Regional Cooperation for Limited Area Modeling in Central Europe

LACE
nwp central europe

An introduction to code to code translation

Martina Tudor (© Daan Degrauwe, Denis Haumont et al)

GeoSphere Austria · DHMZ · Czech Hydrometeorological Institute · HungaroMet Hungarian Meteorological Service · IMGW · METEO ROMANIA · SHMU · ARSO METEO Slovenia

Introduction

Source to source (S2S) translation tools are tools (programs!) that take a program source code and translate it to a program source code, the resulting code should

- essentially compute the same thing, but
- Be optimized/adapted for a particular compiler and hardware platform
- Be (more) efficient :)

S2S tools should allow scientists to write their code without being (too much) bothered about the platform(s) on which their code should run.

S2S tools can impose strict/additional coding rules to the input source code in order to be able to process it (properly). One of the reasons for the code refactoring started in 2022 was to prepare the code for running on GPUs.

S2S tools are also code which can contain bugs and other features.

Why to use GPUs

GPUs have massive computation power

- AMD GPU MI250x peaks at 42.2 TFLOPS

GPUs benefits from highly parallel problems like NWP

GPUs are power efficient

GPUs are here to stay: machine learning working horses

A minimum amount of GPU code is required for some allocations

However:

- There is currently no GPU standard, different manufacturers require different coding approach
- Sometimes new model of GPU from the same vendor requires different GPU directives
- Compilers do not handle the above

That is why we should use code to code translators.

Code adaptation for GPUS

Guidelines

- Current performance on CPU (used for operations) must not degrade (also for vector machines)
- GPU porting should be as transparent as possible for NWP scientists

The code adaptation should allow extending the code in the future

- keep a single code base (that
- code remain readable to scientists

Codebase is large: allow progressive porting, by phases

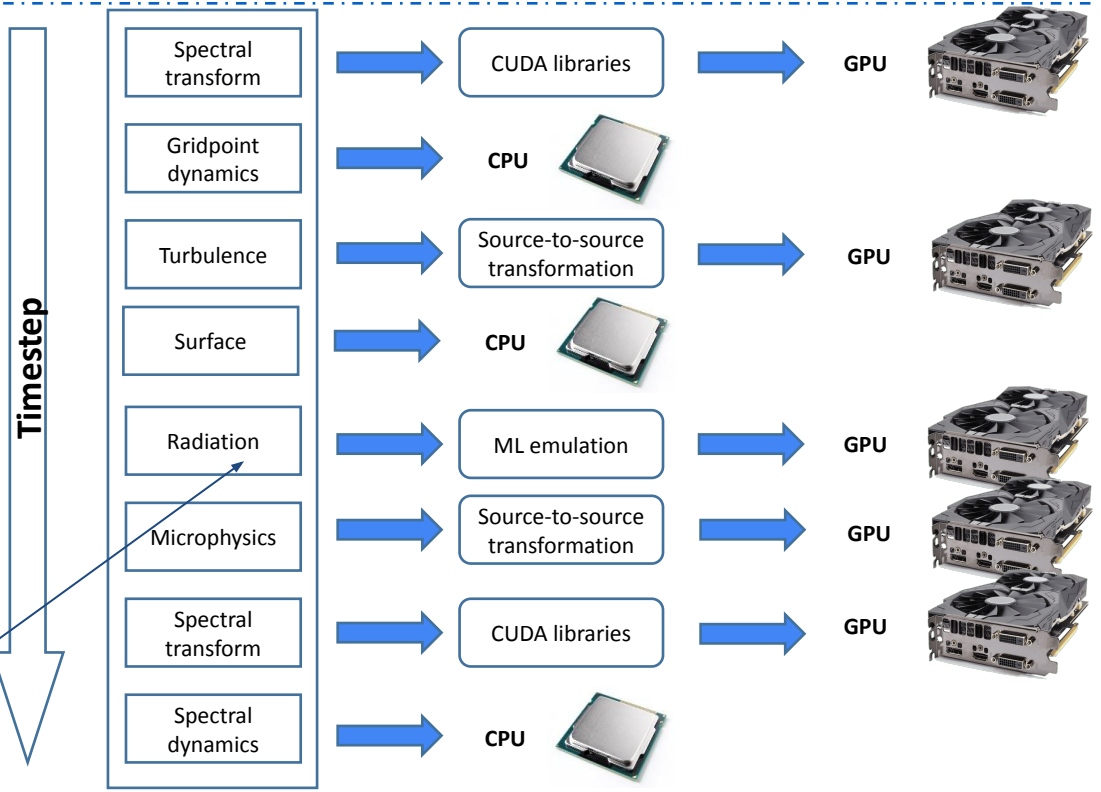
The adapted code should use code to code translators before compilation for specific GPU.

