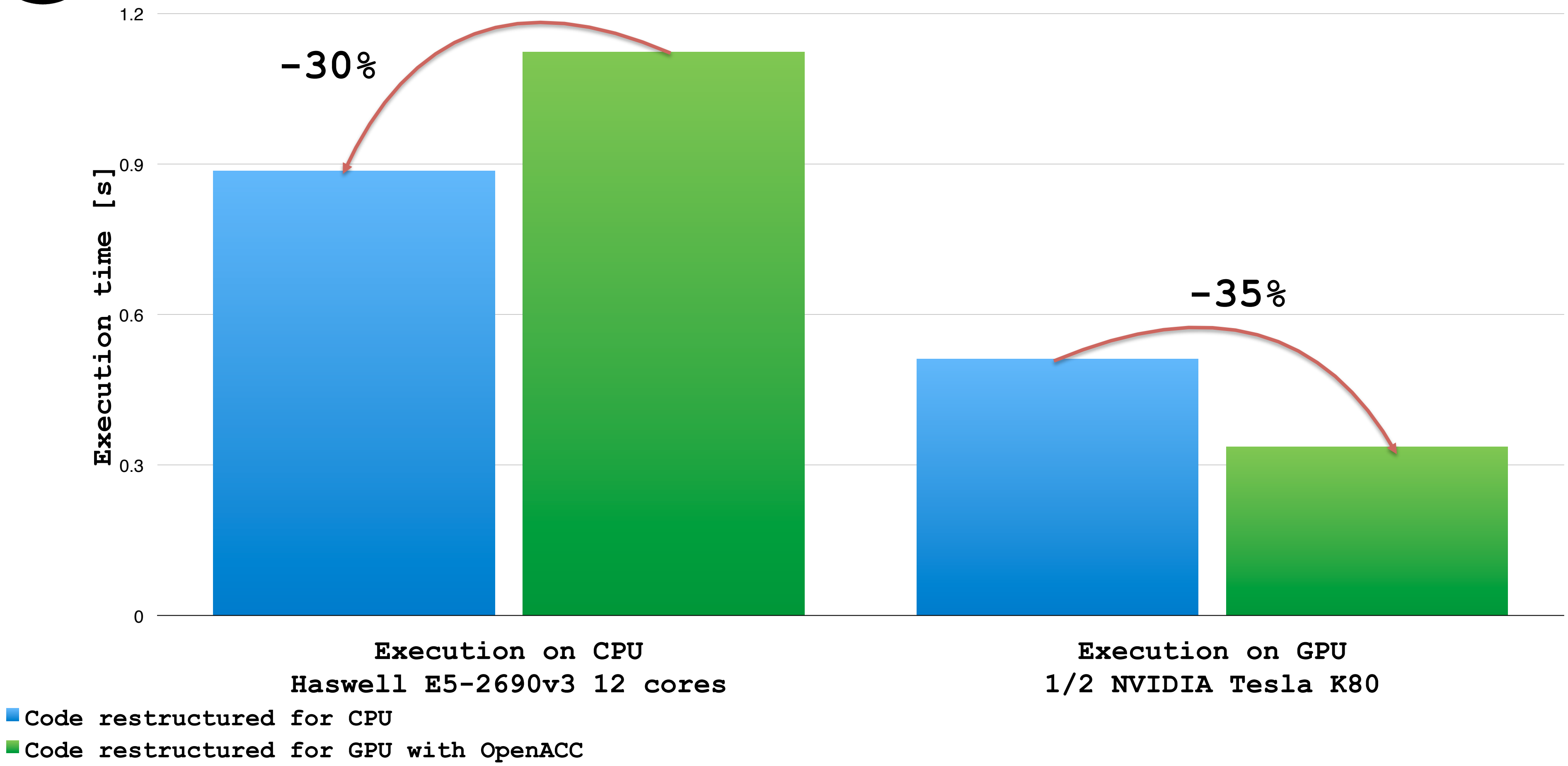




Porting COSMO to hybrid architecture



Performance portability problem - COSMO Radiation



Performance portability problem - COSMO Radiation

CPU structure

```
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
```

GPU structure

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

Performance portability problem - Keep two or more code?

```
#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 loop structure

GPU loop structure

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

What kind of code base are we dealing with?

- Massive code base (200’000 to >1mio LOC)
  - Several architecture specific optimization survive along the versions
  - Most of these code base are CPU optimized
    - Not suited for some architecture
    - Not suited for massive parallelism
  - Few or no modularity
    - Physical parameterization hardly extractable to the main model

COSMO Model - loc

Climate and local area model used by Germany, Switzerland, Italy …

Language	files	blank	comment	code
Fortran 90	173	53998	109381	211711
C/C++ Header	148	5595	11827	29888
C++	121	5050	6189	26580
Python	37	1454	1444	5764
Bourne Again Shell	17	246	381	3206
Bourne Shell	33	544	594	2349
XML	11	272	193	2143
CMake	9	103	98	793
make	1	36	27	230
CUDA	58	4	0	58
SUM:	620	68232	130684	286710

DWD ICON - loc

Global model from Germany - at least two times bigger than COSMO

