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
source-to-source translation for performance portability

XcalableMP Workshop, Akihabara, Tokyo, Japan
October 31, 2017

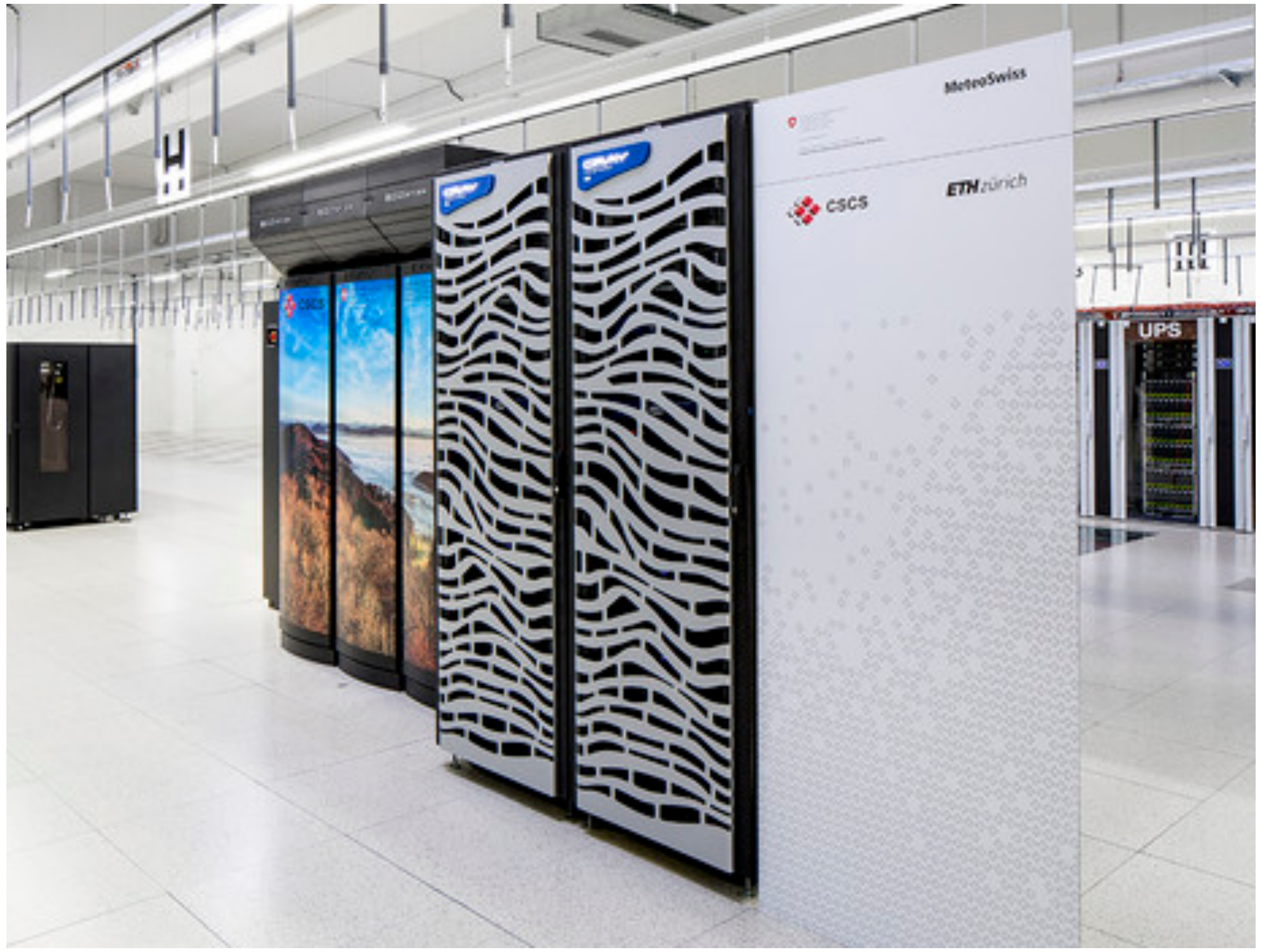
Valentin Clement
valentin.clement@env.ethz.ch

Summary

- Performance portability problem in COSMO
  - Single source code
- Portability
  - Performance
  - Portability of code
- CLAW FORTRAN Compiler
  - Single column abstraction
  - Future project

C2SM
Center for Climate
Systems Modeling

GPU machine in Switzerland - Piz Kesch (MeteoSwiss)



Production + R&D

Each rack is composed of 12 Compute nodes:

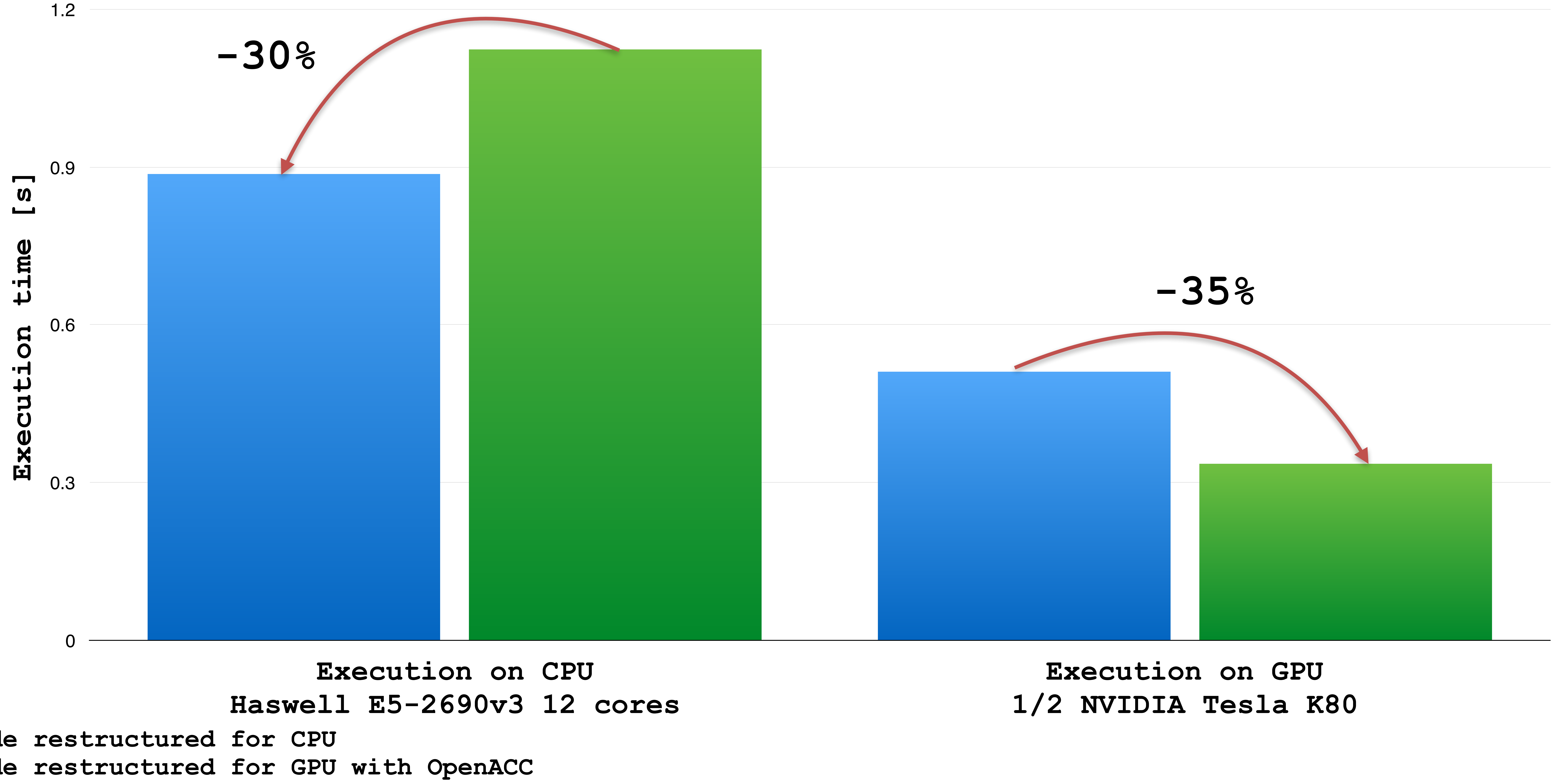
- 2x Intel Haswell E5-2690v3
  2.6 GHz 12-core CPUs (total of 24 CPUs)
- 256 GB 2133 Mhz DDR4 (total of 3TB)
- 8x Dual NVIDIA TESLA K80 GPU (total of 96 cards - 192 GPUs)