Code adaptation for GPUS (goal)

LACE nwp central europe

An example of what would we like to achieve:

- **Hybrid** execution with part of the code running on GPU and other part on CPU
- Flexibility: the same code can be run on GPUs or CPUs (with different preprocessing and code to code translation tools)
- Source-to-source translation can be used for porting of the scientific code
- Optimized vendor libraries can be used for specific computations (FFT)
- Part of the model can be adapted using S2S translators or emulated via ML

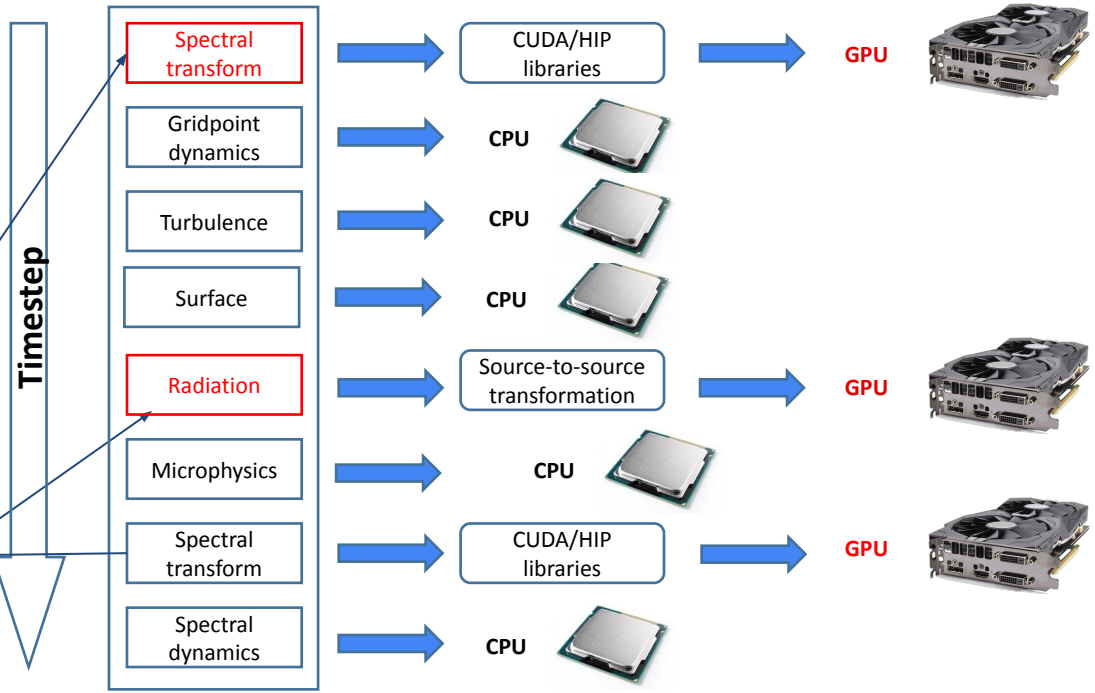


D. Degrauwe & P. Termonia; AMA 2022

Code adaptation for GPUS © Denis Haumont

What worked for ALARO in June 2024:

- **Hybrid** execution with part of the code running on GPU and other part on CPU
- Flexibility: the same code can be run on GPUs or CPUs (with different preprocessing and code to code translation tools)
- Source-to-source translation can be used for porting of the scientific code
- Optimized vendor libraries can be used for specific computations (FFT)
- Radiation is adapted using S2S translators