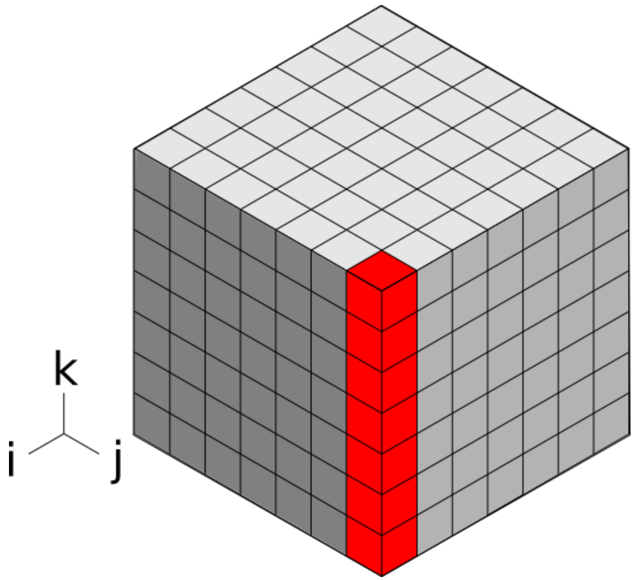
Language	files	blank	comment	code
Fortran 90	**822**	**99802**	**144962**	**447356**
C	219	43854	30991	150781
HTML	307	449	15415	94940
Fortran 77	**463**	**294**	**113285**	**64061**
Java	95	2685	4335	11605
C/C++ Header	106	2194	8359	8332
Python	43	2163	2425	7656
SUM:	2599	174509	346197	931446

Performance portability - next architecture

- What is the best loop structure/data layout for next architecture?
- Do we want to rewrite the code each time?
- Do we know exactly which architecture we will run on?
- Do we want to maintain a dedicated version for each architecture?

?

CLAW Single Column Abstraction (SCA)



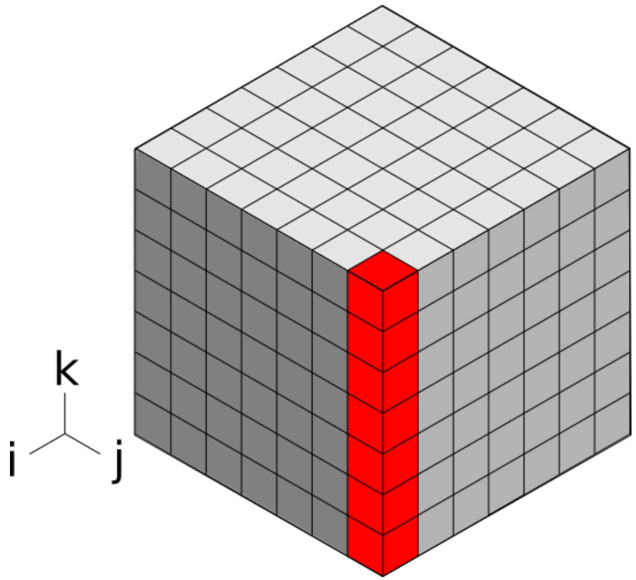
Targets physical parameterization

- Remove independent horizontal dimension
  - Remove do statements over horizontal
  - Demote arrays

Separation of concerns

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

RRTMGP Example - A nice modular code CPU structured



F2003 radiation code

- From Robert Pincus and al. from AER University of Colorado
- Compute intensive part are well located in “kernel” module.
- Code is non-the-less CPU structured with horizontal loop as the inner most in every iteration.

RRTMGP Example - original code - CPU structured

Loop over spectral bands

Loop over vertical dimension

Loop over horizontal dimension