GPU machine in Switzerland - Piz Daint (CSCS)

Cray XC40/XC50

- Intel® Xeon® E5-2690 v3 2.60GHz, 12 cores, 64GB RAM
- NVIDIA Tesla P100

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
```

How to keep a single source code for everyone

- Massive code base (200’000 to >1mio LOC)
  - Several architecture specific optimization survive
  - Most of these code base are CPU optimized
    - Not suited for next generation architecture
    - Not suited for massive parallelism
    - Few or no modularity

What kind of code base are we dealing with?

- Global/local area weather forecast model
  - >10 around the world
  - Monster FORTRAN 77-2008 “monolithic” code
    - Without much modularity

- So far we investigate:
  - COSMO (Local area model consortium) - Several institution
  - ICON - DWD (German Weather Agency) - Will replace COSMO
  - IFS Current Cycle + FVM - ECMWF - Member state usage

What kind of code base are we dealing with?

Example of three code base we investigated so far:

- COSMO
- ICON
- IFS

COSMO Mode - loc

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

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

ECMWF IFS - loc

ECMWF

European Centre for Medium-range weather forecasts - Global Model

Language	files	blank	comment	code
Fortran 90	4003	201338	300427	737289
C/C++ Header	480	846	0	13041
Perl	1	89	13	240
SUM:	4484	202273	300440	750570

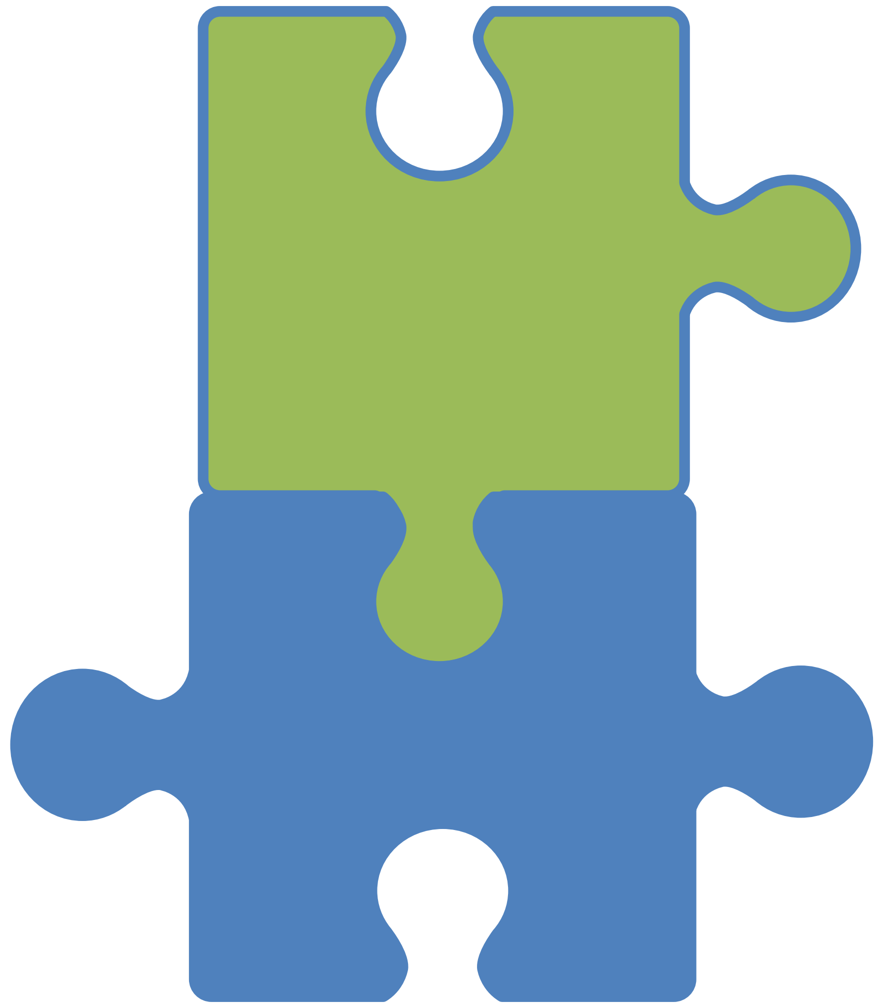
*only source without external modules

DWD ICON - loc

New German Global model with option to used it as local area model

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

Portability of code



- Current code base
  - Optimized for one architecture
    - Often with old optimizations still in place
  - Not really modular
  - Global fields injected everywhere

- Future?
  - Modular standalone package that can be plug
    - In different model
    - For different architecture
  - Abstract data layout where it can be done
  - Abstract model specific data, everything passed as arguments.

Performance portability - current code

Loop structure better for CPUs

```
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
```

```
ilev -> dependency
icol -> no dependency
```

```
nlay size ~= 100
ncol size >= 10'000
```

Performance portability - current code

Sometimes using nproma block optimization for better vectorization

```
 !$omp parallel default(shared)
 DO igpt = 1, ngptot, nproma
   CALL physical_parameteriztaion(…) ! Code from previous silde
 END DO
!$omp end parallel
```

Performance portability - GPU structured code

Loop structure better for GPUs

```
DO icol = 1, ncol
  DO ilev = 1, nlay
    tau_loc(icol,ilev) = max(tau(icol,ilev,igpt) …
    trans(icol,ilev) = exp(-tau_loc(icol,ilev))
  END DO
  DO ilev = nlay, 1, -1
    radn_dn(icol,ilev,igpt) = trans(icol,ilev) * radn_dn(icol,ilev+1,igpt) …
  END DO
  DO ilev = 2, nlay + 1
    radn_up(icol,ilev,igpt) = trans(icol,ilev-1) * radn_up(icol,ilev-1,igpt)
  END DO
END DO
```

