Dynamic features of PALM_MP
Needs and definition

Samuel BUIS, Thierry MOREL, Andrea PIACENTINI

CERFACS Technical Report TR/CMGC/05/67 - October 6, 2005

Abstract

Many aspects of the PALM_MP coupler implementation depend on the definition of its dynamic features. This report aims to recall the reasons of implementing a dynamic coupler and to establish the dynamic features that are supported. For each feature it sketches the main properties, proposes runtime checks, and describe the main interactions with existing mechanisms.

Introduction

This document deals with the dynamic features of the PALM_MP coupler. Taking into account the dynamic issues makes the biggest differences w.r.t. to PALM_RESEARCH. Many implementation issues depend on the choices we make about the accessibility and the scope of the dynamic changes and on the mechanisms to operate the changes. Therefore, this document focuses on the definition of the dynamic features. The implementation details of each feature will be presented in following reports.

We summarize the reasons for implementing a dynamic coupler, providing some basic examples to clarify in which context the dynamic features are needed. In the following, we detail the dynamic features at algorithm, unit and data levels. We concentrate in particular on the issues related to the redefinition of data due to the dynamic load of real-time data, because it is the aspect of data assimilation that motivated most of the need of the dynamic data handling. For each aspects concerned by this issue we fill a template that gives the main lines of the software design. They will be explored and detailed in following reports.

1 Reasons for implementing a dynamic coupler

There are three main motivations that lead us to the implementation of a dynamic coupler. The following features have to be granted by the PALM_MP coupler.

1. Algorithm: loops and conditional control.
   The most important feature of PALM is the ability of implementing control structures like IF, SELECT, DO and WHILE in the branch code. With respect to a static coupler like OASIS, PALM does not know in advance if, when and how many times a unit is executed.
   A particular case that should be mentioned, even if the PALM_MP design is not well suited for it, are the replicated branches. We can easily think of a situation in which the same treatment is applied to many independent data sets, as in the SOFA O.I. implementation in which a local independent analysis is performed for each observation point. If the treatment is implemented as a branch, the whole analysis turns out to be the repeated execution of the same branch. In such a case it would be possible to start the branch instances sequentially or in parallel. The number of parallel instances could be dynamically defined.

2. Units: different parallel instances.
   The same code can play different roles in a PALM application. It means that in a single application a code can have different configurations. It includes the possibility of changing the parallel distribution between instances of a unit in order to balance the application. As an example, think of the direct model in an assimilation/forecast suite. The model can be run on more processes during the forecast phase and on less when it runs concurrently with the observations treatment during assimilation. The number of processes allocated to the model during the assimilation phase can dynamically depend on the estimated load of the observations treatment.

3. Data: real-time loading of fields.
   In real-time environments not all the features of the objects to be manipulated are known in advance. In particular, in data assimilation applications the observed data is known only after being received and preprocessed. Assuming that the nature of the data (i.e. the type and kind of the variables used to represent

it) cannot change in an uncontrolled way, we can state that only the space size and the time/tag instances can change. In some particular case (cf. the implementation of the old 3D-Var suite at Météo-France), the parallel distribution and/or the sub-object mapping can change too.

2 Algorithm: loops and conditional control

The PALM branch codes can already handle the main control structures:

- `IF` and `SELECT` instructions for conditional execution.
- `DO` and `WHILE` instructions for repeated execution.

The previous constructions can be nested in any form and can evaluate local variables of the branch they belong to. Exchange of scalar variables between branches and units can be done via the communication mechanism (PALM_Put, PALM_Get).

However there is an issue that PALM can hardly deal with. Let's assume that part of the algorithm is implemented as a sequence of units on a separate branch and that this part has to be replicated *nb* times in parallel, with *nb* dynamically set during the execution. Invoking `PALM_Start` in a loop would result in the sequential execution of the branch instances, since the scheduler won't start a new instance of a running branch until termination of the previous one. The only way to handle this situation is to describe in PrePALM an overdimensioned set of branches (using the merge functionnality as a copy/paste one), and to invoke conditionally the corresponding `PALM_Start`'s. However, this solution is not satisfactory since it leads to complicated and non-adaptable applications. To address this issue, an automated mechanism for the creation of multiple identical branches could be implemented in PrePALM.

3 Units: different parallel instances

PALM_MP can change the number of processes allocated to a unit at each new launch. The number of processes of a unit is set in PrePALM and can be associated to a local variable of the branch it belongs to.

