# Development constraints

## *including presentation of constraints due to shared code with Arp/Ifs*

Filip Váňa

filip.vana@chmi.cz

ONPP / ČHMÚ - LACE

# Outline

- Basic rules
- Parallelization principles
- Concept of NPROMA
- Data structures

# Basic code rules

- Coding rules and conventions (Karim's talk)

# Basic code rules

- Coding rules and conventions (Karim's talk)
- Bit reproducibility (with respect to different NPROMA values and different no. of PEs)

# Basic code rules

- Coding rules and conventions (Karim's talk)
- Bit reproducibility (with respect to different NPROMA values and different no. of PEs)
- Platform independence - optimized for Scalar and Vector platforms

# Basic code rules

- Coding rules and conventions (Karim's talk)
- Bit reproducibility (with respect to different NPROMA values and different no. of PEs)
- Platform independence - optimized for Scalar and Vector platforms
- Parallel code - allows parallel computation, supports MPI and OpenMP standards

# Basic code rules

- Coding rules and conventions (Karim's talk)
- Bit reproducibility (with respect to different NPROMA values and different no. of PEs)
- Platform independence - optimized for Scalar and Vector platforms
- Parallel code - allows parallel computation, supports MPI and OpenMP standards
- MPI/OpenMP called only through MPL/OML modules (wrappers), CDSTRING should be set to the name of the caller routine

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organization ($S \rightarrow M \rightarrow L \rightarrow G \rightarrow L \rightarrow M \rightarrow S$)

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organization (S $\rightarrow$ M $\rightarrow$ L $\rightarrow$ G $\rightarrow$ L $\rightarrow$ M $\rightarrow$ S)
- No a prior ordering of model fields

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organization (S $\rightarrow$ M $\rightarrow$ L $\rightarrow$ G $\rightarrow$ L $\rightarrow$ M $\rightarrow$ S)
- No a prior ordering of model fields
- Indexing of model arrays is not arbitrary

# Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organization ($S \to M \to L \to G \to L \to M \to S$)
- No a prior ordering of model fields
- Indexing of model arrays is not arbitrary
- all configurations share a single top-level call tree (the control levels has to be preserved:
  MASTER -> CNT0 -> CNT1 -> CNT2 -> CNT3 -> CNT4 -> STEPO
  MASTER -> CNT0 -> CVA1 -> CVA2 -> CONGRAD -> SIM4D -> CNT3 -> ... )

# Parallelization

Computer architecture fundamentals:

$$\underbrace{\left.\begin{array}{c} \underbrace{\left.\begin{array}{c}\text{CPU}\\ \text{CPU}\\ \ldots\end{array}\right\}\text{node}}_{\text{OpenMP}} \\ \underbrace{\left.\begin{array}{c}\text{CPU}\\ \text{CPU}\\ \ldots\end{array}\right\}\text{node}}_{\text{OpenMP}} \\ \ldots \end{array}\right\}}_{\text{MPI}} \text{distributed memory cluster}$$

CPU - vector or scalar

node = collection of CPU with share a common memory

# Parallelization strategy

- MPI = Distributed memory parallelization - available since AL08
- OpenMP = Shared memory parallelization - available since AL29 (for AD code on Vector since AL32T2)
- Mixed/hybrid MPI and OpenMP parallelization - available since AL35T2

# Parallelization strategy - MPI

- Transposition strategy = complete data required is redistributed at various stages of a timestep so that the arithmetic computations between two consecutive transpositions can be performed without any inter-processor communication.
- Inter-processor communication is localized in a few routines and rest of the model need have no knowledge of this activity.
- Communication is realized through relatively long messages ( 1Mbytes)
  *(Remember: short messages are bounded by latency of interconnect; long messages are bounded by bandwidth of interconnect)*

# Parallelization strategy - MPI II.

## Different types of blocking strategy:

**MP_TYPE = 1** blocked mode

**MP_TYPE = 2** buffered mode - MPI_BSEND can return before the receive is called on the receiving processor. (This allows to reuse/destroy the sending array.)

**MP_TYPE = 3** immediate mode - send and receive are returned immediately as the comms are performed in the background. Additional calls are then required to check or wait for the completion of a comm. (Sending array can be reused/destroyed only after MPI is confirmed to do so.)