```
ilev -> dependency
icol -> no dependency
```

```
nlay size ~= 100
ncol size >= 10’000
```

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?

?

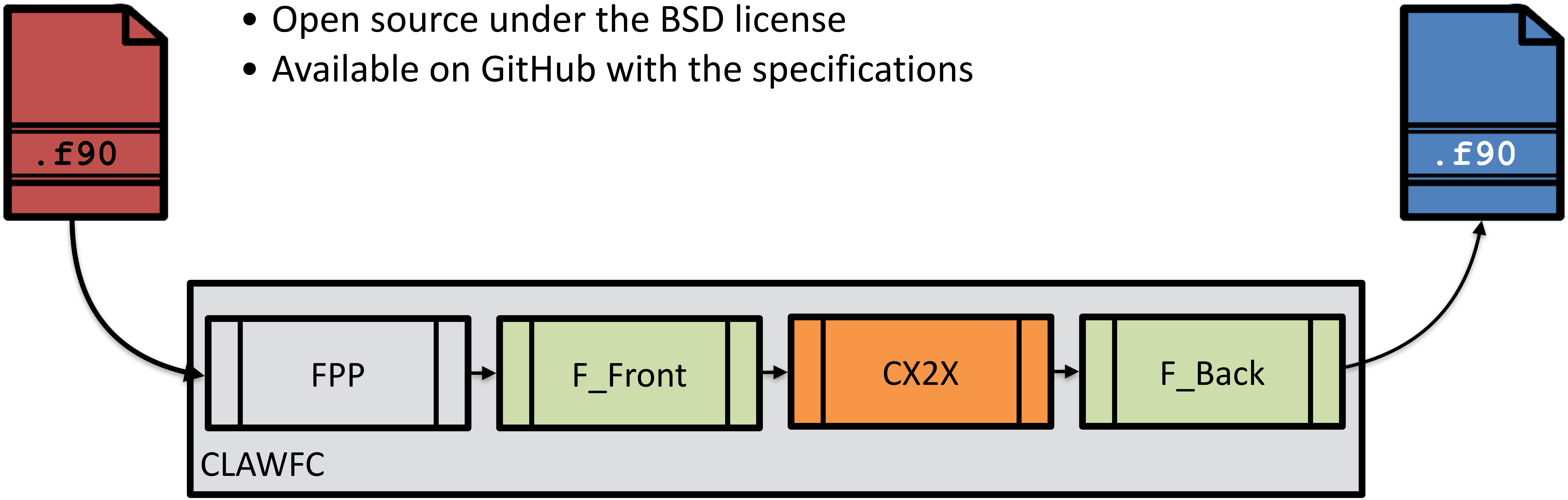
What is the CLAW FORTRAN Compiler?

- Source-to-source translation for FORTRAN code
  - Based on XcodeML/F IR
  - Using OMNI Compiler front-end and back-end
    - Contribution to it via GitHub

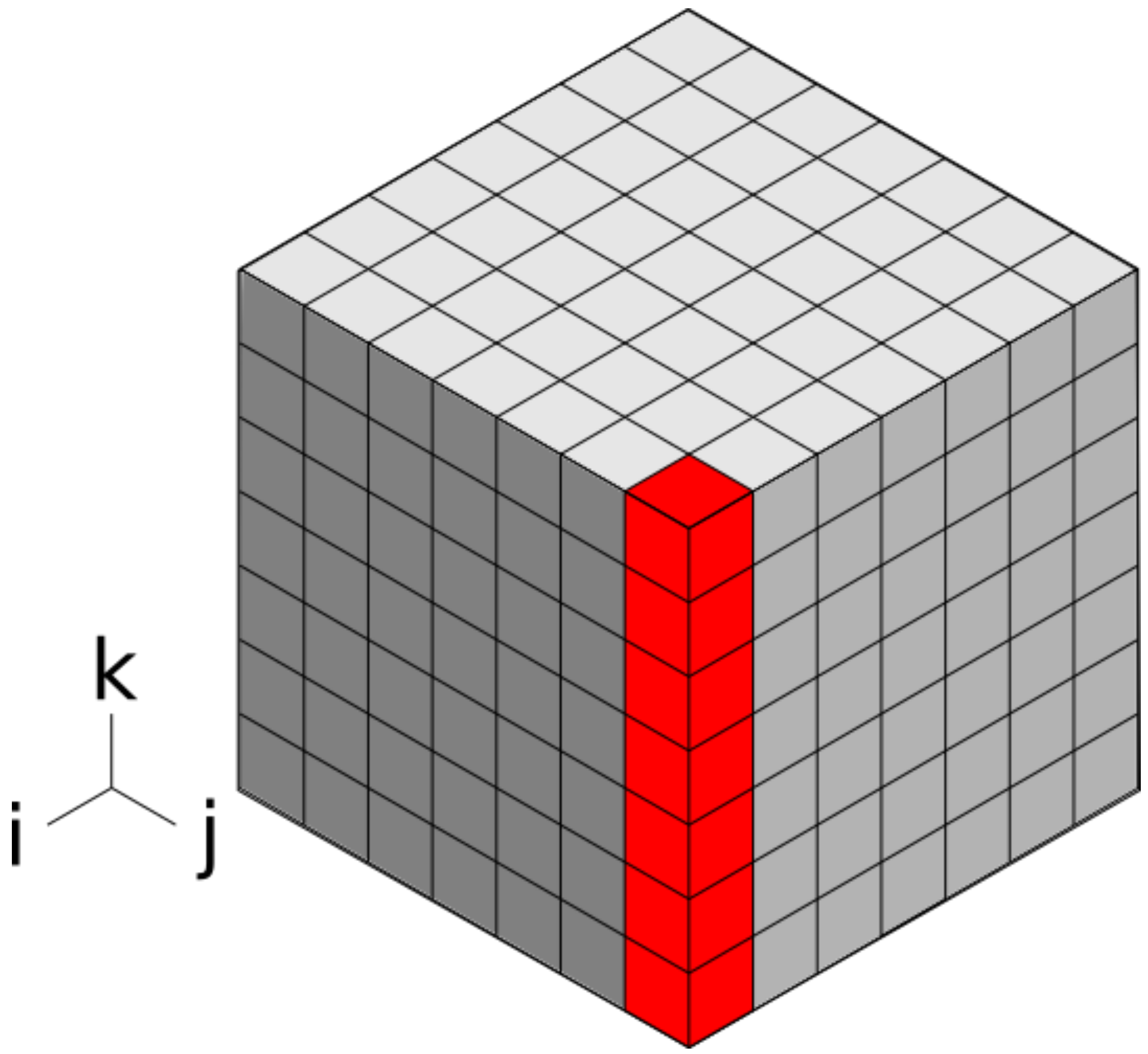
  - Transformation of AST
    - Different transformation applied based on target
      - Promotion of scalar and arrays
      - Insertion of iteration
      - Insertion of OpenACC and OpenMP directives

CLAW FORTRAN Compiler under the hood

- 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



Single column abstraction



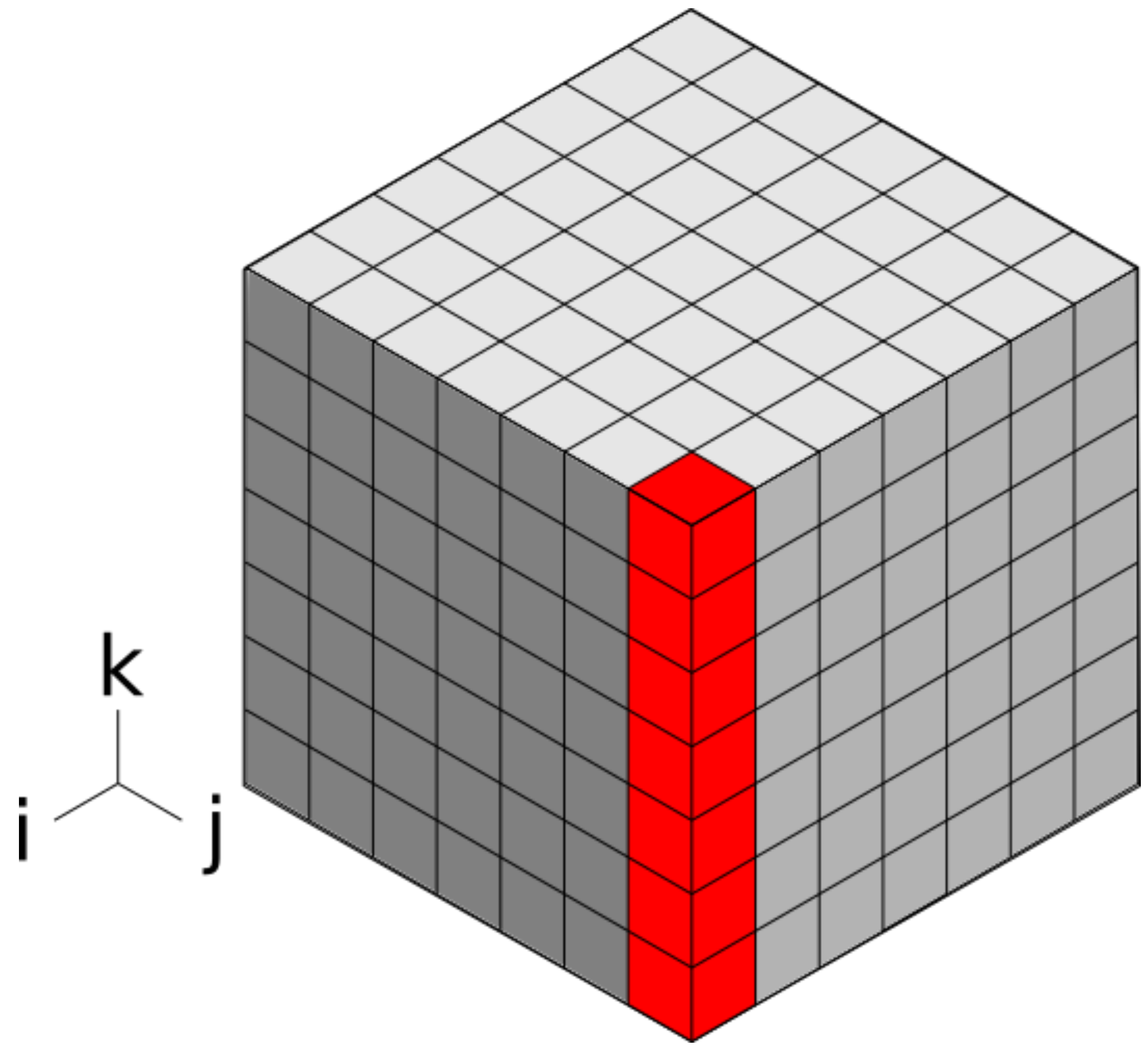
Separation of concerns

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

Achieve modularity

- Standalone physical parameter
- Modular from model specificity

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 intervals

Loop over vertical dimension

Loop over horizontal dimension

