Status and experiences from the transfer of the new fast waves solver into STELLA

COSMO General Meeting
08-11 Sept. 2014, Eretria, Greece

Michael Baldauf (DWD), Andrea Arteaga (ETH)

Goal:

rewrite of the ‚new fast waves solver' (`fast_waves_sc.f90`, COSMO 5.0) with the stencil library (DSEL) STELLA

Status:

- All stencils that are needed for an operational run are available:

  ```
  FastWavesSCCalcTotalTPRho, FastWavesSCCalcDzDx, FastWavesSCInitDivDampCoeff
  FastWavesSCVerticalDivergenceHelper, FastWavesSCUV, FastWavesSCLHS,
  FastWavesSCRHS, FastWavesSCTridiag, FastWavesSCPPTP
  ```
- all these stencils have successfully passed the unit testing framework
- the integration of these stencils in the right sequence (=time loop of small time steps) has been done (work by Andrea Arteaga)
- halo exchange is ready (work by Andrea Arteaga)

Still to do:

- integration of the FW solver into the time integration module of the C++-dycore

code example:

the Fortran subroutine `calc_Z_horiz` and its corresponding part of the `FastWavesSCVerticalDivergenceHelper`-stencil

(only for demonstration; the real stencil contains more than only this subroutine)

additionally: registration of new variables in Repository files; adaptation of some header-Files

```
SUBROUTINE calc_Z_horiz ( u, v, u_sfc, v_sfc, Z_horiz )

  USE grid_metrics_utilities, ONLY: wgtfac_u, wgtfac_v

  IMPLICIT NONE

  REAL (KIND=wp), INTENT (IN)  :: u(ie,je,ke), v(ie,je,ke)
  REAL (KIND=wp), INTENT (IN)  :: u_sfc(ie,je), v_sfc(ie,je)

  REAL (KIND=wp), INTENT (OUT) :: Z_horiz(ie,je,ke1)

  REAL (KIND=wp),     ALLOCATABLE :: Z_x(:,:), Z_y(:,:)

  INTEGER :: i, j, k
  INTEGER :: istat

  ALLOCATE( Z_x (ie, je), STAT=istat)
  ALLOCATE( Z_y (ie, je), STAT=istat)

  Z_horiz(:,:,1:MAX(1,vcoord%kflat)) = 0.0_wp

  DO k=MAX(2,vcoord%kflat+1), ke

    DO j=jstart, jend
      DO i=istart-1, iend
        Z_x(i,j) =   dz_dlam(i,j,k) *                       &
          &   (            wgtfac_u(i,j,k)   * u(i,j,k  )   &
          &   + ( 1.0_wp - wgtfac_u(i,j,k) ) * u(i,j,k-1) )
      END DO
    END DO

    DO j=jstart-1, jend
      DO i=istart, iend
        Z_y(i,j) =   dz_dphi(i,j,k) * crlat(j,2) *          &
```

```
#include "StencilFramework.h"
#include "DycoreConstants.h"
#include "DycoreGlobals.h"
#include "FiniteDifferenceFunctions.h"
#include "HeightLevelFunctions.h"
#include "FastWavesSCVerticalDivergenceHelper.h"
#include "MathFunctions.h"

/**
 * Stencil function extrapolating the surface velocity
 * using a 1st order method
 */
template<typename TEnv>
struct HorizontalZComponent
{
  STENCIL_FUNCTION(TEnv)

  FUNCTION_PARAMETER(0, dz_dalpha)
  FUNCTION_PARAMETER(1, vel)
  FUNCTION_PARAMETER(2, weight)

  __ACC__
  static T Do(Context ctx)
  {
    return ctx[dz_dalpha::Center()] * (
                   ctx[weight::Center()]  * ctx[vel::Center()]
       + ((T)1.0 - ctx[weight::Center()]) * ctx[vel::At(kminus1)]
         );
  }
};

// define parameter enum
enum
{
```