# Parallelization strategy - MPI III.

## GP computation

| | |
|---|---|
| **NPROC** | Total number of processors to be used |
| **NPRGPNS** | Number of PEs in the North-South direction |
| **NPRGPEW** | Number of PEs in the East-West direction |
| **LSPLIT** | Allows the splitting of latitude rows |

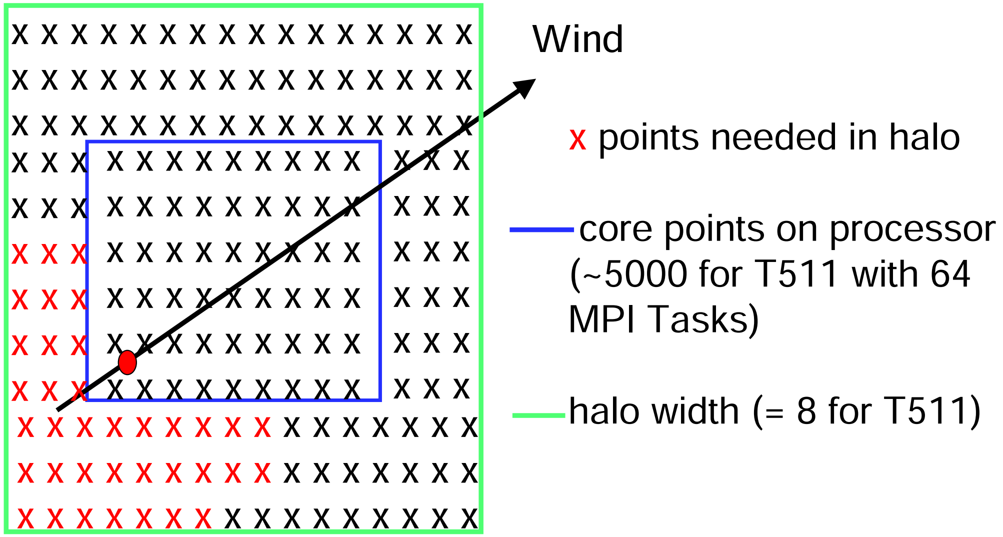
# Parallelization strategy - MPI III.

## GP computation

| | |
|---|---|
| **NPROC** | Total number of processors to be used |
| **NPRGPNS** | Number of PEs in the North-South direction |
| **NPRGPEW** | Number of PEs in the East-West direction |
| **LSPLIT** | Allows the splitting of latitude rows |

## SL comms as a specific feature

- squarer shape of domain = reduced comm volume for SL
- SL on demand - targets (= reduces) the area of comms computed from VMAX2



# Parallelization strategy - MPI IV.

## Fourier transformation

| | |
|---|---|
| **NPRTRW** | Number of processors in zonal/meridional decomposition (usually NPRTRW=NPRGPNS) |
| **NPRTRV** | Number of processors in vertical decomposition (usually NPRTRV=NPRGPEW) |

- Decomposition along latitudes/longitudes * levels (there's no further independence across the fields).
- This means that for example Alaro/CE with 540*432 points and 87 levels reaches scalability limit for transformation at around 432*87=37584 MPI processes. (GP decomposition of the same domain and NPROMA=20 reaches its limit at around 421*540/20=11367 MPI processes.)