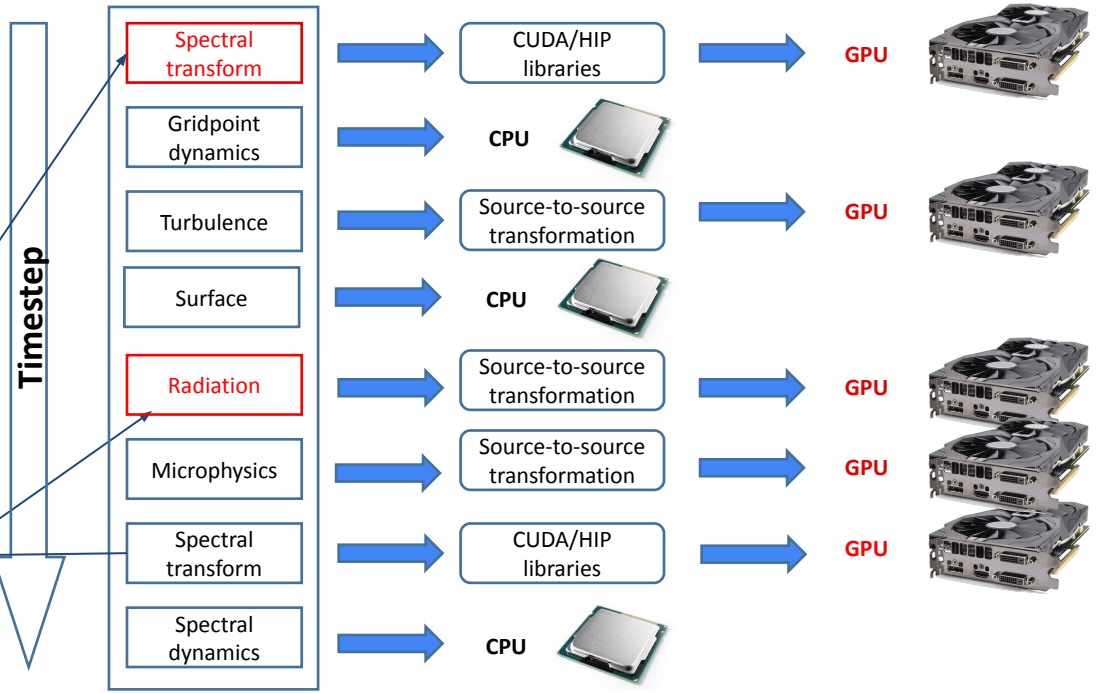
D. Degrauwe & P. Termonia; AMA 2022

Code adaptation for GPUS (update?)

ALARO physics can be run on GPUs (without 3MT for now)

Hybrid execution with part of the code running on GPU and other part on CPU

- Flexibility: the same code can be run on GPUs or CPUs (with different preprocessing and code to code translation tools)
- Source-to-source translation can be used for porting of the scientific code
- Optimized vendor libraries can be used for specific computations (FFT)
- Radiation is adapted using S2S translators



D. Degrauwe & P. Termonia; AMA 2022

Code adaptation for GPUs is based on

- Using hardware-optimized libraries where possible: hipBLAS, hipFFT, …
- Increase flexibility by improving code layout (i.e refactoring) and enforce coding standard rules (https://sites.ecmwf.int/docs/ifs-arpege-coding-standards/fortran/)
- Use of smart data-structure (Field API) to manage memory and data transfer (memory communication between CPU and GPU, can be time consuming)
- The code that is regularly added or changed by scientists goes through the source-to-source translation tool in a (semi)-automated way

  We can use Loki or fxtran scripts to implement an OpenACC directives to the Fortran code

fxtran

Source to source (S2S) translation tools written by Phillipe Marginaud in Perl

- can be seen almost as an advanced text manipulation tool, where strings are replaced by other strings based on certain patterns.
- Therefore fxtran is much faster than loki
- Used in MF on ARPEGE code
- Used by Daan on ALARO (it works for rad, turb, microphysics)
    - We have recipes that transforms successfully all the physics parameterization of Alaro.
    - We also have recipes that work for some part of the dynamical core, mainly the semi-lagrangian.
    - We are waiting for new recipes for the spectral semi-implicit.
    - We also have very efficient GPU code for spectral conversion (FFT) using vendor library
-

The principle of source to source translation is to apply a set of recipes to the code, which explain how to transform the code.

Philippe used Fxtran (ie in Perl) to work out a set of recipes for Arpege (that is also working for Alaro, because the code follow the same structure).

The exact same recipes are now being implemented in Loki (ie in Python).

So Fxtran or Loki can be seen as an "implementation detail", the language chosen for implementing the recipes (which are the important part).

And yes, in the future it will be 100% Loki, but since all the recipes are not yet available, we still relied on the prototype in Fxtran.

show?

loki

ECMWF in-house open source Python package source to source translation tool.

- loki "understands" the code: it is able to link information (type definitions, variable types, subroutine signatures) between files.
- loki offers more advanced things like inlining functions and subroutines (even if they reside in another module); passing module variables as arguments through several nested subroutine calls, etc.

https://github.com/ecmwf-ifs/loki/

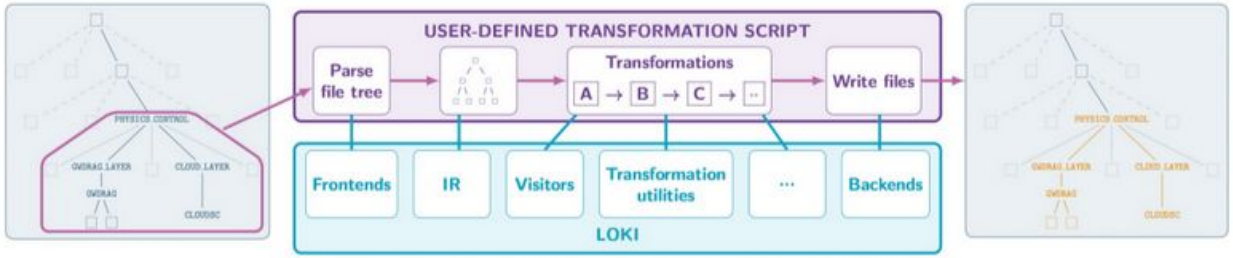
- Automated - it is a tool, it requires instructions from the user, a set of instructions is referred to as the recipe, and then it is executed to transform the input F90 source code to produce GPU compilable (optimized?) F90 code
- domain-specific language (DSL) concept for separating legible science from highly optimised code layers targeting different processor types, and automated source-to-source code transformation tools

