Porting ARPEGE physics to NVIDIA

Principles for porting (from ECMWF)

- No vendor directives in the code, except for well identified parts (spectral transforms, Field API library, etc.)
- Transform grid-point code using scripts
- Encapsulate field data using Field API

The Data

apl_arpege.F90

- Use Field API **everywhere**
- No more PGFL, PGMV, etc.
- No more module variables
- Finalized in cycle 48t3
- Refactor acvppkf.F90 (Judicael Grasset)

Constant & non-constant data on device

- Use YDMODEL/YDGEOMETRY to store these data
- Generate code with fxtran for pushing YDMODEL & YDGEOMETRY on the device.

Used routinely

The Device

Transform NPROMA routines into OpenACC routines

- Scripts based on fxtran
- Best method = “single directive” (aka “SCC”)
- Use a pre-allocated stack buffer for temporary arrays
  - → Port easily large kernels using single directive method
  - → Optimal memory re-use
  - → Process routines independently
- actke.F90 → actke_openacc.F90, etc.

x86 HOST CODE NVIDIA

```
! Distribute blocks on x86 cores
!$OMP PARALLEL DO PRIVATE
DO JBLK=1, NGPBLKS



  KIDIA = 1
  KFDIA = KLON
  CALL ACRANEB2( &
     & YDERDI,YDRIP,YDML_PHY_MF,KIDIA,KFDIA,KLON,KTDIA,KLEV,&
```

```
! Distribute blocks on shared multiprocessors (32 cores)
!$ACC PARALLEL LOOP GANG VECTOR_LENGTH (KLON)
DO JBLK=1, NGPBLKS
  !$ACC LOOP VECTOR PRIVATE (KIDIA,KFDIA)
! Distribute columns of block on cores of shared multiprocessor
  DO JLON = 1, KLON
    KIDIA = JLON
    KFDIA = JLON
    CALL ACRANEB2( &
       & YDERDI,YDRIP,YDML_PHY_MF,KIDIA,KFDIA,KLON,KTDIA,KLEV,&
```

x86 DEVICE CODE NVIDIA

```
DO JLEV=KTDIA,KLEV
  DO JLON=KIDIA,KFDIA  ! AVX vectorization (single core)
    ZZEO2TA=2._JPRB*BSFTA(JAE)*EODTA(JAE)*PDAER(JLON,JLEV,JAE)
    ZZEO2SA=2._JPRB*BSFSA(JAE)*EODSA(JAE)*PDAER(JLON,JLEV,JAE)
    ZEO2TA(JLON,JLEV)=ZEO2TA(JLON,JLEV)+ZZEO2TA
    ZEO2SA(JLON,JLEV)=ZEO2SA(JLON,JLEV)+ZZEO2SA
```

```
JLON = KIDIA
...
DO JLEV=KTDIA,KLEV

  ZZEO2TA=2._JPRB*BSFTA(JAE)*EODTA(JAE)*PDAER(JLON,JLEV,JAE)
  ZZEO2SA=2._JPRB*BSFSA(JAE)*EODSA(JAE)*PDAER(JLON,JLEV,JAE)
  ZEO2TA(JLON,JLEV)=ZEO2TA(JLON,JLEV)+ZZEO2TA
  ZEO2SA(JLON,JLEV)=ZEO2SA(JLON,JLEV)+ZZEO2SA
```

The Host

Transform NPROMA routines into parallel routines

- Scripts based on fxtran
- Method = “pointer parallel”
- Use custom directives
- apl_arpege.F90 → apl_arpege_parallel.F90
- apl_alaro.F90 → apl_alaro_parallel.F90
- etc.

“Pointer parallel”

```
!$ACDC PARALLEL, &
!$ACDC TARGET=OpenMP/OpenMPSingleColumn/OpenACCSingleColumn, &
!$ACDC NAME=ACHMTLS {

  CALL ACHMTLS (YDMODEL%YRCST, YDMODEL%YRML_PHY_MF, ... &
  & ..., YDMF_PHYS_BASE_STATE%YGSP_RR%T, ZGWDCS, ZDSA_LHS, ZPCLS, ... &
  DO JLON=YDCPG_BNDS%KIDIA,YDCPG_BNDS%KFDIA
    ZDPHI(JLON)=PAPHIF(JLON,YDCPG_OPTS%KFLEVG)-PAPHI(JLON,YDCPG_OPTS%KFLEVG)
    ZPRS(JLON)=YDMODEL%YRCST%RD+ZRVMD*YDCPG_MISC%QS(JLON)
    ZRTI(JLON)=2.0_JPRB/(PR(JLON,YDCPG_OPTS%KFLEVG)*...
  ENDDO

!$ACDC }
```

