# Note of use of parallelism

**Summary:**

Any simulation *Code_Aster* can profit from the performance profits which parallelism gets. The profits can be of two types: over the computing time and necessary RAM/disque space (by heart allocated).

*Code_Aster* propose various parallel strategies to adapt under investigation and to the platform of calculation. Some are rather centered on data-processing aspects (distribution of calculations complete or of independent modal calculations, construction of linear systems, basic operations of linear algebra), others are more algorithmic (linear solveurs HPC MUMPS and PETSc).

This document briefly describes the organization of parallelism in the code. Then he points out some fundamental in order to help the user to benefit from these parallel strategies. Then one details their implementations, their perimeters of use and their profit potential. The chainings/office pluralities of various strategies (often natural and parameterized by default) are also approached.

The user in a hurry can from the start refer to the chapter 2 (“parallelism in some clicks! ”). He summarizes the procedure to implement the recommended parallel strategy by default.

**Note:**

*To use Code_Aster in parallel, (at least) three cases can arise:*

- *One has access to the centralized machine Aster and one wishes to use the interface of Astk access,*
- *One carries out calculations on a cluster or a machine multi-hearts with Astk,*
- *Idem that the preceding case but without Astk.*

# Contents

# 1 Parallelism in some clicks!

## 1.1 Why?

Often a simulation *Code_Aster* can **to profit from important profits of performance** by distributing its calculations on several hearts of a PC or one or more nodes of a centralized machine.
One can **to gain in time** (with parallelism MPI and OpenMP parallelism) as in **memory** (only *via* MPI). These profits are variable according to the requested features, their parameter settings, the data file and the software platform used: cf figure 1.1.

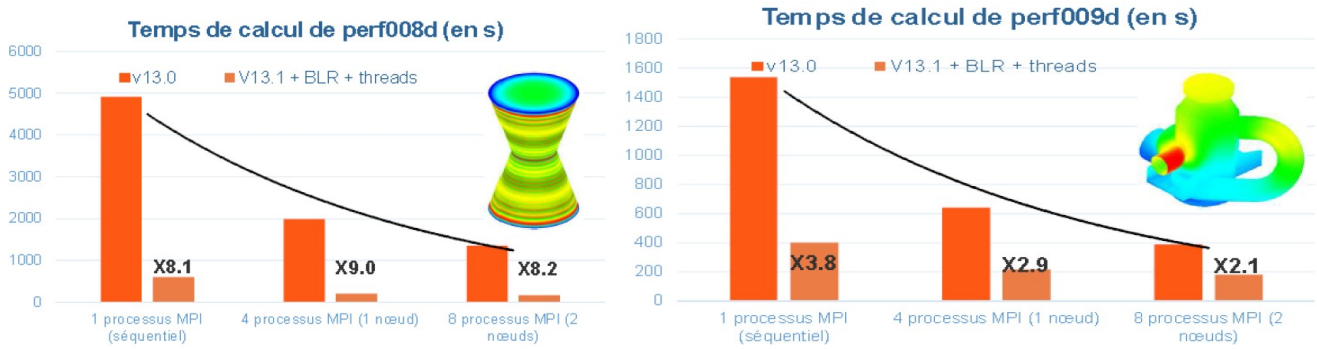


*Figure 1.1. _Example of savings of time gotten by parallelism MPI of Code_Aster v13.0, and with that hybrid MPI+OpenMP (+ compressions low-rank cf [U4.50.01]) of Code_Aster v13.1. Comparisons carried out on the CAS-tests of performance perf008/9d and the centralized machine Aster5.*

## 1.2 How?

In *Code_Aster,* by default, calculation is sequential. But one can activate **various strategies of parallelization**. Those depend on the stage of calculation considered and the selected parameter setting. They are often chaînables or cumulative. One can thus initiate parallel diagrams comprising up to 3 overlapping levels of parallelism.
There are three big classes of problems parallélisables, the second being most current:

- that is to say simulation can be organized in several **independent subcalculations** ,
- either it is not the case but:
  - this one remains **dominated by or not linear linear calculations** (operators STAT/DYNA/THER_NON_LINE , MECA_STATIQUE ...),
  - this one remains **dominated by modal calculations** divisible under frequential bands ( INFO_MODE/CALC_MODES+' BANDE' ).