```
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 - Single column abstraction

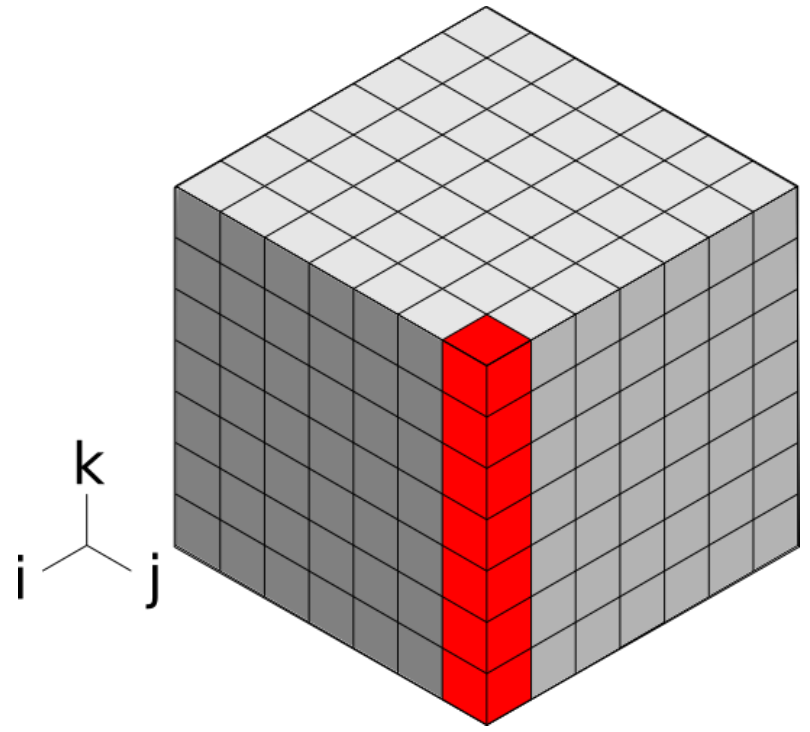
Only dependency on these iteration spaces

```
SUBROUTINE lw_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 lw_solver
```

RRTMGP Example - CLAW code

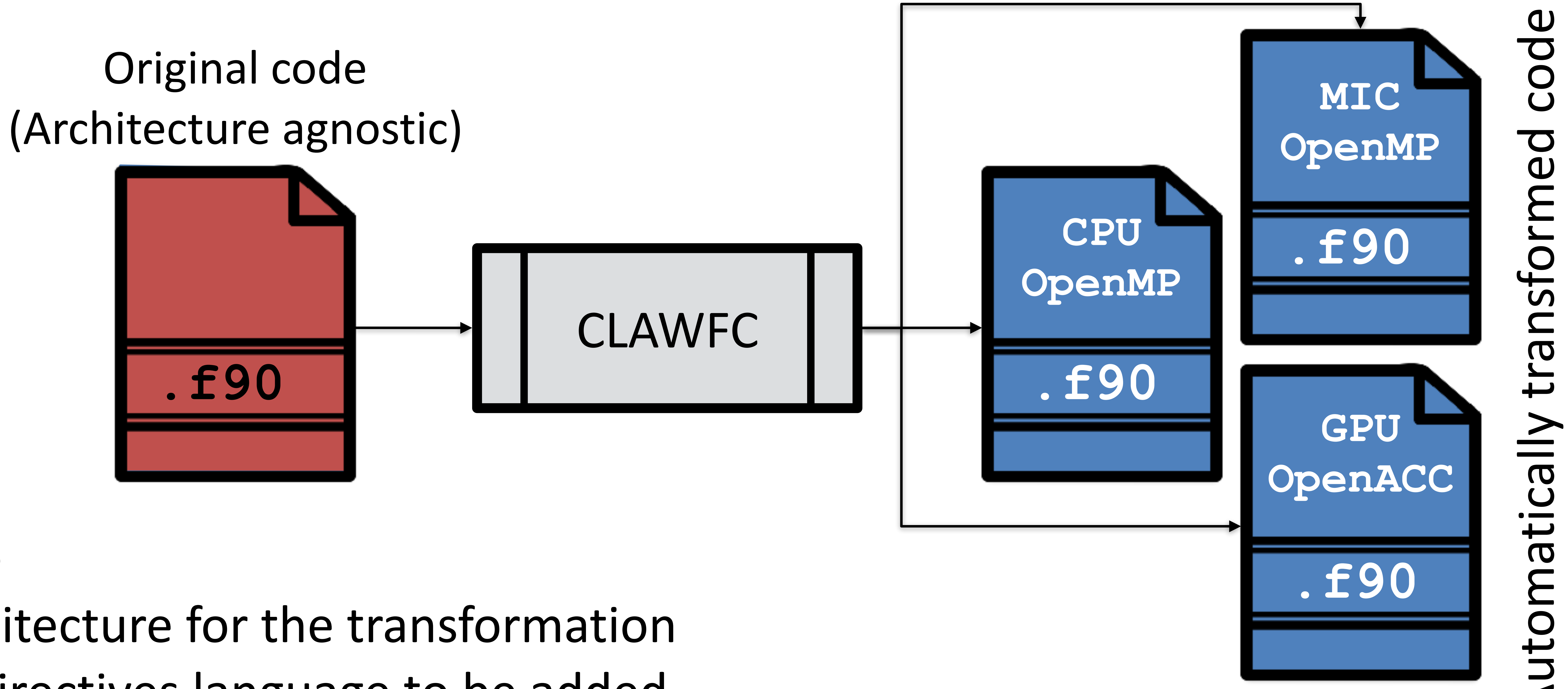
Algorithm for one column only

```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
 !$claw parallelize ! model dimension info located in config
 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 - CLAW 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 - One column - OpenACC - local array strategy

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

RRTMGP Example - GPU w/ OpenACC

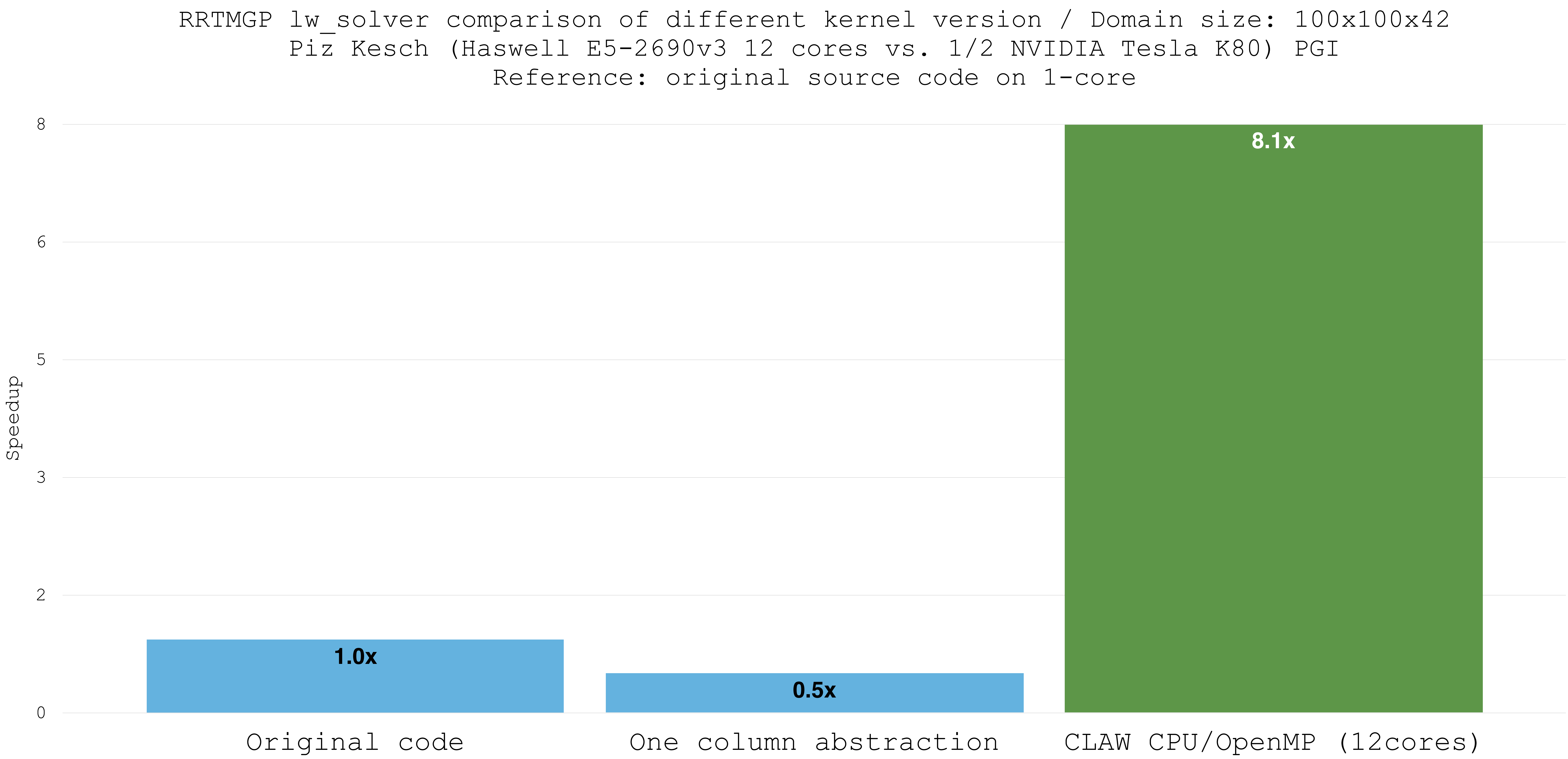
```
SUBROUTINE lw_solver(ngpt, nlay, tau, …)
! DECL: Fields promoted accordingly to usage
!$acc data present(…)
!$acc parallel
!$acc loop gang vector private(…) collapse(2)
DO icol = 1 , ncol , 1
    DO igpt = 1 , ngpt , 1
    !$acc loop seq
    DO ilev = 1 , nlay , 1
      tau_loc(ilev) = max(tau(icol,ilev,igpt)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

