Development constraints for (Open)IFS

Filip Váňa

filip.vana@ecmwf.int

Outline

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

Basic code rules

- IFS: Over 3 millions lines of code.

Basic code rules

- IFS: Over 3 millions lines of code.
- Coding rules and conventions (last revisited 2011)

Basic code rules

- IFS: Over 3 millions lines of code.
- Coding rules and conventions (last revisited 2011)
- Platform independence - optimised for Scalar and Vector platforms

Basic code rules

- IFS: Over 3 millions lines of code.
- Coding rules and conventions (last revisited 2011)
- Platform independence - optimised for Scalar and Vector platforms
- Parallel code - allows parallel computation, supports MPI and OpenMP standards

Basic code rules

- IFS: Over 3 millions lines of code.
- Coding rules and conventions (last revisited 2011)
- Platform independence - optimised 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

Basic code rules

- IFS: Over 3 millions lines of code.
- Coding rules and conventions (last revisited 2011)
- Platform independence - optimised 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
- Bit reproducibility (with respect to different NPROMA values and different no. of PEs)

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 (FC mode)

Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine (FC mode)
- 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 (FC mode)
- 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 (FC mode)
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organisation (S $\rightarrow$ L^{-1} $\rightarrow$ F^{-1} $\rightarrow$ G $\rightarrow$ F $\rightarrow$ L $\rightarrow$ S)

Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine (FC mode)
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organisation (S $\rightarrow$ L^{-1} $\rightarrow$ F^{-1} $\rightarrow$ G $\rightarrow$ F $\rightarrow$ L $\rightarrow$ S)
- All model arrays are decomposed in the same way.

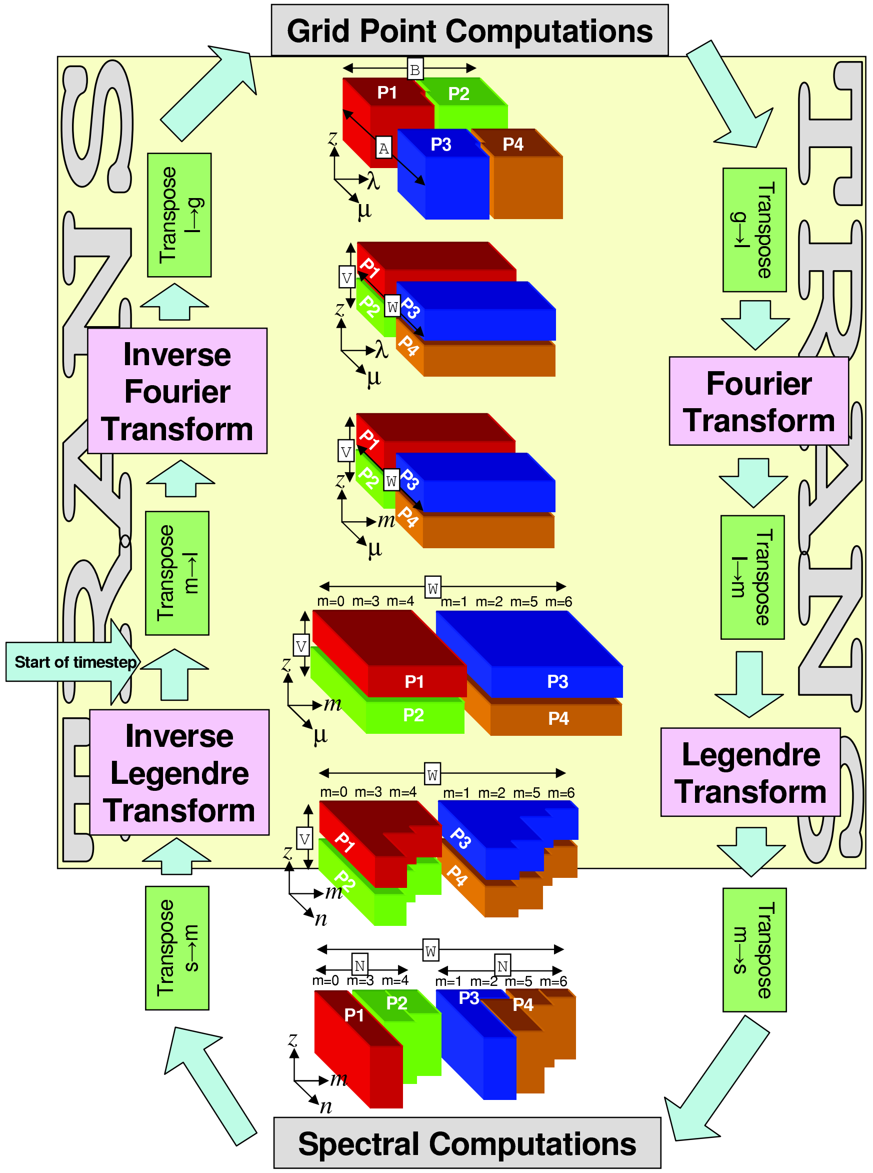
Some more rules...