```
SUBROUTINE sw_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 sw_solver
```

RRTMGP Example - Single Column Abstraction

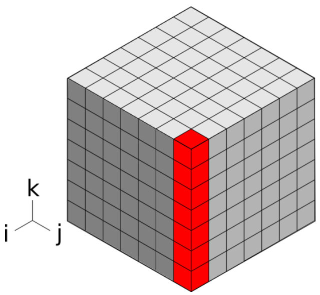
Only dependency on these iteration spaces

```
SUBROUTINE sw_solver(ngpt, nlay, tau, …)
  ! DECL: Fields don't have the horizontal dimension (demotion)
  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 sw_solver
```

RRTMGP Example - CLAW code in subroutine

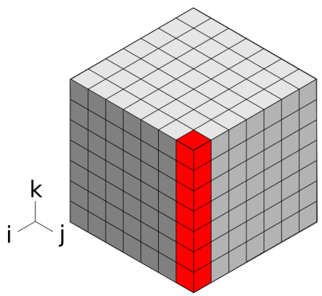
Algorithm for one column only

```
SUBROUTINE sw_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 sw_solver
```



Dependency on the vertical dimension only

RRTMGP Example - CLAW at call site



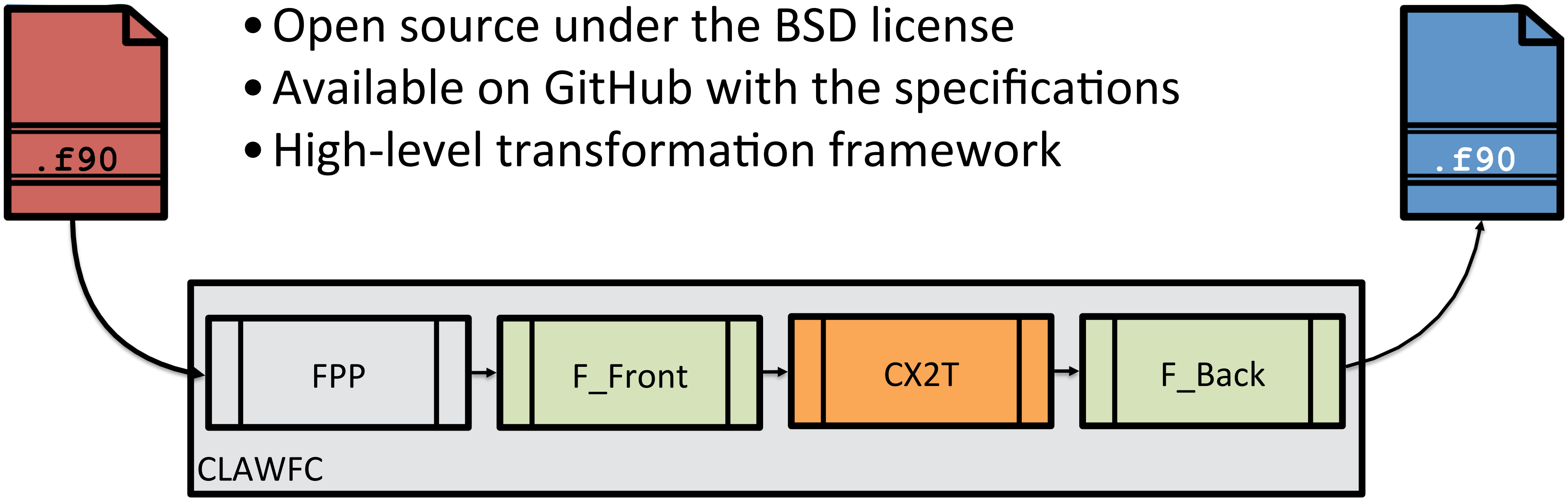