Some personal experiences with STELLA

- Use of *eclipse* was recommended during the workshop (Dec. 2012) but a **standard editor** like *emacs* was sufficient for me as a STELLA *user*
- As a user, you **need help**! (many thanks to Andrea)
  Most of the time I didn‘t really knew what I was doing; it is merely ‚pattern recognising‘ and ‚copy&paste‘ work.
- Very helpful: working on CSCS computers; svn; ‚script.sh‘
- Amount of time for this work for MB:
  - 1 week trainings course (Dec. 2012)
  - 1 week work at MeteoCH (Feb. 2013, thanks to Oliver Fuhrer, Tobias Gysi, Ben Cummings)
  - 3 weeks work at DWD (Feb. – Aug. 2014) with intensive communication (phone, eMail) with Andrea
  - role of thumb: transfer ~100 lines of Fortran code/day into STELLA code (without testing!)
- **compilation**: can be split up into
  - step 1: (‚easy‘) you get errors by violating C++-syntax rules
  - step 2: (‚hard‘) you get errors by violating STELLA syntax rules (often several 1000 lines of error messages for only one mistake)
    My approach: search for the name of the stencil which seems to be suspicious
    New: search for ‚STELLA_ERROR‘

- you should know the syntactical rules of **C++**; however a deeper knowledge of C++ (esp. object-orientation) is not necessarily needed
- There are several ‚syntactical ‘ rules of STELLA you have to know;
  in my opinion, it is *not a library* but some sort of a **new programming language/framework**
- You have to know or be aware of several **internal rules** of STELLA, e.g.
  - 2D versus 3D fields:
    `u_sfc(i,j) = 1/2*( u(i,j,ke) + u(i-1,j,ke) )` ←→
    `ctx[u_sfc::Center()]=0.5*(ctx[u::Center()]+ctx[u::At(Offset<-1,0,0>())]);`
  - Which are the priority rules, if I define several do-methods with intersecting k-ranges:
    static void Do(Context ctx, TerrainCoordinates) { … }
    static void Do(Context ctx, KMaximum) { … }
  - ….
- → **unit testing** is very helpful and necessary!
  But of course, this works only if a reference-code (here: COSMO-Fortran-version) is available.
  Nevertheless setup of the test consumes a lot of working time (again many thanks to Andrea)
- sometimes the Fortran code had to be adapted to successfully run the unit test (example: `wgtfacq_u(:,:, 1:3)` → `(1:ke)` and use only `(ke-2:ke)`)

- I never experienced failures of STELLA itself; seems to be quite **stable**!
- the concept of an **embedded** language (DSEL) is tempting.
  However, the strong syntactical rules of STELLA (and a lot of overhead) heavily disturb these advantages.
  How easy is it to circumvent STELLA (e.g. to implement new developments ‚directly ' in C++) (is it ‚nothing or all '?)