To have one **estimate of the time spent by an operator** and thus of the prevalent stages of a calculation, one can activate the keyword MESURE_TEMPS orders DEBUT/POURSUITE[U1.03.03] on a standard study (possibly shortened or expurgated).
In all the cases, one will advise **to divide largest calculations into various stages** in order to separate those purely **calculative**[1], those concerning of **postings**, **postprocessings** and of **handling of fields**[2].

**In the first case**, one will refer to the dedicated use of the tool Astk (cf. §4.1 and [U2.08.07]).

1 Possibly of various types (n°2 category or n°3 quoted previously) and which will gain with being carried out in parallel.
2 Who will be often faster into sequential because of the risks of cloggings at the time of the accesses report.

**In the second case**, it is necessary to start by locating the resolutions of systems linear in the command file (keyword SOLVEUR) and to modify their parameter settings in order to use one linear solvor HPC [U4.50.01]. With this intention, one specifies the value 'MUMPS'or 'PETSC' with the keyword METHOD.
One distributes then, on processes MPI, the stages of construction and those of resolution of linear systems (cf. §4.2/5.2/5.3). That makes it possible to lower the level of necessary memory and to accelerate simulation.
One can add a second level of parallelism while using, for each process MPI, several threads OpenMP (cf. §4.3). This second level accelerates, on the other hand, only part of the resolution of the linear systems and it does not make it possible to lower the peak in RAM memory.

**In the third case**, one distributes modal calculations on various blocks of processors (cf. §4.4), then, by using the linear solvor MUMPS, one can add a second level MPI of parallelization within each one of its under-blocks (cf preceding case). One can possibly add a third level by activating threads OpenMP within each process MPI of resolution (cf. §4.3). But this scenario is seldom productive. It is to better hold these hearts with a broader distribution in sub-bands.
Let us note that the effectiveness of this strategy requires frequential bands balanced rather well. To gauge these bands, it is advised to use the operator beforehand INFO_MODE[U4.52.01]. Also profits from a parallelism MPI on two levels very performing.

## 1.3 In practice via Astk

**For the implementation effective of the n°2 cases and n°3**, it is necessary to preselect a parallel version of *Code_Aster* (noted *** _mpi), then to specify the number of hearts selected (finely Options of A*stk*) *via* fields following:

- (optional) **ncpus=k**, number of threads allocated in OpenMP; generally used in complement of MPI; parameterized value by default if one filled by the field and that there remains empty.
- **mpi_nbcpu=m**, many allocated processes MPI.
- **mpi_nbnoeud=p**, many nodes on which will be distributed these m X K parallel calculations.

One advises in general **not to allocate all the hearts of a node in MPI alone** . That can cause of **to slow down simulation** because, even if part of calculations is some accelerated because of its distribution on more hearts, as those certain resources memory divide, accesses to the data , they, are slowed down.
To use more effectively and to 100% all the allocated resources one advises rather **to blend and to balance parallelisms MPI and OpenMP** (hybrid parallelism on 2 levels).

If calculation is rather expensive in the digital phase of factorization of MUMPS (the most frequent case, cf [U1.03.03]) and if the accesses disc do not penalize too much the use of its management memory OUT-OF-CORE (use on centralized machine Aster), one can even privilege the second level. It is often the most powerful strategy: for example for a model comprising at least 1M ddls, one can allocate 6 processes MPI, each one of them deploying 12 threads OpenMP: hybrid parallelism called "6MPIx12OpenMP". *Via* the Astk tool, that will be carried out, if the nodes reserved for parallel calculation consist of 24 hearts[3], while positioning: ncpus to 12, mpi_nbcpu with 6 and mpi_nbnoeud with 3.

**To remain effective** , it is of course necessary to take care of **not to exceed the physical capacities platforms** :

- not more threads OpenMP (ncpus) that hearts sharing a physical memory (12 even 24 on Aster5);
- not more process MPI (mpi_nbcpu), possibly multiplied by the preceding number of threads, that physical hearts available: by node (24 on Aster5) and on the whole (size of the machine);

3 C'is the case of the current centralized machine, Aster5.

- to respect the constraints of the administrative batch and, in particular, the resources maxima allowable by only one user.