Generate **3 parallel** sections

- OpenMP : traditional OpenMP, on the host
- OpenMPSingleColumn : host, for validation
- OpenACCSingleColumn : device
- Select section at runtime

Traditional OpenMP

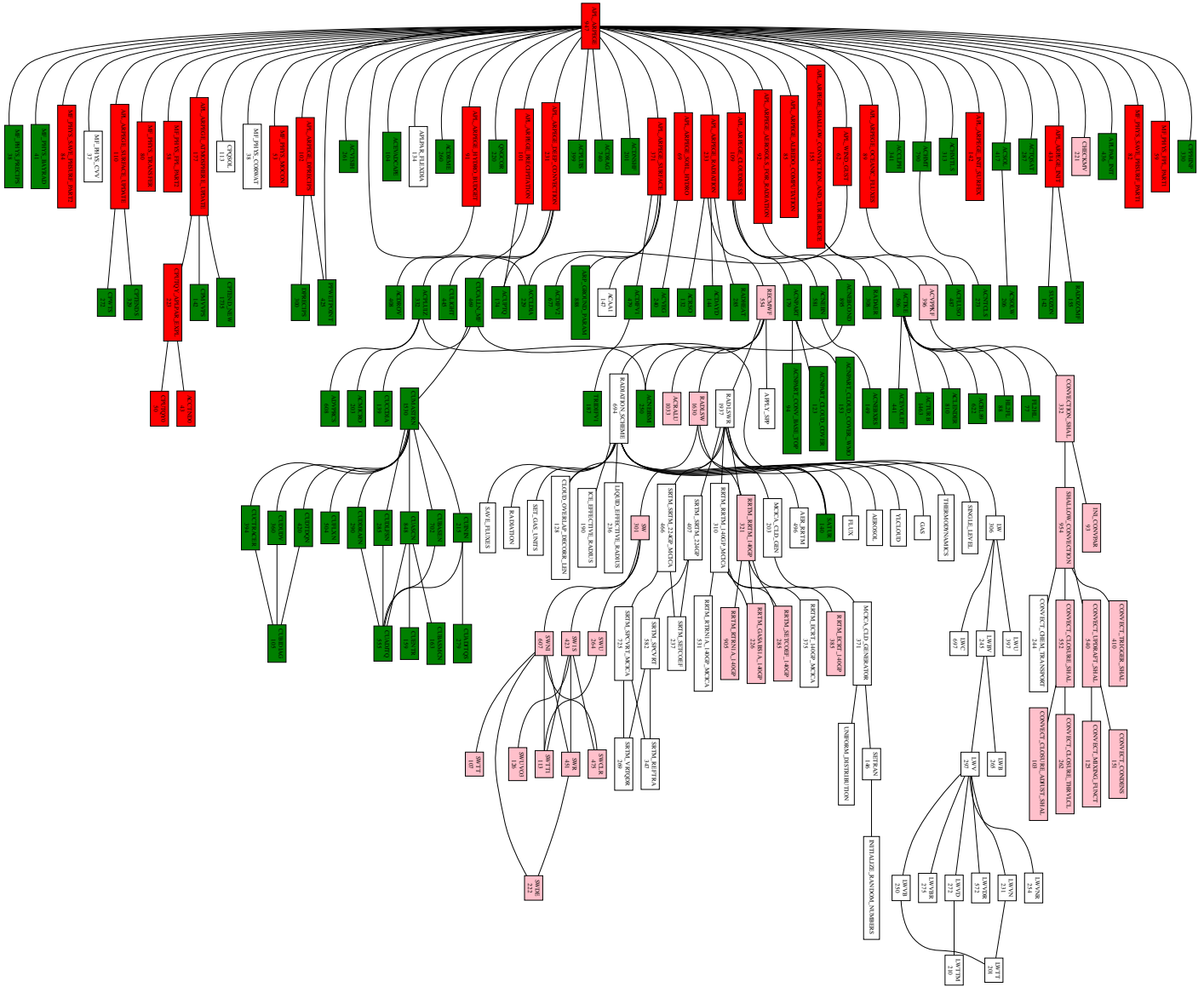
```
Z_YDMF_PHYS_BASE_STATE_YGSP_RR_T => &
   & GET_HOST_DATA_RDONLY (YDMF_PHYS_BASE_STATE%YGSP_RR%F_T)    ! Get pointers to data
ZGWDCS => GET_HOST_DATA_RDWR (YL_ZGWDCS)

!$OMP PARALLEL DO PRIVATE (JBLK, JLON, YLCPG_BNDS)

    DO JBLK = 1, YDCPG_OPTS%KGPBLKS
      YLCPG_BNDS = YDCPG_BNDS
      CALL YLCPG_BNDS%UPDATE (JBLK)

      CALL ACHMTLS (YDMODEL%YRCST, YDMODEL%YRML_PHY_MF, YLCPG_BNDS%KIDIA, &
       & YLCPG_BNDS%KFDIA, YDCPG_OPTS%KLON, YDCPG_OPTS%KFLEVG, &
       & Z_YDMF_PHYS_BASE_STATE_YCPG_DYN_PHI(:,:,JBLK), &
       & Z_YDMF_PHYS_BASE_STATE_YCPG_DYN_PHIF&
...
       & ZPCLS(:,JBLK), ZFLU_CD(:,JBLK), ZFLU_CDN(:,JBLK))

        DO JLON=YLCPG_BNDS%KIDIA, YLCPG_BNDS%KFDIA
          ZDPHI (JLON,JBLK)=Z_YDMF_PHYS_BASE_STATE_YCPG_DYN_PHIF(JLON, ...&
...
```