```fortran
! Location in the model where the physical parameterization is
! plugged

!$claw parallelize forward
DO icol = 1, ncol
   CALL sw_solver(ngpt, nlay, tau(icol,:,:), …)
END DO
```

Fully working code if compiled with a standard compiler

Only standard Fortran

What is the CLAW Compiler?

- Source-to-source translator
- Based on the OMNI Compiler Project
- Fortran 2008
- Open source under the BSD license
- Available on GitHub with the specifications
- High-level transformation framework



OMNI Compiler Project

F_Front F_Back

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

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

Development team

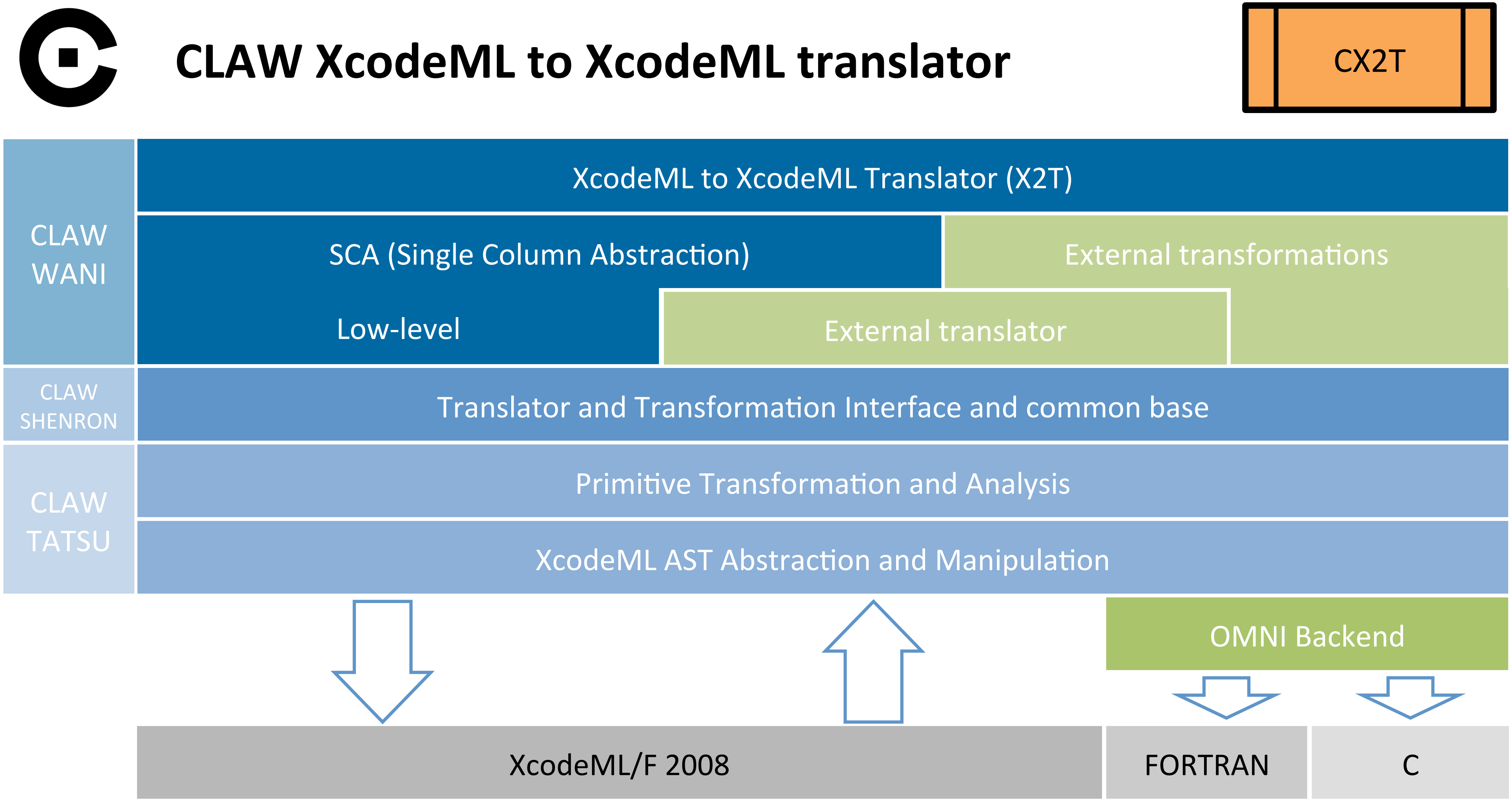
- Programming Environments Research Team from the RIKEN Center for Computational Sciences (R-CCS), Kobe, Japan