CLAW - One column - OpenACC - local array strategy

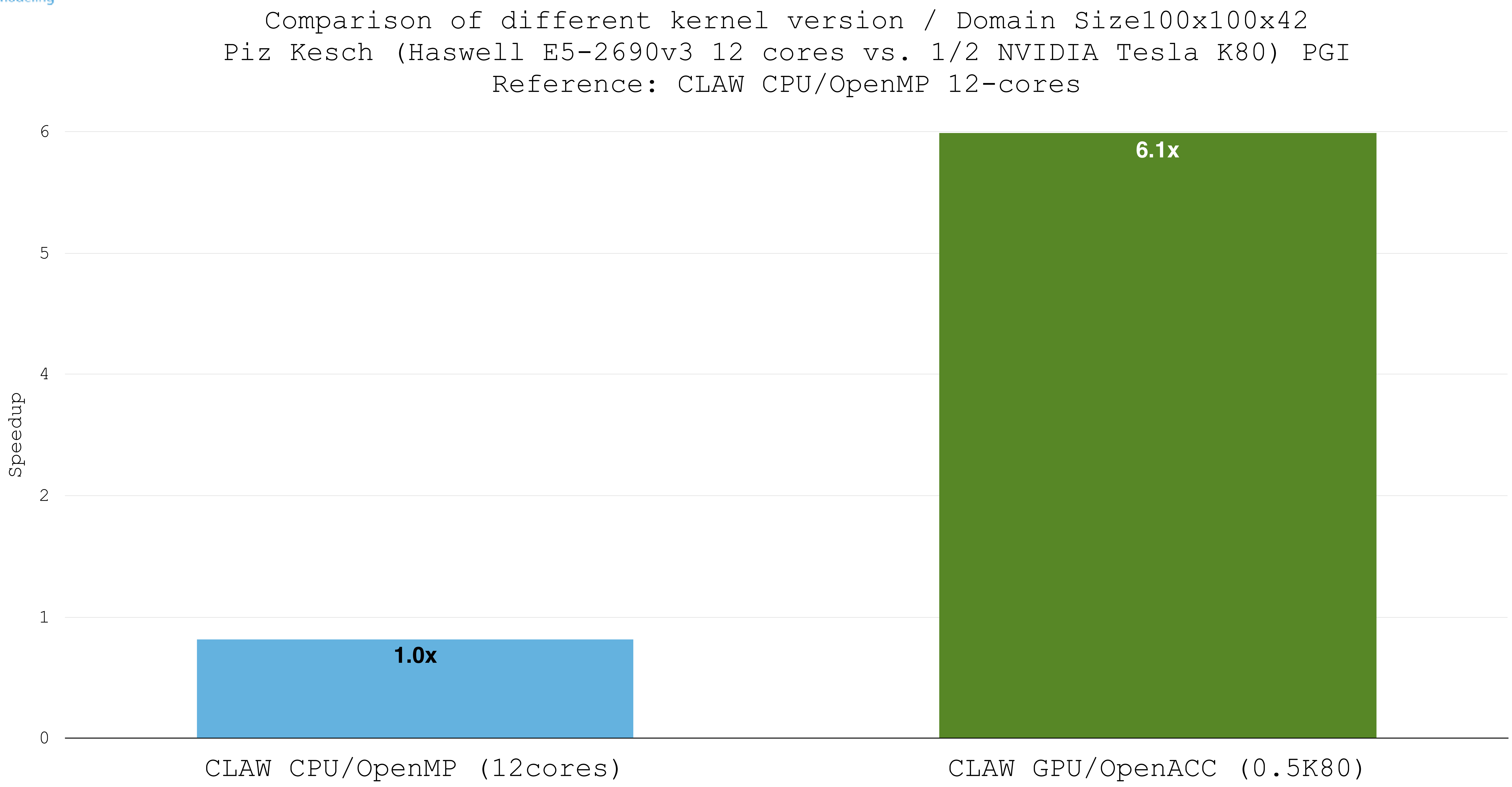
Example of different strategy easy to test with an automatize workflow:

1. Privatize local arrays
   - Make local arrays private (unsupported for allocatable arrays)
2. Promote arrays
   - Reduce allocation overhead

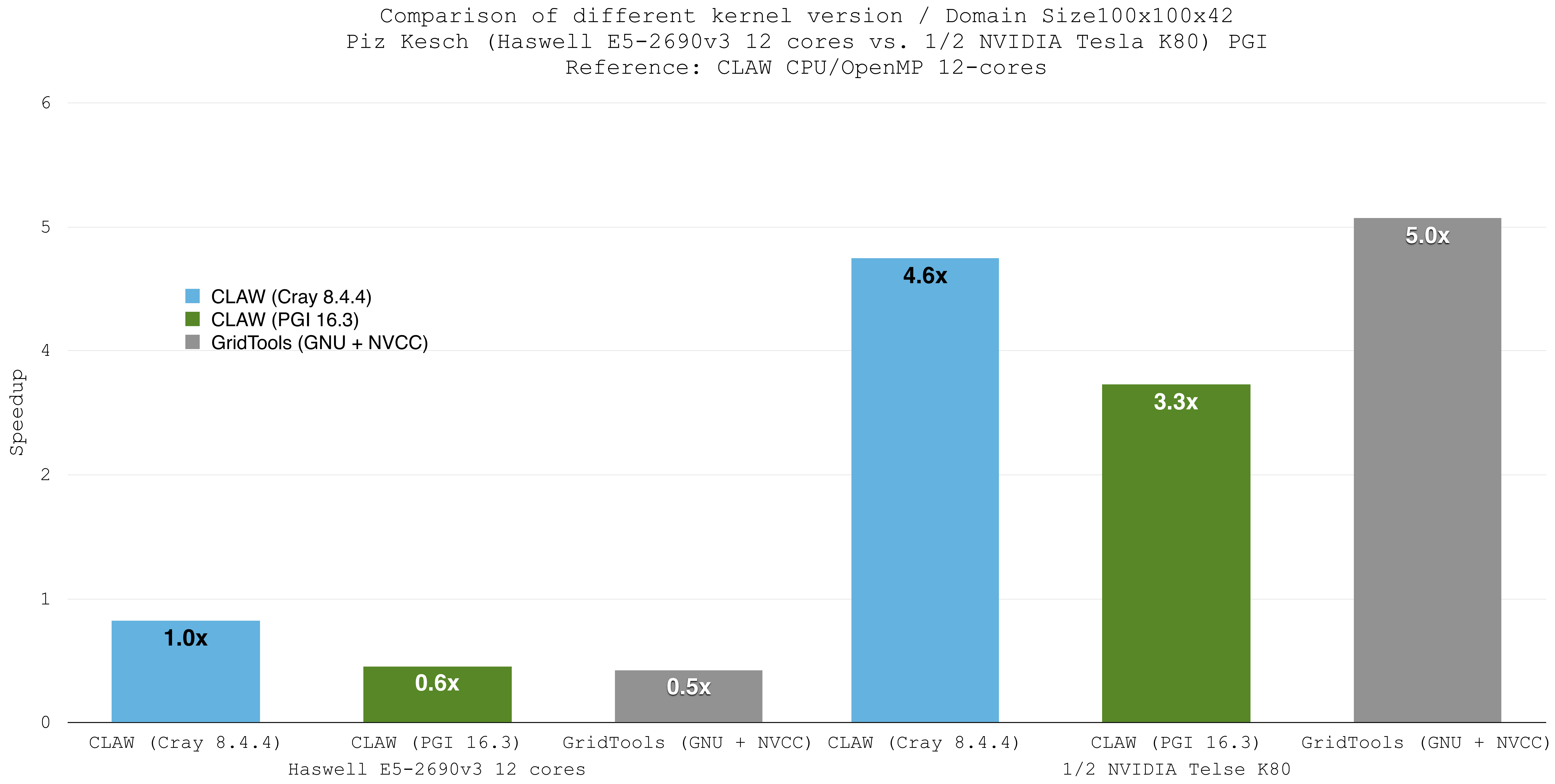
RRTMGP lw_solver - Original vs. CLAW CPU/OpenMP