OpenMP & Single column

```
ZDPHIT => GET_HOST_DATA_RDONLY (YL_ZDPHIT)          ! Get pointers to data (on the host)
ZDPHI => GET_HOST_DATA_RDWR (YL_ZDPHI)

!$OMP PARALLEL DO PRIVATE (JBLK, JLON, YLCPG_BNDS, YLSTACK)
DO JBLK = 1, YDCPG_OPTS%KGPBLKS
  DO JLON = 1, MIN (YDCPG_BNDS%KLON, YDCPG_BNDS%KGPCOMP - (JBLK - 1) * YDCPG_BNDS%KLON)
    YLCPG_BNDS%KIDIA = JLON                                  ! Select single column
    YLCPG_BNDS%KFDIA = JLON
    YLSTACK%L = stack_l (YSTACK, JBLK, YDCPG_OPTS%KGPBLKS)   ! Stack setup
    YLSTACK%U = stack_u (YSTACK, JBLK, YDCPG_OPTS%KGPBLKS)
    CALL ACHMTLS_OPENACC (YDMODEL%YRCST, YDMODEL%YRML_PHY_MF, YLCPG_BNDS%KIDIA, &
  & YLCPG_BNDS%KFDIA, YDCPG_OPTS&%KLON, YDCPG_OPTS%KFLEVG,  &
...
  & ZMRIPP(:,:,JBLK), ZDSA_CPS(:,JBLK), ZGWDCS(:,JBLK), ZDSA_LHS(:,JBLK), &
  & ZPCLS(:,JBLK), ZFLU_CD(:,JBLK), ZFLU_CDN(:,JBLK),YDSTACK=YLSTACK)
```