**Note:**

- *For only parallelism MPI of the stages of handling of linear systems (second case) it is not useful to allocate too much process. In general, granularity of 1 process MPI for 30 or 50. $10^3$ ddls is largely sufficient.*
  *If one wishes to continue to accelerate calculation, one can initiate a second level of parallelism and reduce this granularity to a few thousands of ddls by threads. For example, a comprising model 0.6M ddls will be able to most probably profit from an effective parallelism by allocating 12 MPI X 5 OpenMP=60 hearts.*
- *For `INFO_MODE` or `CALC_MODES` (third case), one starts by distributing the sub-bands, then one takes account of the possible parallelism of the linear solvor (if `MUMPS` ). For example, for a modal calculation comprising 8 sub-bands, one can pose `mpi_nbcpu=32` . Each sub-band then will use `MUMPS` on 4 processes MPI. And on each one of these independent resolutions one can require the preceding parallel diagram. That is to say potentially three overlapping levels of parallelism.*

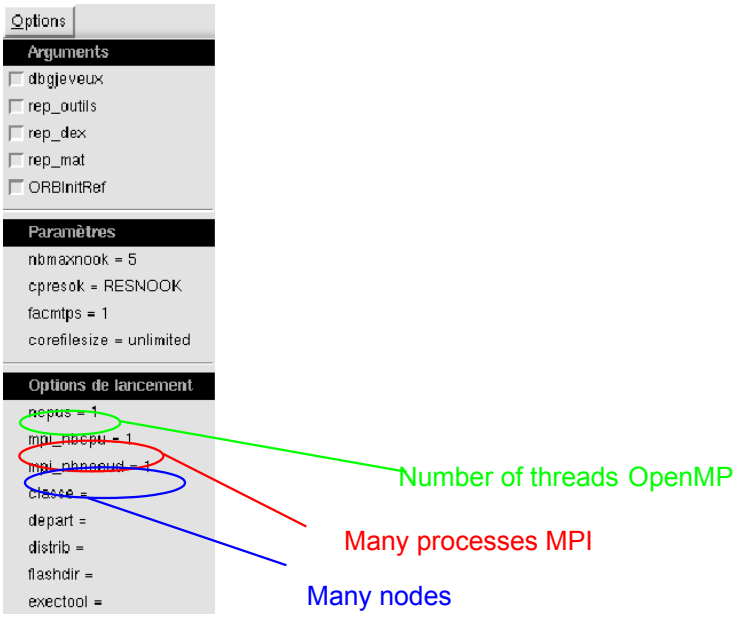


*Figure 1.2. _Parameters of Astk dedicated to parallelism.*

The chapters of this document detail all these elements. Once those fixed, one can launch his calculation as one would do it into sequential (on the centralized machine, in batch mode only). Except, that with parallelism, one can of course reduce the specifications in time and memory of calculation informed in Astk (cf [U1.03.03]).
For example, in the second case, the fact of distributing calculation on 12 processes MPI generally makes it possible to decrease by at least:

- a X2 factor its peak RAM report (by process MPI, cf field ' `Total memory` 'of Astk),
- a X3 factor its execution time (time known as 'elapsed' or 'effective' latency back from calculation, cf field ' `Time` 'of Astk).

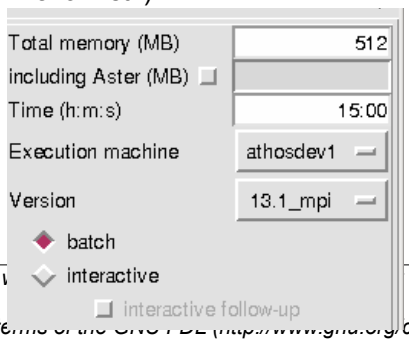

*Figure 1.3. _Parameters of Astk devoted to the resources time and in RAM memory allocated (for each process MPI).*

# 2 General information

**Any simulation *Code_Aster* can profit from the performance profits which parallelism gets.** Since it carries out elementary calculations/assemblies, resolutions of systems linear, large modal calculations or independent/similar simulations. The profits can be of two types: over the computing time and necessary RAM/disque space (by process MPI).
As most codes general practitioners in mechanics of the structures, *Code_Aster* propose various strategies to adapt under investigation and to the platform of calculation. Once parameterized, the majority are connected and coupled in an automatic way. The paragraphs following make the synoptic one.