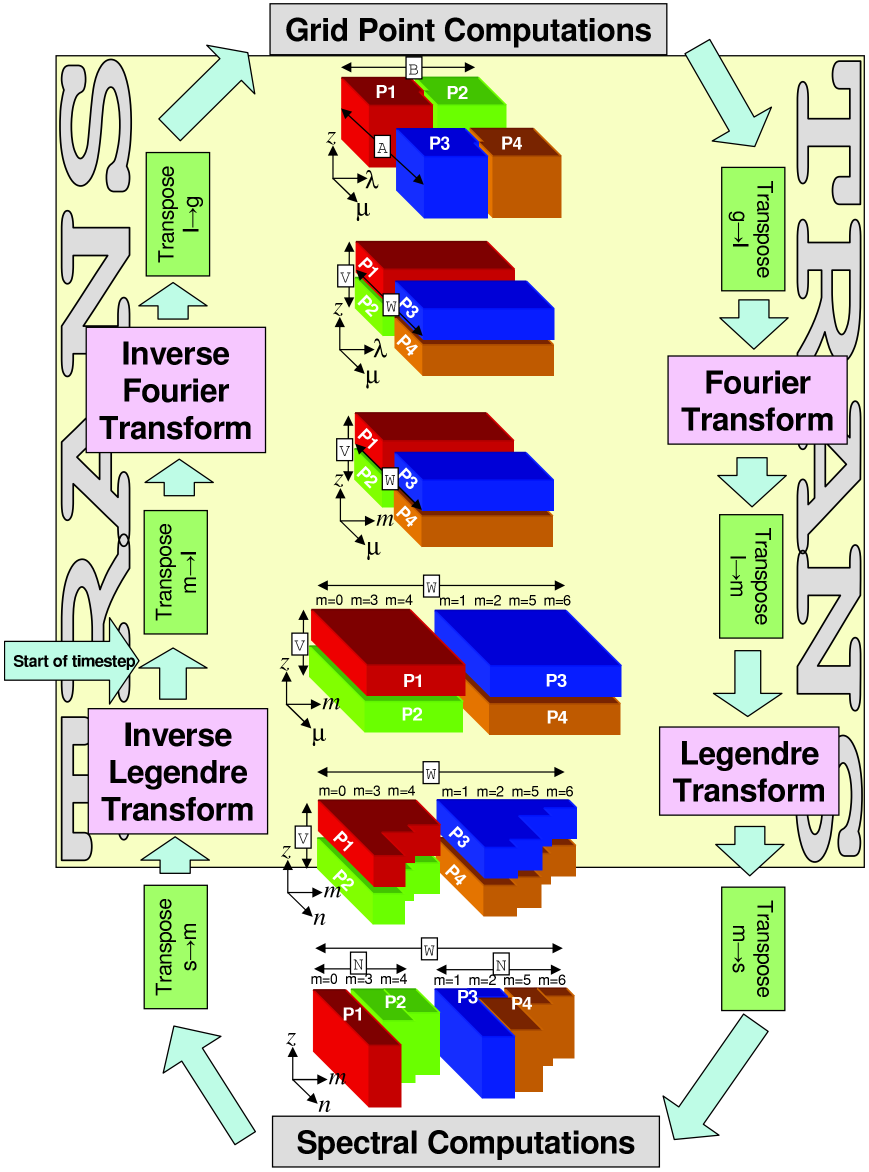
# Parallelization strategy - MPI V.

## Spectral SI calculation

- decomposition along **NPRTRN** = NPRTRV - trivial as there's only vertical dependency for SI, (but might be more complicated for LIMPF=.T.)
- transpositions inside spectral space computation

# Parallelization strategy - MPI VI.

## Summary



# Parallelization strategy - OpenMP

- Parallelize Loops between MPI calls
- High level (all GP computation is done within only 3 OpenMP parallel regions) and Loop level (leftovers like I/O)
- Strong sequential equivalence required to obtain bit-wise identical results - if multiple threads combine results into a single value, sequential order must be enforced (weak SE allowed but optionally only)
- Easy to implement but requires more maintenance to remain thread-save (bugs can lurk unknown)

# Parallelization - MPI+OpenMP

## Pros

- Lower MPI overheads
- Memory saving (if done properly!!!)
- Frees up processors for OS functions
- Helps balancing

## Cons

- Whole code needs to be done (but not comms)
- Need some special care for vector platform (high values of NPROMA requires further optimization w.r.t. number of threads)

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now good for cache

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now good for cache
- NPROMA : Long for vector; short for scalar/cache