RRTMGP lw_solver - CLAW CPU vs. CLAW GPU



RRTMGP lw_solver - CLAW vs. GridTools

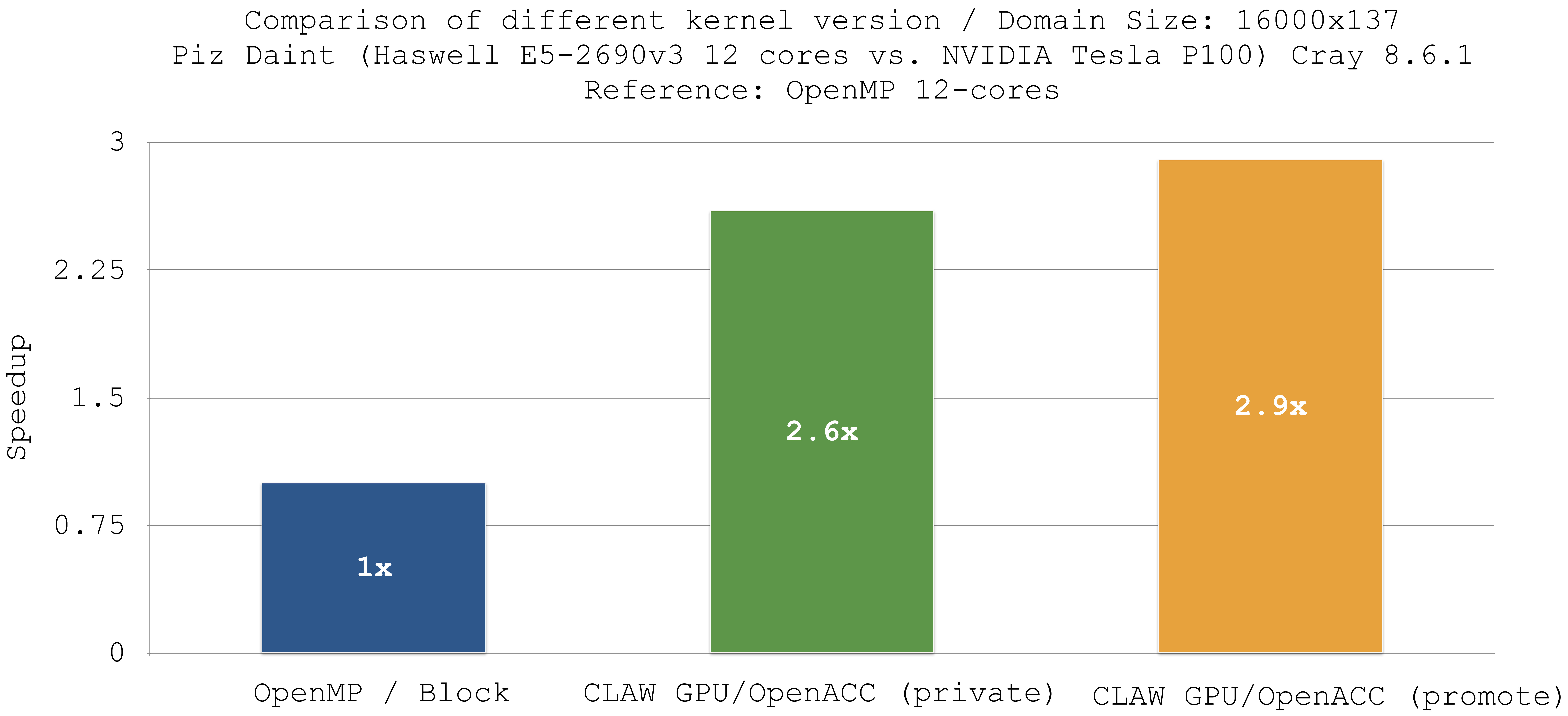


IFS-CloudSC - one column version

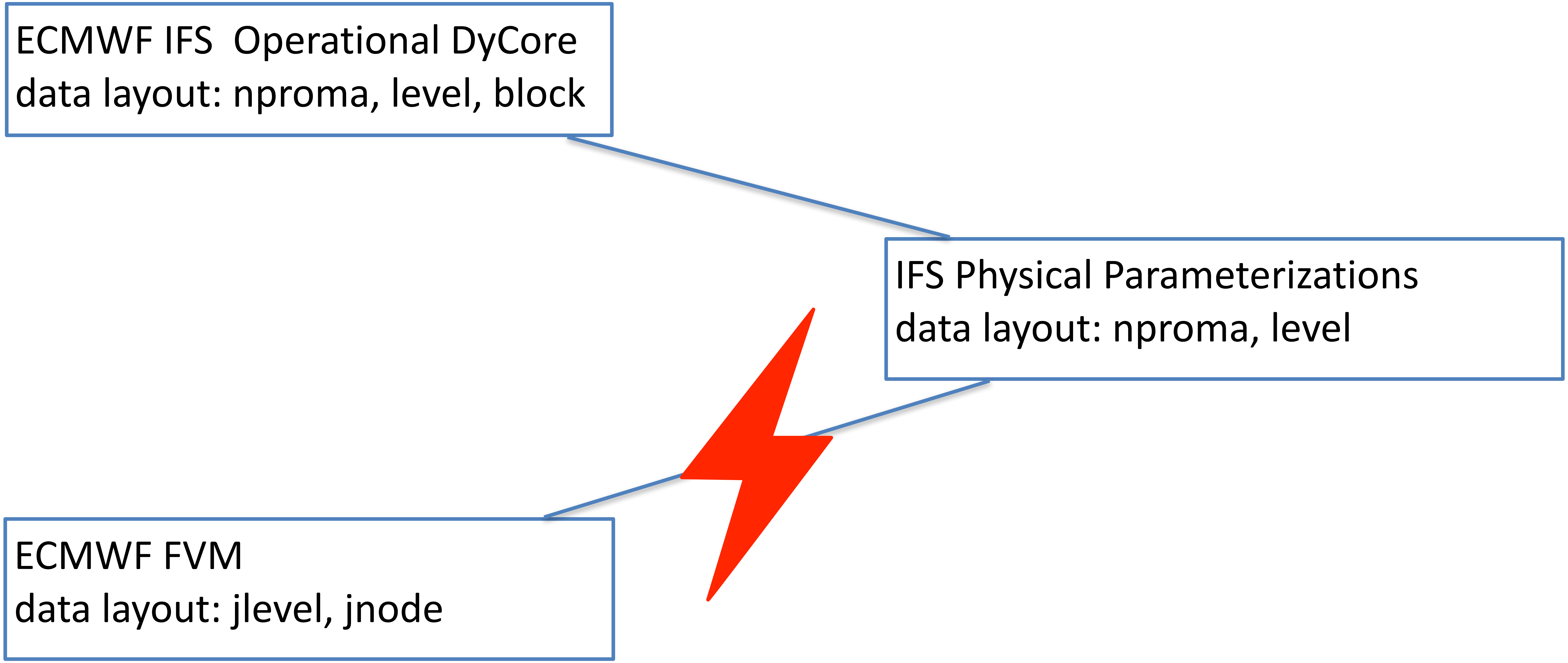
CloudMircophysics Scheme

- Take less than a day to create a one column version
- Can play with it and apply different strategy
  - OpenACC privatization of local arrays
  - OpenACC promotion of local arrays