- Source code written in FORTRAN (F90, F77) and C (soon also C++)
- DGEMM is only standard library routine (FC mode)
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organisation (S $\rightarrow$ L^{-1} $\rightarrow$ F^{-1} $\rightarrow$ G $\rightarrow$ F $\rightarrow$ L $\rightarrow$ S)
- All model arrays are decomposed in the same way.
- No fixed 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 (FC mode)
- Error trapping usable for operational applications
- 64 bit arithmetic and 64 bit addressing
- Spectral model = specific timestep organisation (S $\rightarrow$ L^{-1} $\rightarrow$ F^{-1} $\rightarrow$ G $\rightarrow$ F $\rightarrow$ L $\rightarrow$ S)
- All model arrays are decomposed in the same way.
- No fixed ordering of model fields
- 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 strategy



Parallelization strategy

- MPI = Distributed memory parallelization
- OpenMP = Shared memory parallelization
- Mixed/hybrid MPI and OpenMP parallelization

- Further distribution (for massive computer)
- Use of accelerators

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.
- Transpositions never involve global communication, but only communication within each subset.
- Inter-processor communication is localised in a few routines and rest of the model need have no knowledge of this activity.
- Communication is realised through relatively long messages (1Mbytes)

  (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 cont.

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

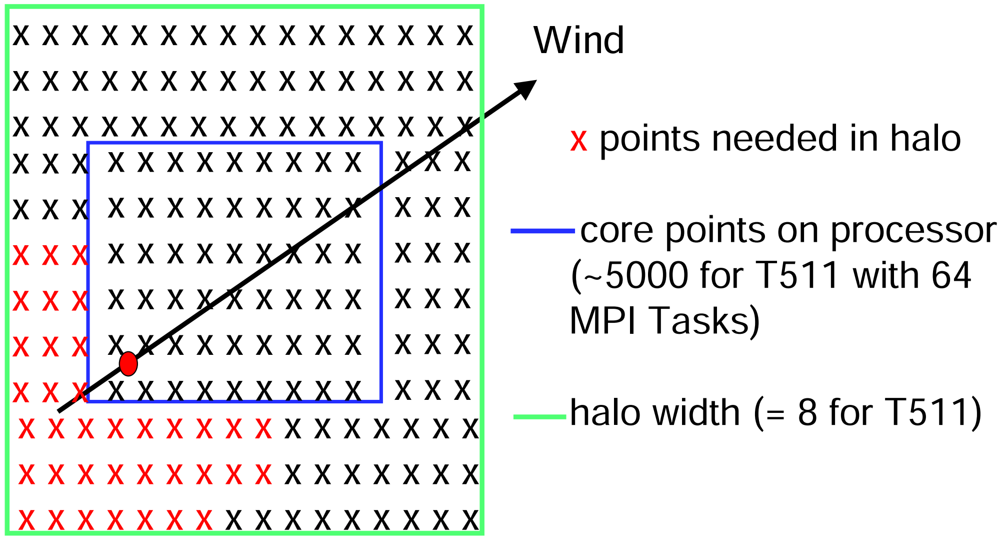
Parallelization strategy - MPI cont.

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


Wind
x points needed in halo
core points on processor (~5000 for T511 with 64 MPI Tasks)
halo width (= 8 for T511)

Parallelization strategy - MPI cont.

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 T511 with and 91 levels reaches scalability limit for transformation at around 511*91=46501 MPI processes. GP decomposition of the same domain (NGPTOTG=348528) with the chunk size NPROMA=10 reaches its limit at around 348528/10 = 34852 MPI processes.)