- High Performance Computing System Lab, University of Tsukuba, Tsukuba
- CLAW Project is actively collaborating in this project

The Fortran front-end and the backends are used in the CLAW Compiler

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

RIKEN R-CCS RIKEN Center for Computational Science

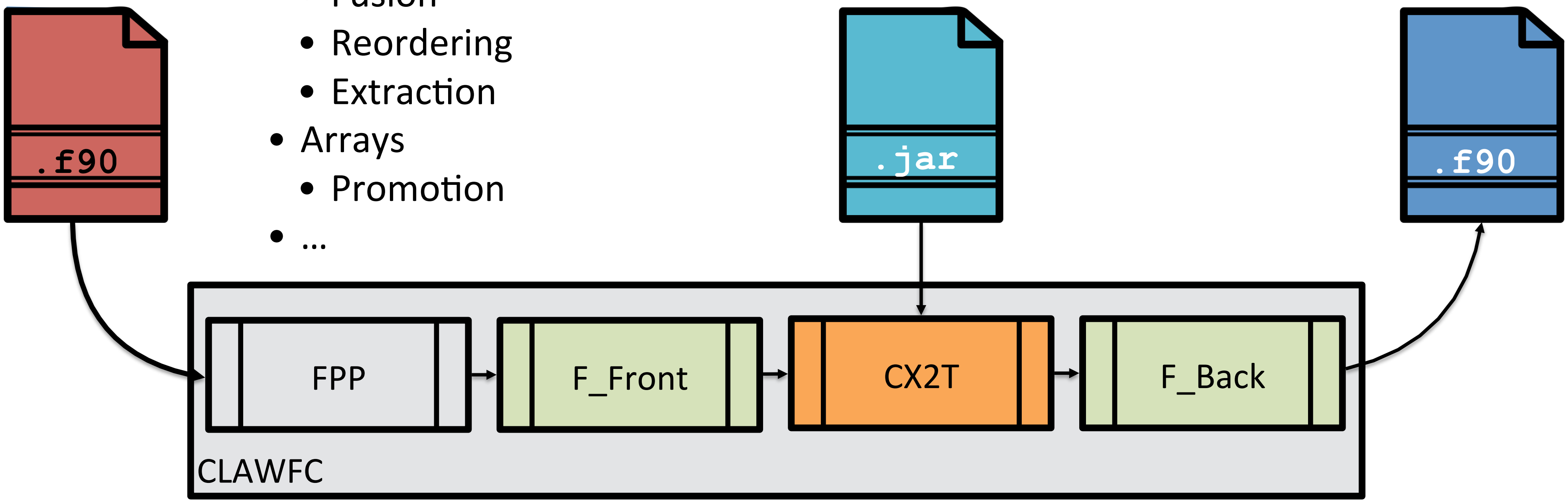
CLAW XcodeML to XcodeML translator



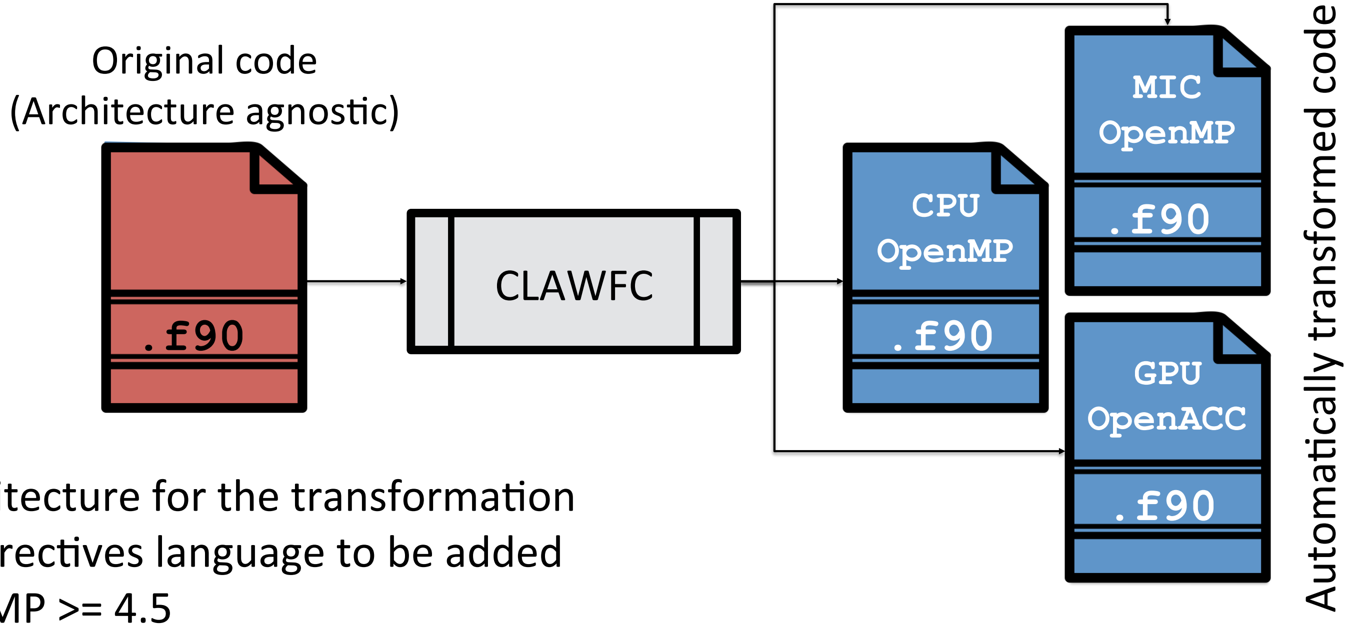
CLAW CX2T - External transformation

Easy integration of new transformation build on top of “building blocks”

- Primitive transformation
  - Loops
    - Fusion
    - Reordering
    - Extraction
  - Arrays
    - Promotion
  - …



RRTMGP Example - CLAW transformation



- A single source code
- Specify a target architecture for the transformation
- Specify a compiler directives language to be added
  - OpenACC or OpenMP >= 4.5