OpenACC & Single column

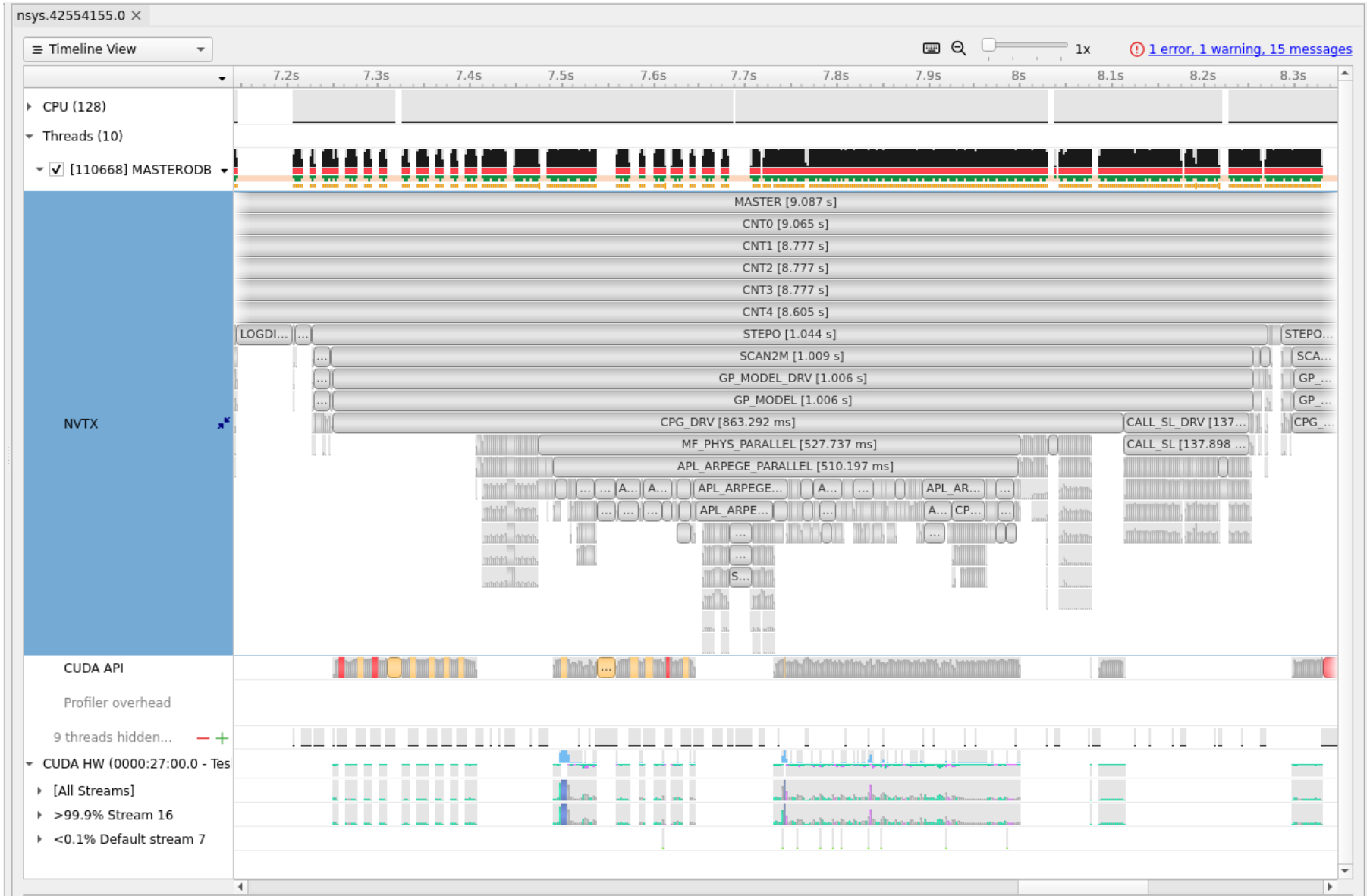
```
!$ACC PARALLEL LOOP GANG &
!$ACC&PRESENT (YDMODEL, ZDPHI, ZDPHIT, ZDSA_CPS, ZDSA_LHS, ZFLU_CD, ZFLU_CDN, ZGWDCS, &
!$ACC&         Z_YDMF_PHYS_OUT_GZ0H) &   ! State that all data is present on device
!$ACC&PRIVATE (JBLK, YLCPG_BNDS)
DO JBLK = 1, YDCPG_OPTS%KGPBLKS
  !$ACC LOOP VECTOR &
  !$ACC&PRIVATE (JLON, YLCPG_BNDS, YLSTACK)

  DO JLON = 1, MIN (YDCPG_BNDS%KLON, YDCPG_BNDS%KGPCOMP - (JBLK - 1) * YDCPG_BNDS%KLON)
    YLCPG_BNDS%KIDIA = JLON                                    ! Single column selection
    YLCPG_BNDS%KFDIA = JLON
    YLSTACK%L = stack_l (YSTACK, JBLK, YDCPG_OPTS%KGPBLKS)  ! Stack setup
    YLSTACK%U = stack_u (YSTACK, JBLK, YDCPG_OPTS%KGPBLKS)
    CALL ACHMTLS_OPENACC (YDMODEL%YRCST, YDMODEL%YRML_PHY_MF, & ! Call OpenACC SCC routine
   & YLCPG_BNDS%KIDIA, YLCPG_BNDS%KFDIA, YDCPG_OPTS%KLON, YDCPG_OPTS%KFLEVG, &
   & Z_YDMF_PHYS_BASE_STATE_YCPG_DYN_PHI(:,:,JBLK), &
```



apl_arpege.F90

Red = parallel routines
Green = OpenACC/SCC
White = unused
Pink = not yet ported

~ 100 routines transformed
~ 70 parallel regions



T31L15 on V100

Limitations

- Transfers before/after each kernel for now
- Works only where Field API members are explicitly referenced
  - → Will not work with cpg_gp.F90/cpg_dyn.F90
  - → Constraints on kernel boundaries
- **Problem with pointer aliasing**

Performances (from CERFACS)

- 10000$ = CPU node (128 core AMD Rome or 112 core Intel Sapphire Rapids)
- 50000$ = node with 4 GPU A100
- single A100 ~ 140 to 160 CPU cores (in terms of $)

Kernel	x86 cores equivalent to A100
ACDRAG	42
ACTKE	94
CUCALLN	223
ACVPPKF	coming soon

Note : Comparisons made with x86 “fine” granularity mode

A few figures

	Time	Lines +	Lines -	Commits
Physics refactoring	8 mm	51k	27k	~900
cpg_gp.F90/cpg_dyn.F90 refactoring	5 mm	20k	17k	~600

- ARPEGE/AROME code re-engineering ~ 90%
- Source to source transformation ~ 10%
- Re-use of ESCAPE components, principles or estimates = 0