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
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)
Cray XC40/ XC50

- Intel® Xeon® E5-2690 v3 2.60GHz, 12 cores, 64GB RAM
- NVIDIA Tesla P100
Performance portability problem - COSMO Radiation

Execution time [s]

-30%

Execution on CPU
Haswell E5-2690v3 12 cores

Execution on GPU
1/2 NVIDIA Tesla K80

-35%

Code restructured for CPU
Code restructured for GPU with OpenACC
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
      END DO
      ! 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 ...

<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fortran 90</strong></td>
<td>173</td>
<td>53998</td>
<td>109381</td>
<td>211711</td>
</tr>
<tr>
<td>C/C++ Header</td>
<td>148</td>
<td>5595</td>
<td>11827</td>
<td>29888</td>
</tr>
<tr>
<td>C++</td>
<td>121</td>
<td>5050</td>
<td>6189</td>
<td>26580</td>
</tr>
<tr>
<td>Python</td>
<td>37</td>
<td>1454</td>
<td>1444</td>
<td>5764</td>
</tr>
<tr>
<td>Bourne Again Shell</td>
<td>17</td>
<td>246</td>
<td>381</td>
<td>3206</td>
</tr>
<tr>
<td>Bourne Shell</td>
<td>33</td>
<td>544</td>
<td>594</td>
<td>2349</td>
</tr>
<tr>
<td>XML</td>
<td>11</td>
<td>272</td>
<td>193</td>
<td>2143</td>
</tr>
<tr>
<td>CMake</td>
<td>9</td>
<td>103</td>
<td>98</td>
<td>793</td>
</tr>
<tr>
<td>make</td>
<td>1</td>
<td>36</td>
<td>27</td>
<td>230</td>
</tr>
<tr>
<td>CUDA</td>
<td>58</td>
<td>4</td>
<td>0</td>
<td>58</td>
</tr>
</tbody>
</table>

**SUM:** 620 68232 130684 286710
### ECMWF IFS - loc

European Centre for Medium-range weather forecasts - Global Model

<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 90</td>
<td>4003</td>
<td>201338</td>
<td>300427</td>
<td>737289</td>
</tr>
<tr>
<td>C/C++ Header</td>
<td>480</td>
<td>846</td>
<td>0</td>
<td>13041</td>
</tr>
<tr>
<td>Perl</td>
<td>1</td>
<td>89</td>
<td>13</td>
<td>240</td>
</tr>
<tr>
<td>SUM:</td>
<td>4484</td>
<td>202273</td>
<td>300440</td>
<td>750570</td>
</tr>
</tbody>
</table>

*only source without external modules*
New German Global model with option to used it as local area model

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

ilev -> dependency nlay size ~= 100
icol -> no dependency ncol size >= 10’000
Performance portability - current code

Sometimes using nproma block optimization for better vectorization

```fortran
!$omp parallel default(shared)
DO igpt = 1, ngptot, nproma
   CALL physical_parameterization(…) ! Code from previous slide
END DO
!$omp end parallel
```
Performance portability - GPU structured code

Loop structure better for GPUs

```plaintext
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  nlay size ~= 100
icol -> no dependency  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
• 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.
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), tau_loc(icol,ilev))
      trans(icol,ilev) = exp(-tau_loc(icol,ilev))
    END DO
  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

```
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
```
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
END DO
DO ilev = 2, nlay + 1
  radn_up(ilev,igpt) = trans(ilev-1) * radn_up(ilev-1,igpt)
END DO
END DO
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

```bash
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)
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 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 SUBROUTINE lw_solver

RRTMGP Example - GPU w/ OpenACC
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 comparison of different kernel version / Domain size: 100x100x42
Piz Kesch (Haswell E5-2690v3 12 cores vs. 1/2 NVIDIA Tesla K80) PGI
Reference: original source code on 1-core

1.0x

0.5x

8.1x

Original code
One column abstraction
CLAW CPU/OpenMP (12cores)
RRTMGP lw_solver - CLAW CPU vs. CLAW GPU

Comparison of different kernel version / Domain Size 100x100x42
Piz Kesch (Haswell E5-2690v3 12 cores vs. 1/2 NVIDIA Tesla K80) PGI
Reference: CLAW CPU/OpenMP 12-cores

<table>
<thead>
<tr>
<th>Speedup</th>
<th>CLAW CPU/OpenMP (12cores)</th>
<th>CLAW GPU/OpenACC (0.5K80)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0x</td>
<td>1.0x</td>
<td>6.1x</td>
</tr>
</tbody>
</table>

Valentin Clement

5th XcalableMP Workshop - 31st of October 2017
Comparison of different kernel version / Domain Size 100x100x42

Piz Kesch (Haswell E5-2690v3 12 cores vs. 1/2 NVIDIA Tesla K80) PGI
Reference: CLAW CPU/OpenMP 12-cores

- CLAW (Cray 8.4.4)
- CLAW (PGI 16.3)
- GridTools (GNU + NVCC)

### Speedup

- **CLAW (Cray 8.4.4)**: 5.0x
- **CLAW (PGI 16.3)**: 4.6x
- **GridTools (GNU + NVCC)**: 5.0x
- **CLAW (Cray 8.4.4)**: 3.3x
- **CLAW (PGI 16.3)**: 1.0x
- **GridTools (GNU + NVCC)**: 0.6x
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

Comparison of different kernel version / Domain Size: 16000x137
Piz Daint (Haswell E5-2690v3 12 cores vs. NVIDIA Tesla P100) Cray 8.6.1
Reference: OpenMP 12-cores

<table>
<thead>
<tr>
<th>Technology</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP / Block</td>
<td>1x</td>
</tr>
<tr>
<td>CLAW GPU/OpenACC</td>
<td>2.6x</td>
</tr>
<tr>
<td>CLAW GPU/OpenACC</td>
<td>2.9x</td>
</tr>
</tbody>
</table>

CLAW GPU/OpenACC (private)
CLAW GPU/OpenACC (promote)
Portability with performance

ECMWF IFS Operational DyCore data layout: nprom, level, block

IFS Physical Parameterizations data layout: nprom, level

ECMWF FVM data layout: jlevel, jnode
Portability with performance

ECMWF IFS Operational DyCore
data layout: nprom, level, block

IFS Physical Parameterizations
data layout: nprom, level

IFS Physical Parameterizations
data layout: level

IFS Physical Parameterizations
data layout: level, jnode

ECMWF FVM
data layout: jlevel, jnode

model cfg

model cfg
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

Change of data layout with copy

Original layout
Optimized layout
Optimized layout w/ copy

Different layout

- Original layout
- Optimized layout
- Optimized layout w/ copy

%
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
  • Maybe automatic offloading to FPGA (FORTRAN to C translation)
Only a problem of MeteoSwiss?
Possible future collaboration

Hartree Centre
Science & Technology Facilities Council

ECMWF

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
valentin.clement@env.ethz.ch

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