“NWP codes have been developed over decades and comprise a large monolithic code base. Over the course of their lifetime, compute architectures have evolved from vector computers to distributed memory multi-processors and hybrid accelerated supercomputers. NWP codes are run on a larger variety of systems and have to target diverse hardware architectures today. Meanwhile, the objective of performance portability using a single programming model remains elusive and accommodating multiple bespoke and sometimes conflicting optimisations for specific hardware architectures becomes increasingly unsustainable.

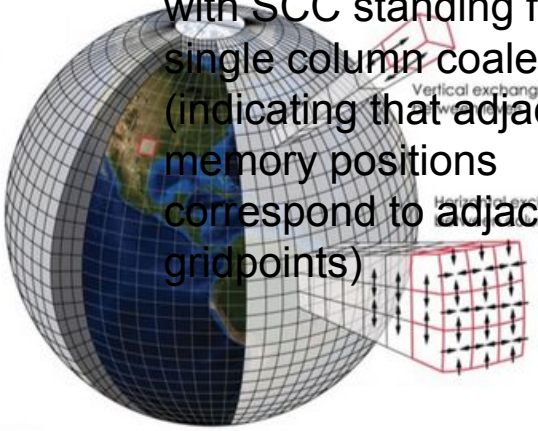
a freely-programmable API and inter-procedural analysis features to encode custom transformations that are applied programmatically across large source trees

loki

Loki: Source-to-source translation package written in Python



- Library of tools and APIs to build source transformation recipes
- Detailed talk by B. Reuter on Wednesday, 10.30, HG F 1
- **Recipes depend on "Single Column" formatted code** [2]
  - Additional coding conventions that allow tools to re-optimize code for GPUs via loops flips and changing variable declarations
  - SIMD to SIMT conversion with IFS-specific memory layout (memory blocking with packed arrays and sparse slices)



the "single column" idea is somewhat outdated since it doesn't fit with our nproma-sliced memory layout. Therefore the loki recipes are named SCC* with SCC standing for single column coalesced (indicating that adjacent memory positions correspond to adjacent gridpoints)

[2] **Valentin Clement et al.** "The CLAW DSL: Abstractions for Performance Portable Weather and Climate Models". In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '18. 2018.

Loki-SCC - GPU-specific transformation pipelines

Single Column Coalesced loop / array format

1. **Devectorize:** Remove all loops across the horizontal
   - Use application knowledge (eg. loop variable names)
2. **Demote** vector-level temporaries to scalars
   - Reduce number of (serial) memory allocations on device
3. **Revectorise** with high-level vector loops
   - SIMT-friendly loop layout – uses "-gvmode" flag on Nvidia
4. **Annotate** with either OpenACC or OpenMP
5. **Option**: Deal with remaining temporaries
   - Hoist to driver and pre-allocate before invoking kernel
   - "Stack" pool allocator using Cray pointers into single allocation

```
SUBROUTINE KERNEL(klon, klev, a)
  real, intent(inout) :: a(klon, klev)
  real                :: tmp(klon)
  integer             :: j, k

  do i=1, klon
    tmp(i) = <f>(a(i, 1)
  end do

  do k=2, klev

    do i=1, klon
      tmp2(I, k) = <g>(a(i, k) + tmp(ji)
    end do
  end do
END SUBROUTINE KERNEL
```

```
SUBROUTINE KERNEL(klon, klev, a)
  real, intent(inout) :: a(klon, klev)
  real                :: tmp
  integer             :: j, k

  do i=1, klon
    tmp(i) = <f>(a(i, 1)

    do k=2, klev
      tmp2(I, k) = <g>(a(i, k) + tmp
    end do
  end do
END SUBROUTINE KERNEL
```

Note that this is just one recipe already implemented in loki: various alternatives exist, with the horizontal loop either in the driver or in the kernel, with different options to fuse loops or split loops, etc.
While the different recipes don't make a huge difference on performance, on future platforms they might. This richness of recipes in Loki is another benefit wrt fxtran

Useful links

https://opensource.umr-cnrm.fr/projects/accord/wiki/CRA_Documentation

https://sites.ecmwf.int/docs/ifs-arpege-coding-standards/fortran/

DEODE stuff (?)

https://github.com/ecmwf-ifs/loki/

Session - PASC 2024

Presentation - PASC 2024

Modernisation of the Integrated Forecasting System | ECMWF

From the Scalability Programme to Destination Earth | ECMWF

Thank you for your attention!