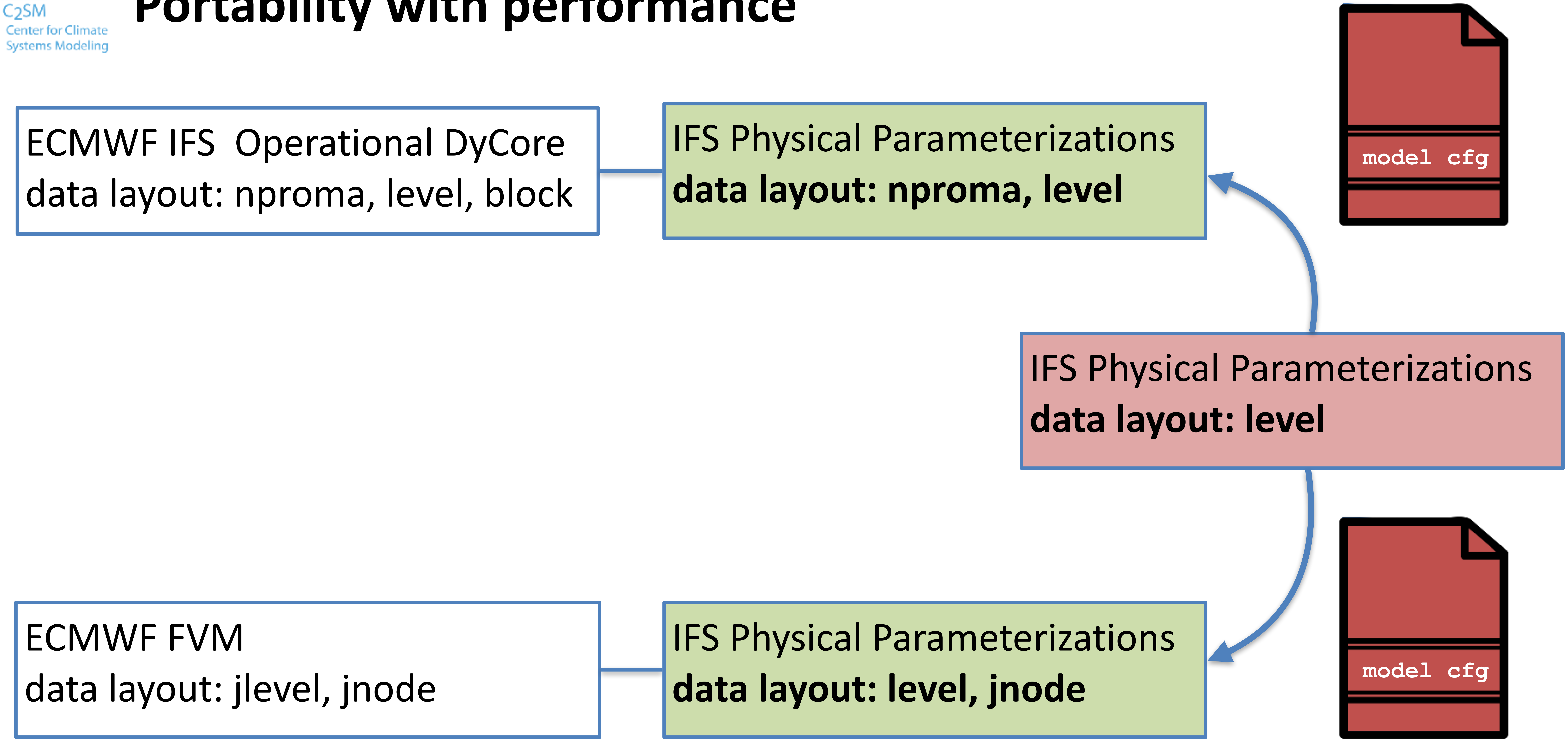
IFS-CloudSC - one column version: early results



Portability with performance



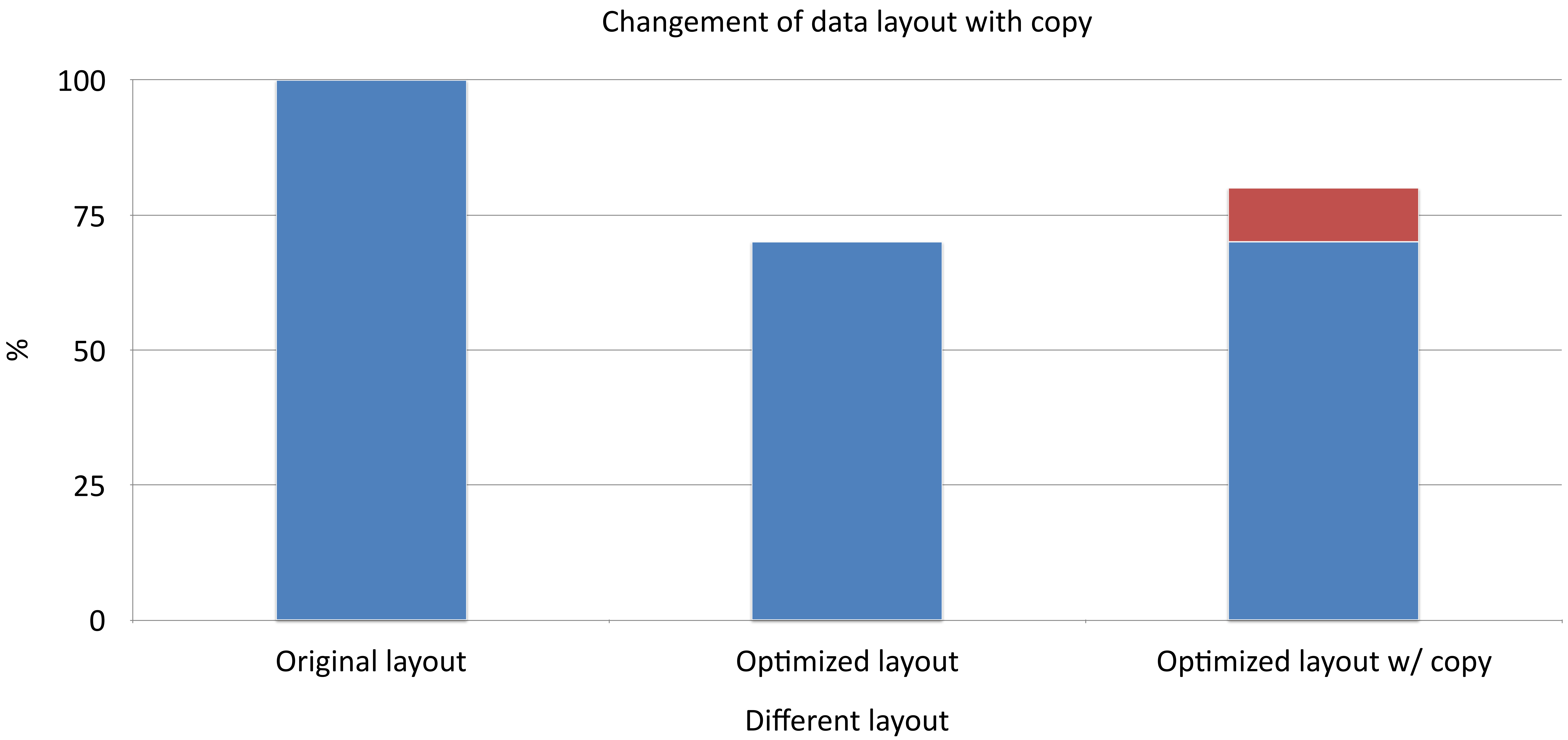
Portability with performance



Portability with performance

- Transformed code might be the same
  - More parallelism on outer loop for GPU is better
    - Independent from data layout
- Might have to introduce copy
  - Spending small time copying to new data layout might worth in overall performance

Portability with performance



CLAW collaboration with the OMNI Compiler Project

- Only viable FORTRAN source-to-source framework
  - Only one currently maintained
  - Very responsive people
  - Accept Pull Requests
    - 61 issues opened as today -> 51 closed
    - 61 PR as today -> 57 closed

Open source takes some effort but it is rewarding!