Once a unit has started, it cannot dynamically increase the number of processes it uses. Therefore, PALM_MP does not fully support master-slave type units, even if they are in a block. If such a unit is launched by PALM_MP, only the master will be recognized by PALM_MP.

The current implementation of the PALM_MP process management model implies collective synchronising MPI operations when a unit terminates. Therefore, no processes are released before the end of the unit even if the unit is unbalanced. An example in which this feature could cause a waste of ressources is a unit performing parallel computations on *np* processes, then gathering the results on one process and let it run longer to perform post-processing and output. This aspect could be optimized in a future if necessary.

What can change The number of processes of the unit.

Level The change is invoked by the unit's branch, thus the number of processes of the unit can only be changed between two instances of the unit.

Mechanisms Branch variable used to define the number of processes of the unit in PrePALM : it takes a run-time value at the PALM_Launch

Interaction with other mechanisms Changing the parallel mapping of a unit implies the redefinition of its objects distribution (localisations and distributors). The mechanism for dynamically changing the distributors is detailed in section 4. Concerning localisations, they might be automatically recomputed at the beginning of units to be updated in function of its number of processes. This redefinition of localisations and distributors implies a possible change of the number of communications of the units (in case of replicated objects) and/or of the number of processes taking part in a given communication. The structure of the communication state table must then be updated for a correct storage of the dynamic informations concerning the state of the communications. The structure of the MAILBUFF main data structure must be updated for the same reason.

Caveats For some particular cases, the number of processes of a unit must not be changed. For example, the localisation of the target object of a given tube should not change during the communication treatment (i.e. from the moment on the corresponding PALM_Put has been performed and before the PALM_Get has been called).
The number of processes of a block is defined as the maximum number of processes used by a unit in this block. Thus, all the variables defining the number of processes of the units in a block must have been initialized before

launching this block.
Inside a block, the number of processes of a unit can only be set to a value inferior or equal to the number of processes of the block.

Run-time checks If the number of processes of a unit has not been initialized, and if an entity tries to exchange an object with it, the application must abort. For the user, a solution to avoid this problem is to do this communication through the BUFFER.
The application must abort if a unit in a block is tried to be launched on more processes than the block runs on.

4 Data: real-time loading of fields

As sketched in section 1, several features of PALM_MP are concerned by the real-time loading of fields. In particular we will concentrate on the dynamic change of the spaces, of the distributors, of the time and tag lists and association rules and of the sub-object mappings.
For each item we try to fill the same template:

- What can change and what cannot
- At which level the change can be invoked
- Mechanisms to change the definition
- Interaction with other mechanisms
- Caveats
- Run-time coherence checks

4.1 Spaces

The most common case in data assimilation is the use of observation fields which can differ at every instance of the suite. The simplest change we have to take into account is the change of the size of the field, i.e., in the PALM formalism, the *shape* of the space describing the observation field.

What can change and what cannot The shape of the space can change, the rank and the element size cannot.

The fact that the rank and element size are not allowed to be changed does not come from a major difficulty to implement this feature but more from a wish to not let the user do anything with PALM.

Level The change is invoked by units for their own spaces.

Mechanisms A primitive called PALM_Space_set_shape could be created to change the definition of a space:

CALL PALM_Space_set_shape(cl_name, il_rank, ila_shape, il_err)
with

- cl_name (IN): the name of the space
- il_rank (IN): the space rank
- ila_shape (IN): an array containing the new space shape
- il_err (OUT): an error code that take a non-zero value in case of error in the space name, ...

This primitive will inform the driver about the new definition of the space. Some aspects are still to be studied for its use in parallel unit: should it be collective ? synchronised ? In any case, spaces will have to be defined/updated before being used by the PALM_Put and PALM_Get primitives. Nevertheless, this issue could be of the responsability of the user.

A primitive to get the current value of the shape of a space could be useful for inherited spaces:

CALL PALM_Space_get_shape(cl_name, il_rank, ila_shape, il_err)
with

- cl_name (IN): the name of the space
- il_rank (IN): the space rank
- ila_shape (OUT): an array containing the space shape
- il_err (OUT): an error code that take a non-zero value in case of error in the space name, ...

This one will ask the driver for the current shape of the space.