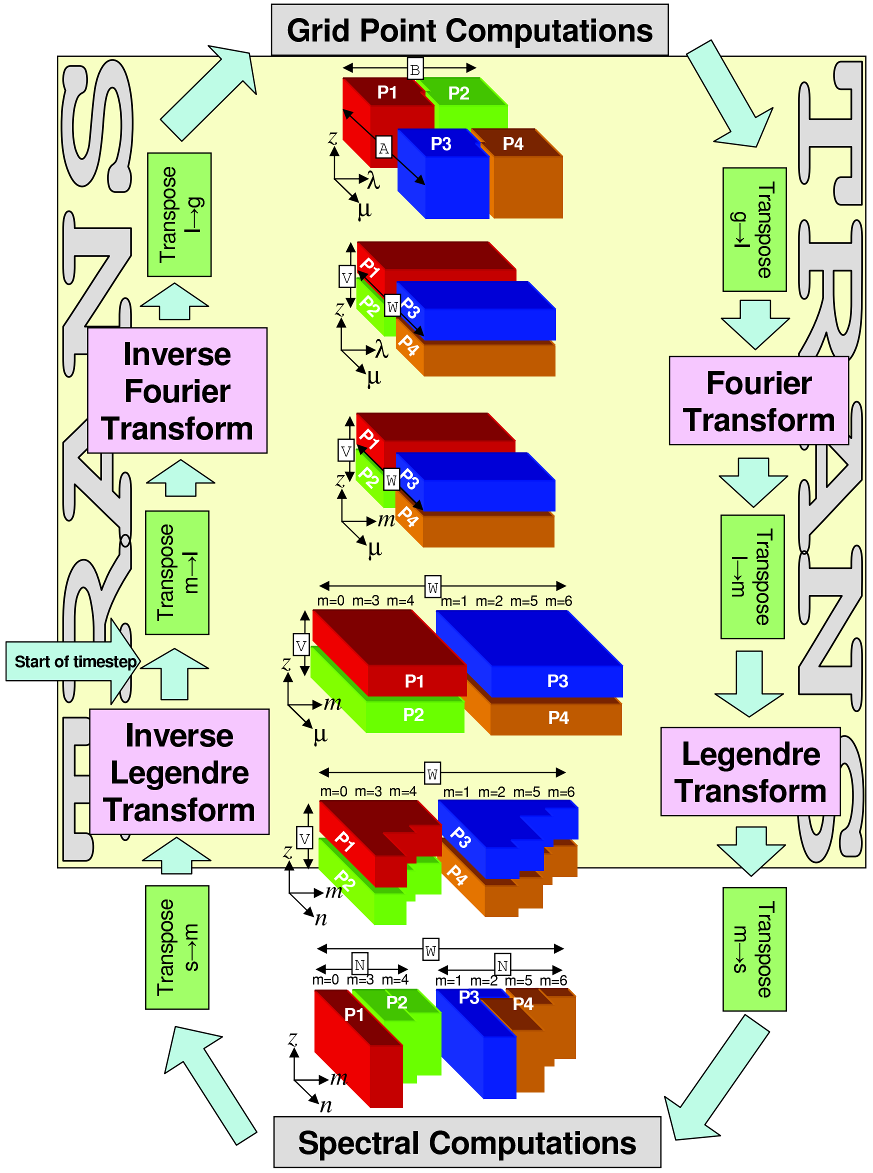
Parallelization strategy - MPI cont.

Spectral SI calculation

- decomposition along **NPRTRN** = NPRTRV - trivial as there's only vertical dependency for SI,
- transpositions inside spectral space computation

Parallelization strategy - MPI cont.

Summary



Parallelization strategy - OpenMP

- Parallelize Loops between MPI calls
- High level (all GP computation processed within only 4 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

Best strategy so far

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

But...

- Deserves no 'critical' regions
- Need some special care with respect to geometry setup when close to saturation limit (NPROMA requires further optimisation 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 GP 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 GP 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 GP Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now proven to be good for cache optimisation

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 GP Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now proven to be good for cache optimisation
- 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 GP Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now proven to be good for cache optimisation
- NPROMA : Long for vector; short for scalar/cache
- So far only two such parameters: NPROMA & NRPROMA

NPROMA

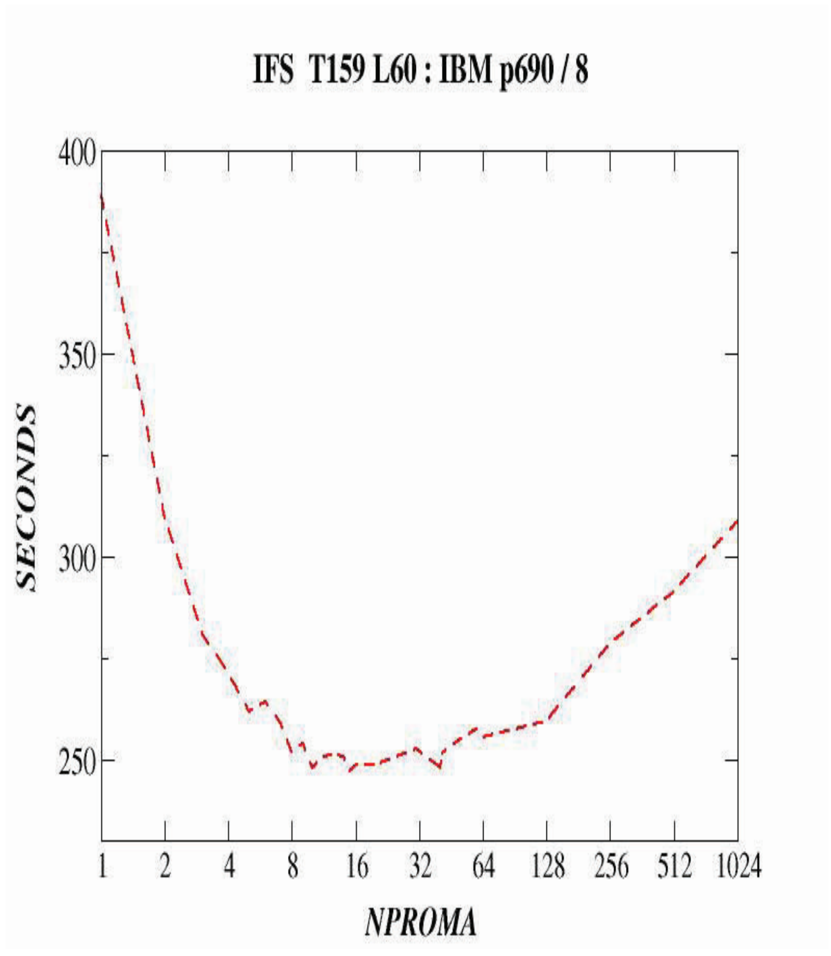
- Original code (designed for vector computers) coded with inner loops over horizontal in groups of NPROMA to give long vectors
- No dependency in horizontal (important for avoiding memory conflicts)
- Physics and GP Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no.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