# NPROMA

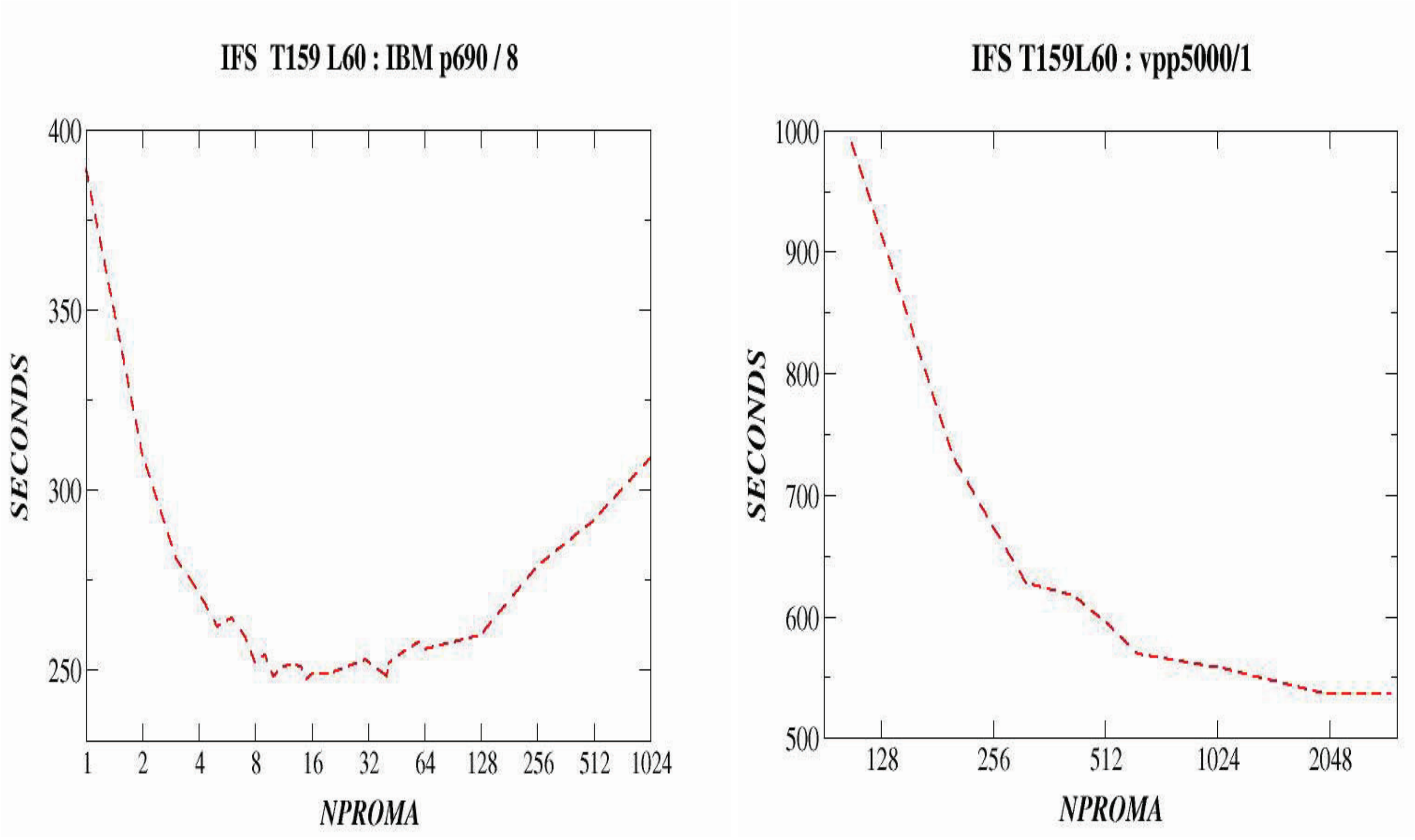
- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now good for cache
- NPROMA : Long for vector; short for scalar/cache
- Memory saving and easy OpenMP implementation

# NPROMA

- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now good for cache
- NPROMA : Long for vector; short for scalar/cache
- Memory saving and easy OpenMP implementation
- Variability of NPROMA allows to keep control over memory conflicts (by over-dimensioning)

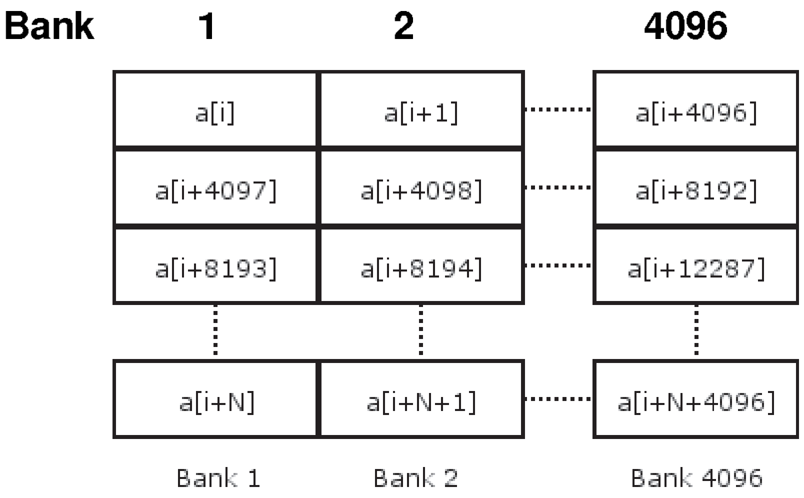
# NPROMA II.