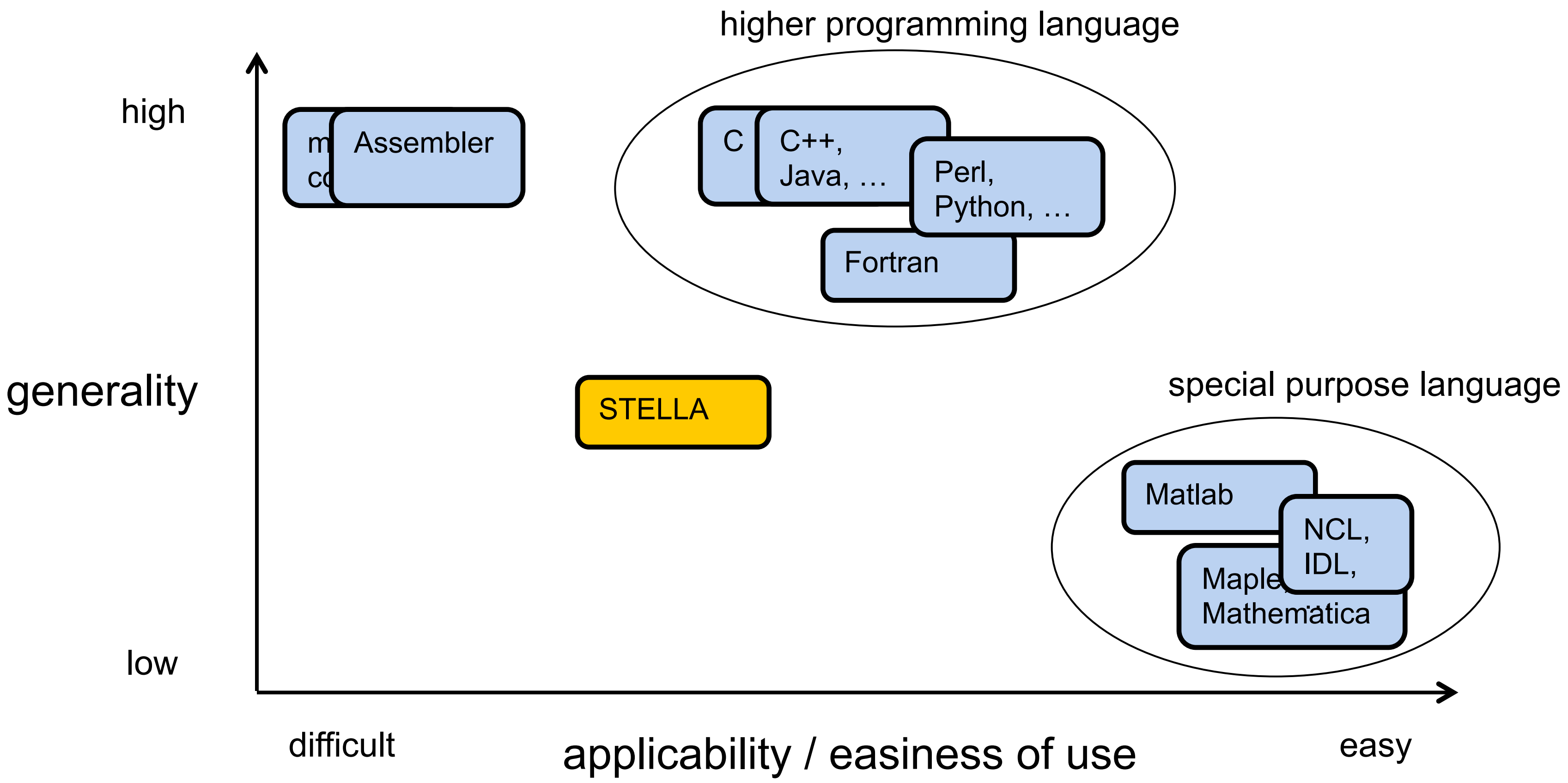
Some further considerations:

- Several **namelist-parameters** now are hard-wired (→ increase in efficiency for the template meta-programming).
  Is this still needed? (it does not match with our daily development working style)
- I couldn 't test efficiency. However, it is clear that efficiency doesn 't come for free: you have to define the stencils in a way that cache efficiency can be achieved → also STELLA user needs some knowledge about such things

Summary

- the STELLA-version of the new fast waves solver is available (only those options are implemented, which are currently operationally used)
- STELLA is doubtlessly an impressive development from the computational science perspective.
- During this code transfer, no errors in STELLA itself have been occured
- the STELLA version of the code is needed if you want to use GPU based computers
- However, STELLA is not easy to use.
  - The user needs a lot of help from experienced developers.
  - Time to implement a stencil can be a factor 5-10 larger than for the equivalent Fortran code (means: time to transfer existing code)
  - More documentation is needed
  - It is rather a new programming framework than a library.

The landscape of programming languages and frameworks concerning two aspects … personal view



of course, e.g. the important dimension ‚efficiency‘ is not contained here

Outlook

•concept of an ‚embedded ' DSEL sounds tempting, but my experience until now shows, that the user has only little advantages from it.
Should one think about a sort of artificial macro language (e.g. with a syntax close to Matlab, …)?

•**Recommendation**:
from my point of view it is absolutely necessary to **keep the Fortran version** of the COSMO model instead to replace it entirely by a STELLA-version.

- Otherwise, significant further developments of the dynamical core are *stopped!*
- unit testing wouldn 't be available any more
- model development by testing via namelist-parameters

•a permament transfer of possible new developments from the Fortran-version to the STELLA-version must be organized
(this cannot be done by the model developers!) → Task for STC