```
clawfc --directive=openacc --target=gpu -o mo_sw_solver.acc.f90 mo_sw_solver.f90
```

```
clawfc --directive=openmp --target=cpu -o mo_sw_solver.omp.f90 mo_sw_solver.f90
```

```
clawfc --directive=openmp --target=mic -o mo_sw_solver.mic.f90 mo_sw_solver.f90
```

CLAW SCA to target specific code - recipe

- Data analysis for promotion and generation of directive
  - Potentially collapsing loops
  - Generate data transfer if wanted
- Adapt data layout
  - Promotion of scalar and arrays to fit model dimensions
  - Detect unsupported statements for OpenACC
- Insertion of do statements to iterate of new dimensions
- Insertion of directives (OpenMP/OpenACC)

CLAW Compiler has various options - example for GPU

- **Local array strategy** for GPU transformation
  - **Private** - issue a copy of the array for each “thread”
  - **Promote** - promote the array and keep a unique copy for all the “thread”

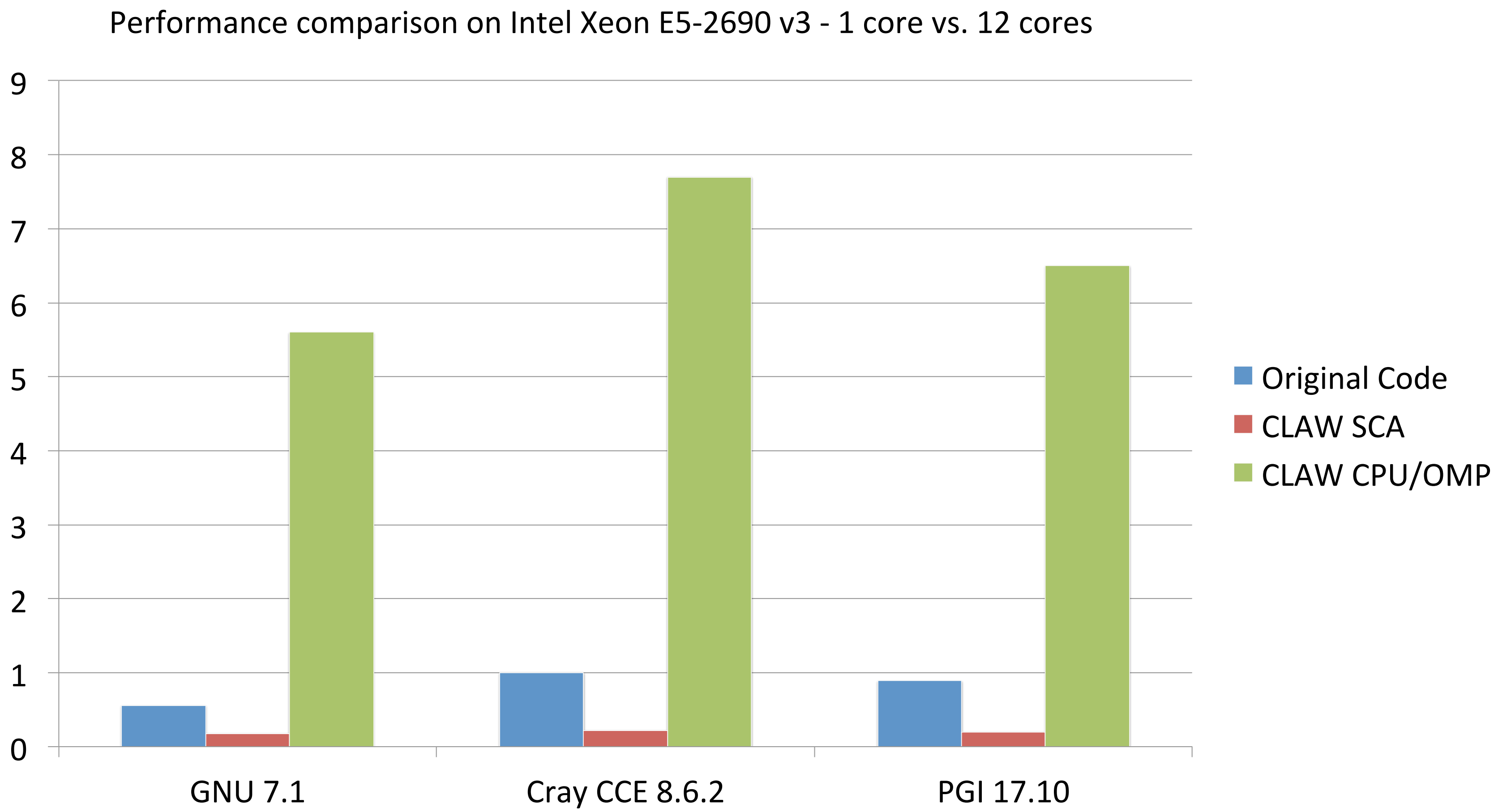
- **Data movement strategy** for GPU transformation
  - **Present** - assume that data are present on the device, no data transfer
  - **Kernel** - data movement is generated for each kernel
  - **None** - no data region generated

- **Collapse strategy** - true/false

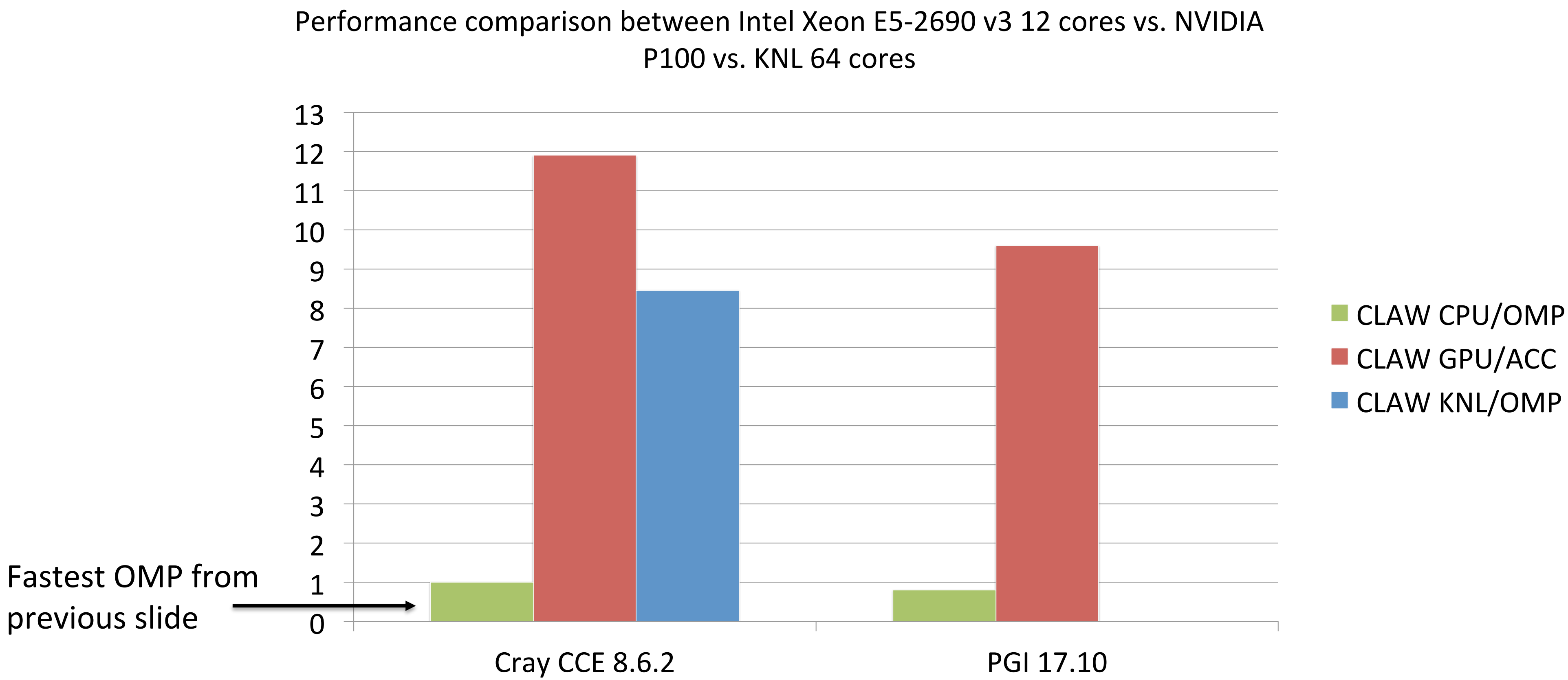
RRTMGP Example - CLAW target=gpu directive=openacc