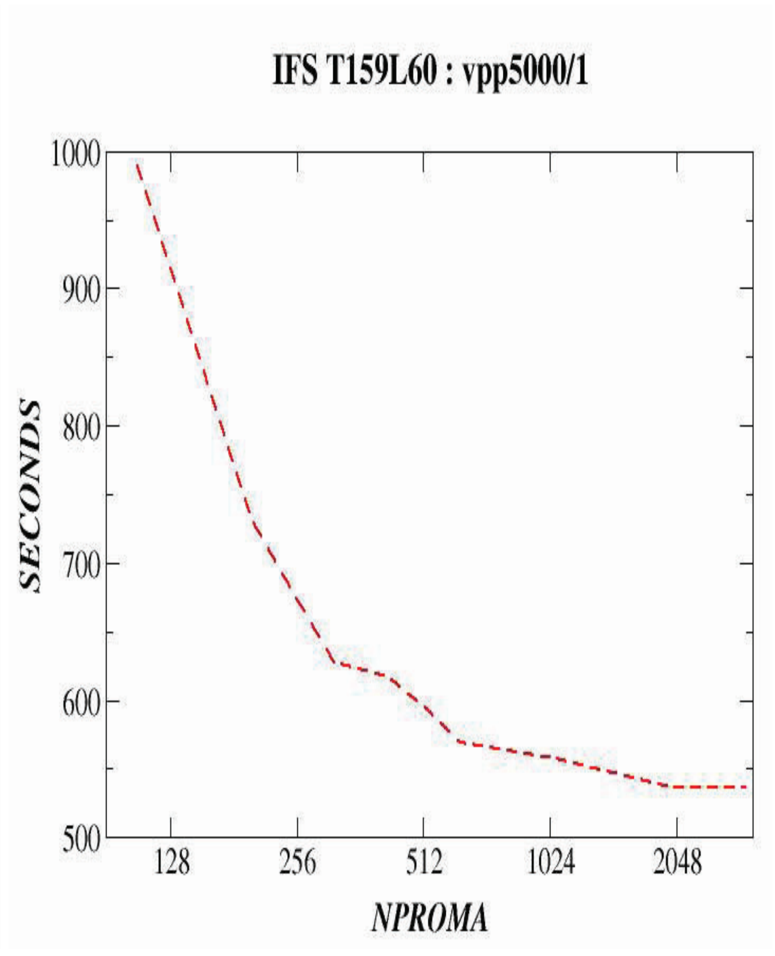
NPROMA

- 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 GP Dynamics computed in blocks of NPROMA
- Bit reproducible with different NPROMA & no. of PEs
- The same design now proven to be good for cache optimisation
- NPROMA : Long for vector; short for scalar/cache
- So far only two such parameters: NPROMA & NRPROMA
- 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





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 (GFLS) 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 arrays!!!

NFLSUR	(over) number of vertical level (bank conflict!)
NSPEC2	number of spectral coefficients
NS3D, NS2D	number of 3D/2D spectral fields

Future code evolution

- IFS = Integrated Forecast System
  Same source code used in forecast and assimilation tasks: One executable allowing for different functionality.

Future code evolution

- IFS = Integrated Forecast System
  Same source code used in forecast and assimilation tasks: One executable allowing for different functionality.
- OOPS = Object-Oriented Prediction System
  - Isolate the data assimilation from the complexity of the model and observation operator
  - OOP - abstract layer in C++ and model specific layer mostly coded in Fortran.
  - One executable for complete set of tasks - saving in I/O handling, extended parallelism,...