Interaction with other mechanisms The remapping mechanism computes the intersection between the distributors of the source and target objects of the communications. Until now, these computations were done at the very beginning of the execution of the PALM_MP driver. For the sake of genericity, even the SINGLE_PROC-SINGLE_PROC remapping must be performed in order to store the characteristics of the data to exchange in the adapted data structure (Distributed Object Representation). This computation needs the characteristics of the space of the concerned objects. Thus, in case of dynamic space, the intersection computation must not be computed at the very beginning of the execution but only when the characteristics of the space are known. However, it should not be performed in the PALM_Space_set_shape primitive but in the PALM_Put/PALM_Get ones. Indeed, this computation can also depend on the dynamicity of the source and target distributors, and, in this case, the knowledge of the characteristics of these distributors depends on the state of the communication.

Caveats The inheritance of dynamic spaces poses the question of synchronisation between the definition of the space in the "parent" unit and its use in the "child" units. By definition, the characteristics of an inherited space can not be changed within a unit (the name of this space is NULL). Thus, the characteristics of an inherited space at a given time will be the current ones of the "parent" space. This OBLIGES to synchronise the execution of "parent" and "child" units so that the concerned space is not modified by the "parent" unit before the "child" ones have accomplished their tasks. This point will be detailed in the report dedicated to the implementation of dynamic spaces.
Can different instances of a same space be simultaneously defined within a unit ? This question seems to be relevant for replicated objects. However, the conceptual problems (defining a localisation for a space does make non sense) and the data coherence problems it leads to may hardly be solved. Moreover, the case of objects that have different shapes according to the process number of the unit they belong to can be seen as an incorrect use of replicated object. In practice, this issue should be treated using distributed objects. That's why this problem might not be considered.
Can spaces that define data structures be dynamic ones ? In practice, the shape of these spaces will be modifiable as those of the other dynamic spaces. However, one can wonder if the shape of the fields of the structure they define can change. This might not be possible for the data structures defined in an old style way (i.e. like in PALM_Research, by defining the type and length of

the field through the element size of the space) since the element size of a space is not allowed to be changed. Nevertheless, it could be possible for the data structures defined with the new style (i.e. using -item) by changing the shape of the spaces of the items.

Run-time checks An object must be exchanged between two entities only if the source and target spaces are compliant. This is verified in PrePALM for static spaces and must be checked runtime for dynamic ones. To implement this check, the size of the associated space should be attached to the objects stored in the MAILBUFF and in the BUFFER.

4.2 Distributors

If the size of a space changes, the parallel distribution of the related objects changes too. This feature can be useful also in case of parallel redistributions with constant size (namely for load balance purposes) or if the number of processes of a unit changes. Therefore it has to be considered also independently of the space size changes.

What can change and what cannot All the data contained in a distributor can change, except its type, its rank, and the name of the distribution function. Moreover, one can recall that a distributor is associated to one or several objects through communications, and that these objects must have the same rank (all the data contained in the distributor have this rank) and shape.

Level The change is invoked by units.

Mechanisms To change the definition of a distributor, the primitive PALM_Distr_set could be implemented.

CALL PALM_Distr_Set(cl_name, il_rank, ila_shape, il_distr_nbprocs, il_array_sz, ila_array, il_err)
with

- cl_name (IN): the name of the distributor
- il_rank (IN): the rank of the distributed object
- ila_shape (IN): the shape of the distributed object
- il_distr_nbprocs (IN): the number of processes of the distributor
- il_array_sz (IN): the size of ila_array

- ila_array (IN): an array to let the user pass its own informations to the distribution function
- il_err (OUT): an error code.

As for the PALM_Space_set primitive, one can wonder if the PALM_Distr_set primitive must be collective and/or synchronised. Indeed, it must be ensured that the distributor of an object has been defined/updated before the object is send/received. This point will be detailed in the report dedicated to dynamic distributors.

Interaction with other mechanisms The dynamic definition of distributors has of course huge consequences on the remapping mechanism. In particular, it does not affect the way the intersection between distributors are computed, but the place where it is done and the moment when it is done. For example, if the target distributor of a given communication is dynamic and if the corresponding PALM_Get has not been posted when the PALM_Put is done, the entire object must be send to the MAILBUFF, where the computation of the distributors intersection will be performed. Indeed, in this case, the real value of the target distributor to use in the remapping will not be known until the PALM_Get is posted. This implies to attach the corresponding distributors characteristics to the objects in the MAILBUFF and in the BUFFER.
Depending on the case, the computation of the distributors intersection could be performed in the MAILBUFF, in the BUFFER, and / or in the PALM_Put and PALM_Get primitives. A mechanism to ensure the validity and coherence of the version of the distributors used to compute these intersections will be studied.