```
SUBROUTINE sw_solver(ngpt, nlay, tau, …)
! DECL: Fields promoted accordingly to usage
!$acc data present(…)
!$acc parallel
!$acc loop gang vector private(…) collapse(2)
DO icol = 1 , ncol , 1
  DO igpt = 1 , ngpt , 1
    !$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 sw_solver
```

RRTMGP Example - Speedup on CPU



RRTMGP Example - Speedup CPU vs. GPU vs. KNL



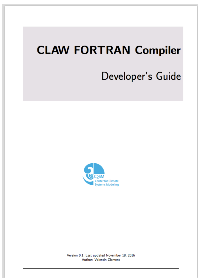
PASC ENIAC Project (2017-2020)

- Enabling ICON model on heterogenous architecture
  - Port to OpenACC
  - GridTools for stencil computation (DyCore)
  - Looking at performance portability in Fortran code
    - Enhance CLAW Compiler capabilities
    - Apply SCA on some physical parameterization
    - Enhance transformation for x86, XeonPhi and GPUs

CLAW Compiler & Directives - Resources

`https://claw-project.github.io`

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



CLAW Compiler developer's guide

Summary

- Single source code with high-level of abstraction
- Domain scientist can focus on their problem
- Little to no change in current code
- Standard Fortran
- Open source project
- CLAW is easily extensible to new architecture or new transformation

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 zürich

CLAW

valentin.clement@env.ethz.ch

https://claw-project.github.io
https://github.com/omni-compiler