ENIAC Project (2017-2020)

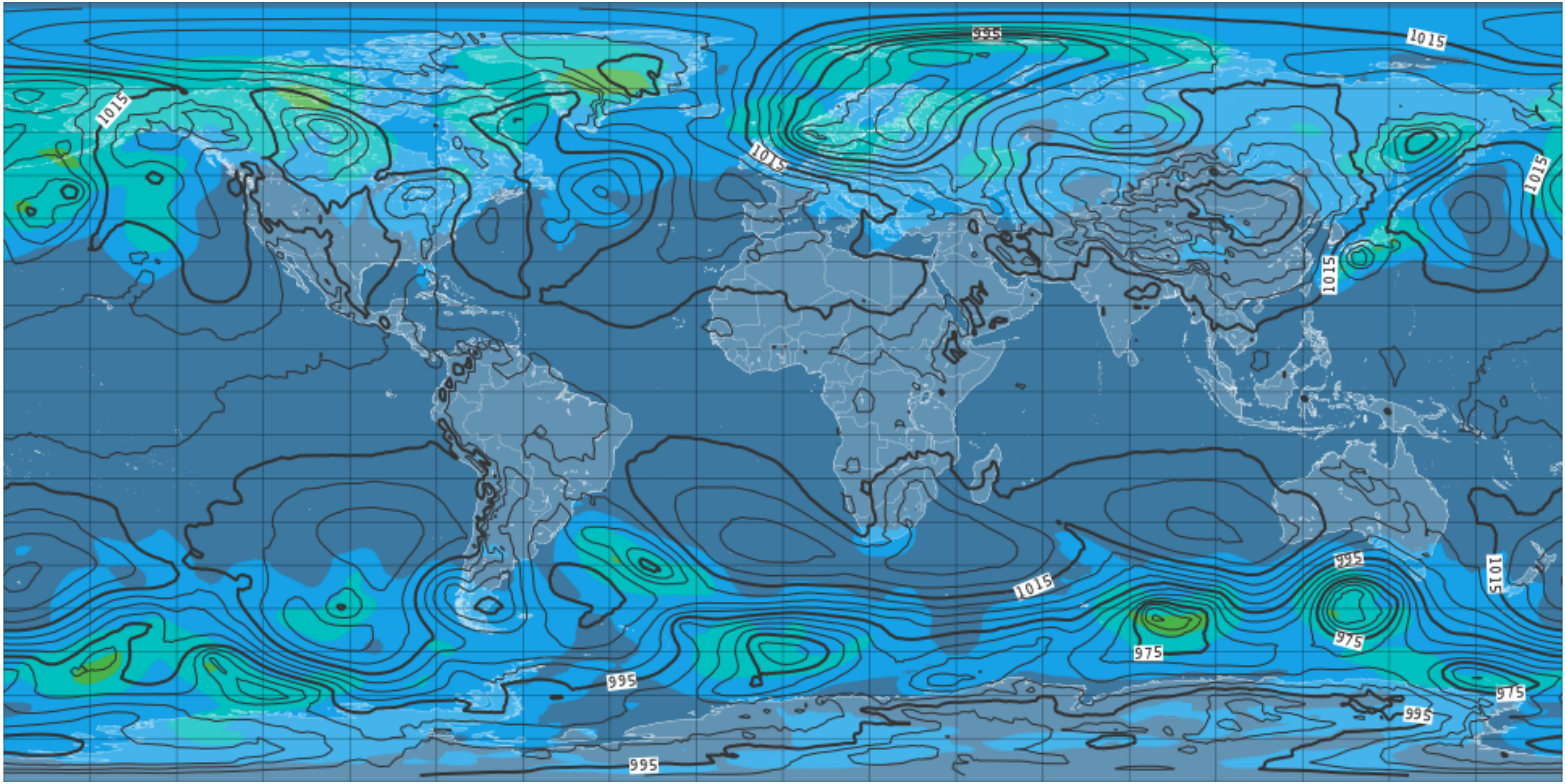
- Enabling ICON model on heterogenous architecture
  - Port to OpenACC
  - GridTools for stencil computation (DyCore)
  - Looking at performance portability in FORTRAN code
    - Enhance CLAW FORTRAN Compiler capabilities
    - Move physical parameterization to single column
      - Getting more numbers :-)
    - Apply transformation for x86, XeonPhi and GPUs

EuroExa Project

- Machine will be hosted at STFC in UK
  - ARM processor node
  - Loaded with FPGA

- ECMWF will investigate single column abstraction
  - Specific transformation for ARM processor
  - Mabye automatic offloading to FPGA (FORTRAN to C translation)

Only a problem of MeteoSwiss?



C2SM Center for Climate Systems Modeling

Possible future collaboration

Hartree Centre
Science & Technology Facilities Council

Met Office

ECMWF

C2SM Center for Climate Systems Modeling

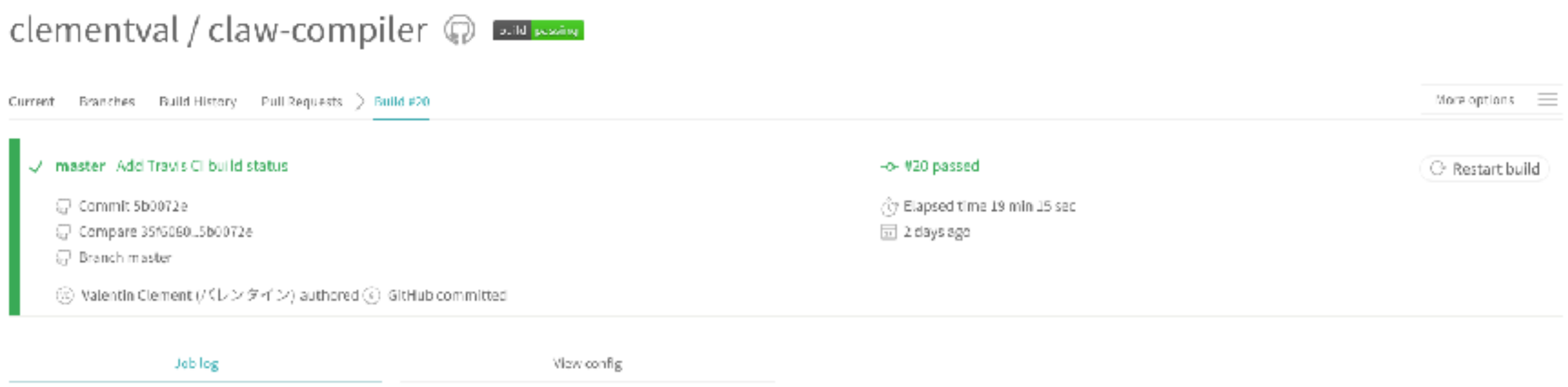
Max-Planck-Institut für Meteorologie

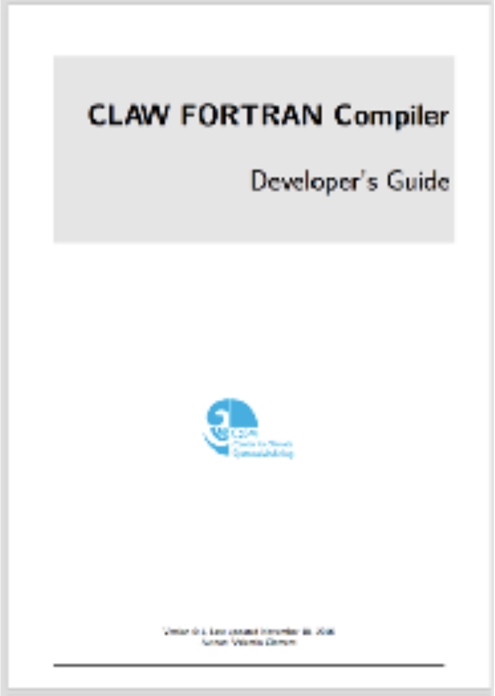
MeteoSwiss

CLAW FORTRAN Compiler - Resources

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

https://github.com/omni-compiler





CLAW FORTRAN Compiler developer’s guide

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

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