Caveats Distributors are private to the unit they belong to, which avoid conflicts of definition. However, the distributors defined in units can be used to describe the distribution of objects in the BUFFER. In this case, when an object is put in the BUFFER, the characteristics of its target distribution will be the current ones of the given distributor.
The PALM_MP algebra toolbox now implements parallel units. The description of the data distribution of these units is automatically computed from static informations given by the user in PrePALM. A mechanism to provide runtime these informations is still to be studied in order to make their distributors become dynamic ones.

Run-time checks Each time a distributor will be changed the function that tests the coherency of a distributor will have to be called.

4.3 Times and tags

To avoid confusion, it is important to recall what *Time and tag list* and *Time and tag association* stand for. In the PALM protocol, a communication is defined by the description of the objects made available on the source side, by the description of the objects required on the target side and by the association between production and consumption. In PALM_RESEARCH the time stamps and the tags of the produced and the required objects should necessarily be equal and therefore they were described only at the moment to describe the tube. In PALM_MP we introduce another degree of freedom: the time stamps and the tags on the source and on the target side can differ (under some coherence conditions) and the user provides an association rule to match the source and target side. Therefore we are in the presence of three different items which can change dynamically: the list of the produced time stamps and tags, the list of the required time stamps and tags and the rule to associate them.
A mechanism to dynamically change the source and target times and tags of a communication has already been tested. It consists in defining time/tag lists in units using a new primitive. These time/tag lists can then be used by branches and could be used in communications as source and/or target time/tag list. This mechanism does not allow to change the association rule but might be sufficient for the treatment of most of the applications needing dynamic times.

What can change and what cannot The source and target time/tag list of a communication, and their association rule.

Level and mechanism A primitive could be created to define time/tag lists in units. A mechanism could then be implemented to use these lists in the time-tag functions.

Interaction with other mechanisms The dynamic change of time and tag list can lead to a modification of the number of communications of a given tube. The structure of the communication state table must then be adapted to this change.

Run-time checks

4.4 Sub-object mappings

Even if the low level handling of sub-objects relies on the same mechanisms than the remapping, the sub-object are conceptually closer to the time/tag association rules. The sub-objects provide a mapping between the location of data in the source side representation and the location in the target side representation. The definition of sub-objects depends on the applications and therefore is given in PrePALM (for more information about sub-objects, see "SUB-OBJECTS IN PALM-MP", Samuel BUIS, CERFACS Technical Report, WN-CMGC-02-80).
A sub-object is directly dependent on the definition of the object it is associated to. Thus, if the definition of the space of this object changes, the definition of the sub-object has to be updated. Moreover, it could also be useful to dynamically modify the definition of sub-objects even if the space of the object it is associated to does not change. Let's consider for example the following application: a unit provides a matrix which columns must be treated the one after the other, in a loop, by the other units of the application. A sub-object should be used to extract the columns from the matrix, but the definition of this sub-object must be updated at each iteration of the loop although the space of the matrix does not change.

What can change and what cannot All the data contained in a sub-object can change except its rank.

Level and mechanism This point must be studied. As said before, sub-objects must be updated if the space they are associated to changes. Thus, we can imagine to automatically update sub-objects when a PALM_Space_set_shape primitive is called. However, this mechanism does not allow to treat the case previously mentionned. A primitive PALM_So_set to call in branches may partly solve this problem.

Interaction with other mechanisms The dynamic change of sub-objects has the same consequences than the dynamic change of spaces or distributors: it influences the remapping mechanism. The intersection between the distribution of two objects can not be computed if their sub-objects have not been defined.

Run-time checks Each time a sub-object will be changed the function that tests the coherency of sub-objects will have to be called.

Conclusion

This study recalls the needs of dynamic features in PALM_MP and gives an overall point of vue of the problems to be considered for their implementation. This implementation might be particularly tricky since it has numerous interactions with already implemented mechanisms and since it inevitably poses some problems of synchronisation and data coherence. Moreover the dynamic features have not been experimented in PALM_Research. Thus, a thorough study of each one of the dynamic functionnalities needs to be carried out on the basis of this document before beginning their implementation.