## Illustration of NPROMA influence to model performance



# Memory conflict

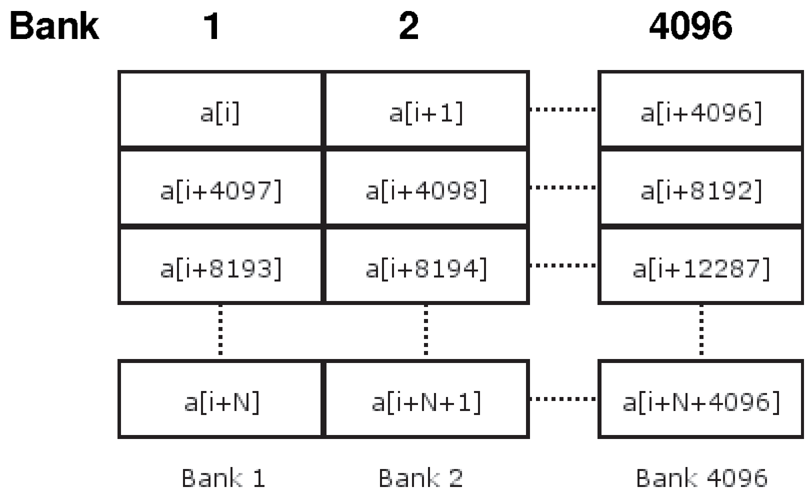
SX Shared Memory with 32 x 128 = 4096 memory banks and vector a(:)




If a(:) becomes A(NPROMA,NFLEVG) and by chance NPROMA=4096. In such case any loop over second dimension will cause bank conflict.

# Memory conflict

SX Shared Memory with 32 x 128 = 4096 memory banks and vector a(:)




If a(:) becomes A(NPROMA,NFLEVG) and by chance NPROMA=4096. In such case any loop over second dimension will cause bank conflict.

$\Rightarrow$ Situation becomes much better when array is over-dimensioned to NPROMA=4097.

# Data structures

## Model arrays decomposition

- usually no decomposition over levels and fields
  `Example for GP arrays:`

  `Model_Data(1:Decomp_2D_Field,1:NFLEVG,1:NFIELDS)`

  $\Rightarrow$

  `Model_Data(1:NPROMA,1:NFLEVG,1:NFIELDS,1:NGPBLKS)`
- various places (GLMS) use different decomposition $\Rightarrow$ transpositions are moving data between processors to form a new decomposition

# Data structures - GP space

## GMV

- prognostic variables involved in the SI
- only attribute is field pointer (MU, MV,...)
- three modules:
  - YOMGV : contain the main GP arrays (GMV, GMVT1, GMV5, GMV_DEPART, GMVS, GMVT1S, GMV5S, GMVS_DEPART)
  - TYPE_GMVS: type descriptor to address the GMV arrays: (YT0, YT9, YT1, YPH9, YT5, YAUX)
  - GMV_SUBS: Contains subroutines used for setting up GMV
- usage (inside parallel regions):

```
DO JLEV=1,NFLEVG
  DO JROF=KST,KPROF
    PGMVT1(JROF,JLEV,YT1%MU)=PGMVT1(JROF,JLEV,YT1%MU)-POMVRL(JROF)
    PGMVT1(JROF,JLEV,YT1%MV)=PGMVT1(JROF,JLEV,YT1%MV)-POMVRM(JROF)
  ENDDO
ENDDO
```

# Data structures - GP space II

## GFL

- all other variables
- can be GP or SP
- plenty of attributes - very flexible field definition through namelist
- ...

# Data structures - GP space II

## GFL

- all other variables
- can be GP or SP
- plenty of attributes - very flexible field definition through namelist
- ...

## SL buffers

| | |
|---|---|
| PB1(NASLB1,NFLDSLB1) | buffer for interpolations |
| PB2(NPROMA,NFLDSLB2,NGPBLKS) | buffer to communicate non lagged to lagged dynamics |

| | |
|---|---|
| NASLB1 | (over) number of columns in the core+halo region |
| NFLDSLB1 | number of fields times vert. dimension in PB1 |
| NFLDSLB2 | number of fields times vert. dimension in PB2 |

# Data structures - Spectral space

- Module YOMSP contains:

  **SPA1(NFLSUR,2)** mean wind (in LAM only)

  **SPA2(NSPEC2, NS2D)** 2D spectral arrays

  **SPA3(NFLSUR, NSPEC2,NS3D)** 3D spectral arrays

- They are not NPROMA arryas!!!

  NFLSUR (over) number of vertical level (bank conflict!)

  NSPEC2 number of spectral coefficients

  NS3D, NS2D number of 3D/2D spectral fields