Distributed data structures and Summarized

parallelism:

Contents

1 Qu'est-cequ' a data structure distributed?

Currently, the code proposes three strategies to function in parallel via MPI:

- Pre and/or postprocessings using of elementary computations (e.g. with the operator *Aster* CALC_CHAMP),
- Construction, assembly and resolution of the linear systems with the external products MUMPS or PETSc (e.g. STAT_NON_LINE). One solves here in parallel the usual total problem: one is in mono-field.
- The same thing as previously but with an internal linear solver multi-fields: FETI (e.g. MECA_STATIQUE).

These three strategies of parallelism implement a distribution of the tasks and associated data. I.e. the processors will allocate usual data structures, them to initialize but they will fill them only partially. To facilitate the maintenance and the legibility of the code, one generally does not recut these data structures with just. They thus comprise many zero values. To detect the complete or incomplete character such SD, one of their attribute take value **MPI_COMPLET** or **MPI_INCOMPLET**.

Note:

For more information on these tools (usable also into sequential), one can consult R the handbooks of Reference 6.01.02/03 and R6.02.03, as well as the instruction manual U4.50.01.
To gain in performance, the code proposes two other complementary strategies of parallelism: in memory shared (OpenMP *) with internal linear solver* MULT_FRONT *(R6.02.02) and, with the tools* ASTK *which make it possible to launch independent studies.*

2 How is made the distribution?

Code_Aster being a code finite elements, the distribution of data put in work to control parallelism is a distribution by mesh (*"EOD"* for *Element Oriented Distribution*). Each processor is thus responsible for a package for meshes (physical or late) and will thus carry out only elementary computations, the assemblies and the data structure fillings corresponding

This distribution of meshes between the processors is done on the level of the assignment of the model (command AFFE_MODELE). According to the mode of distribution chosen by user (MAIL_DISPERSE, MAIL_CONTIGU.) and the number of processors allocated for the job, the code operates distribution MAILLE/PROC and stores it in data structure PARTITION. This distribution can be besides modified in the course of computation *via* command *MODI_MODELE*.

Note:

Meshes tar divine: Meshes which does not exist in the initial mesh and which is added during computation to facilitate the setting in data. One generally meets them with the conditions of Dirichlet imposed with AFFE_CHAR_MECA or when one uses the continuous method of contact
to facilitate the management of late meshes those are automatically assigned with the main processor. This extra work generally induces weak a déséquilibrage of load. In the contrary case, the user can meshes restore the equilibrium by discharging this processor (via key word CHARGE_PROC0_MA/SD) from a certain percentage from physical which are initially allotted to him (or of a certain number of subdomains if the distribution is directed subdomains).

3 What are they the data structures concerned?

For time one "distributes" only the SD resulting from an elementary computation (*RESU_ELEM* and *CHAM_ELEM*) or from a matric assembly (*MATR_ASSE*). Vectors (*CHAM_NO*) are not distributed because the induced gains report would be weak and, in addition, as they intervene in the evaluating of many algorithmic criteria, that would imply too many additional communications. Characterization *MPI_COMPLET* or *_INCOMPLET* intervenes in:

- Does CELK (7) for *the CHAM_ELEM* (D4.06.05 – data structure field),
- NOLI (3) for *the RESU_ELEM* (D4.06.05 – data structure field)
- *REFA (11) for the MATR_ASSE* (D4.06.05 – data structure sd_matr_asse),

4 How handle one such data structures?

Since these data structures preserve the same organization and the same dimensions that their sequential version, one can handle them with the utility routines standards. However, for each total processing (i.e. implying all potentially meshes model), it is necessary to detect their complete or incomplete character (*via* the flags of the preceding paragraph) and to adopt an ad hoc *strategy*.

For example, before a product matrix-vector one can supplement *the MATR_ASSE* so necessary (*via sdmpic*) or decide to stop in *ERREUR_FATALE* if this scenario corresponds to an unforeseen software advance. In other cases, one does not seek to supplement the SD but one establishes communications MPI corresponding to the total processing concerned (e.g. search for maximum as in *assmam*).

Note:

To facilitate maintenance, the installation and the validation of the code, one limits calls MPI to some utility routines: mummpi, fetmpi and mpicm1 . If one wants to develop new communications, it is better thus to enrich one by these routines
sequential Version: One should in any rigor qualify them centralized. Because even a parallel computation can lead to MATR_ASSE or not distributed RESU_ELEM. It is enough for that the user chooses, for all computation or right for a portion, the parallelism of the type CENTRALISE in operators AFFE/MODI_MODELE . Elementary computations and the assemblies are then not paralleled, only the linear solver is (MUMPS or PETSc).

5 Some cas particuliers (FETI, MATR_DISTRIBUEE)

1) In the case of FETI solver , it is natural that certain data are incomplete. It is for example the case of the matrixes and the second local members assembled by *assmam* and *assvec*. There is no communication particular to establish. Each local linear system with a subdomain constitutes others indépendemment. On the other hand, as during the elementary computation options which generate *the specific* RESU_ELEM to these subdomains, one imposes, to gain in performance (one anticipates the needs and one groups various kinds of elementary computations), also the construction of some *CHAM_ELEM*, it is necessary to supplement the latter (*via sdmpic*). With this intention one "tagge" beforehand these SDs in *MPI_INCOMPLET* (in *alrslt*).

2) When one the MUMPS solver carries out a computation distributed with , if option MATR_DISTRIBUEE is activated , the specific ends of MATR_ASSE to each processor are recut so as not to store of zero useless (REFA (11) is then taggé in MATR_DISTR) . But as a result the handling of these assembled matrixes distributed "thinned down" is more complex. During certain total processing (produced matrix-vector), they are not supplemented and one stops in ERREUR_FATALE . It will be necessary to enrich the perimeter by use of this functionality step by step.

Note:

Decomposition in subdomains leans on the data structure (D4.06.21 – Data format SD_FETI). To distinguish the data associated with each subdomain, each one comprises a SD_SOLVEUR , a NUME_DDL , a MATR_ASSE and CHAM_NO S adapted to its dimensions. In addition, each processor is responsible for a SD_SOLVEUR , a NUME_DDL , a MATR_ASSE and CHAM_NOs Masters which will be quasi-vacuums and whose only function is to point towards the SD slaves of the subdomains of which the processor with management. It is another form of SD distributed which thus implies this recursion on two levels.

5.1 Regulate to respect during the programming of flood of data/parallel processing

At the end of the operator *Aster ,* all the global databases of the processors must be identical because it is not known if the operator who follows in the command file envisaged an incomplete flood of data input. It is thus necessary to organize, for example, the completeness of ad hoc *fields* in the routines of archivage (while trying its to base on MPI_ALLREDUCE simpler and more effective to implement). To offer a more effective parallelism it will be necessary one day to make jump this bolt. But by then!