## 2.1 Data-processing parallelisms

- **1a/ Launching of slopes of independent/similar calculations** (parametric calculations, tests…).
  *Tool:* scriptage Shell.
  Profit: in time elapsed.
  *Launching:* standard *via* Astk [U2.08.07].
  *Chaining*: none.
  *Office plurality*: possible with the other parallel diagrams but only of advanced use (overload of sources).

- **1b/ Distribution of elementary calculations and the assemblies** matric and vectorial in the pre ones/postprocessings and linear constructions of systems. Parallelism in distributed memory.
  *Tool:* language MPI.
  Profit: in time elapsed, even in peak RAM report with `MUMPS+MATR_DISTRIBUEE`.
  *Launching:* standard *via* Astk (`mpi_nbcpu/mpi_nbnoeud`).
  *Chaining*: useful with 2b or 2c; possible but not very useful with 1c or 1d only; possible but useless with 2a.
  *Office plurality*: none.

- **1c/ Distribution of calculations of linear algebra basic** (under-stages of MUMPS, library BLAS)**.** Parallelism in memory shared activated only with package MUMPS (cf 2b)**.**
  *Tool:* OpenMP language.
  Profit: in time elapsed (but increase in time CPU).
  *Launching:* standard *via* Astk (`ncpus`).
  *Chaining*: possible but not very useful with 1b only.
  *Office plurality*: against-productive with 2a, useful with 2b, possible but not very useful with 2c or 1d.

- **1d/ Distribution of modal calculations** (resp. modal calibrations) in the operator **`CALC_MODES`** (resp. **`INFO_MODE`**). Parallelism in distributed memory.
  *Tool:* language MPI.
  Profit: in time elapsed (but increase in the peak RAM report to be counterbalanced by the office plurality with 2b).
  *Launching:* standard *via* Astk (`mpi_nbcpu/mpi_nbnoeud`).
  *Chaining*: possible but not very useful with 1b only.
  *Office plurality*: useful with 2b (even 2b/1c); possible but not very useful with 2a; impossible with 2c.

## 2.2 Digital parallelisms

- **2a/ Direct Solvor `MULT_FRONT`**; Parallelism in shared memory.
  *Tool:* OpenMP language.

Profit: in time elapsed (but increase in time CPU).
*Launching:* standard *via* Astk (ncpus).
*Chaining*: possible but not very useful with 1b, 2b or 2c.
*Office plurality*: against-productive with 1c, possible but not very useful with 1d.

- **2b/ Package MUMPS** (either as a direct solvor, *via* METHODE=' MUMPS' , that is to say as a preconditionnor of PETSC/GCPC *via* PRE_COND=' LDLT_SP' ). Parallelism in distributed memory.
  *Tool:* language MPI.
  Profit: in time elapsed and peak RAM report.
  *Launching:* standard *via* Astk (mpi_nbcpu/mpi_nbnoeud).
  *Chaining*: useful with 1b or 2c, possible but not very useless with 2a.
  *Office plurality* : useful with 1c or 1d.

- **2c/ Iterative Solvor PETSC** (with possibly MUMPS as preconditionnor cf. PRECOND=' LDLT_SP'); Parallelism in distributed memory.
  *Tool:* language MPI.
  Profit: in time elapsed and peak RAM report.
  *Launching:* standard *via* Astk (mpi_nbcpu/mpi_nbnoeud).
  *Chaining*: useful with 1b or 2b, possible but useless with 2a.
  *Office plurality* : possible but not very useful with 1c, out-perimeter with 1d.

The parallel diagrams 1b+2b/1c or 1b+2b+2c are voted by plebiscite. They support an “industrial” use and “general public”. These parallelisms general practitioners and reliable get notable profits in CPU and peak RAM by heart. Their parameterization is simple, their implementation facilitated *via* Astk (cf. §1).
**For a standard use, the user does not have any more to worry about implementation the fine of parallelism**. By informing the dedicated menus of Astk[4], one fixes the number of necessary hearts (for the MPI and/or OpenMP) as well as the number of nodes on which they are distributed.

## 2.3 Parallelization of the linear systems

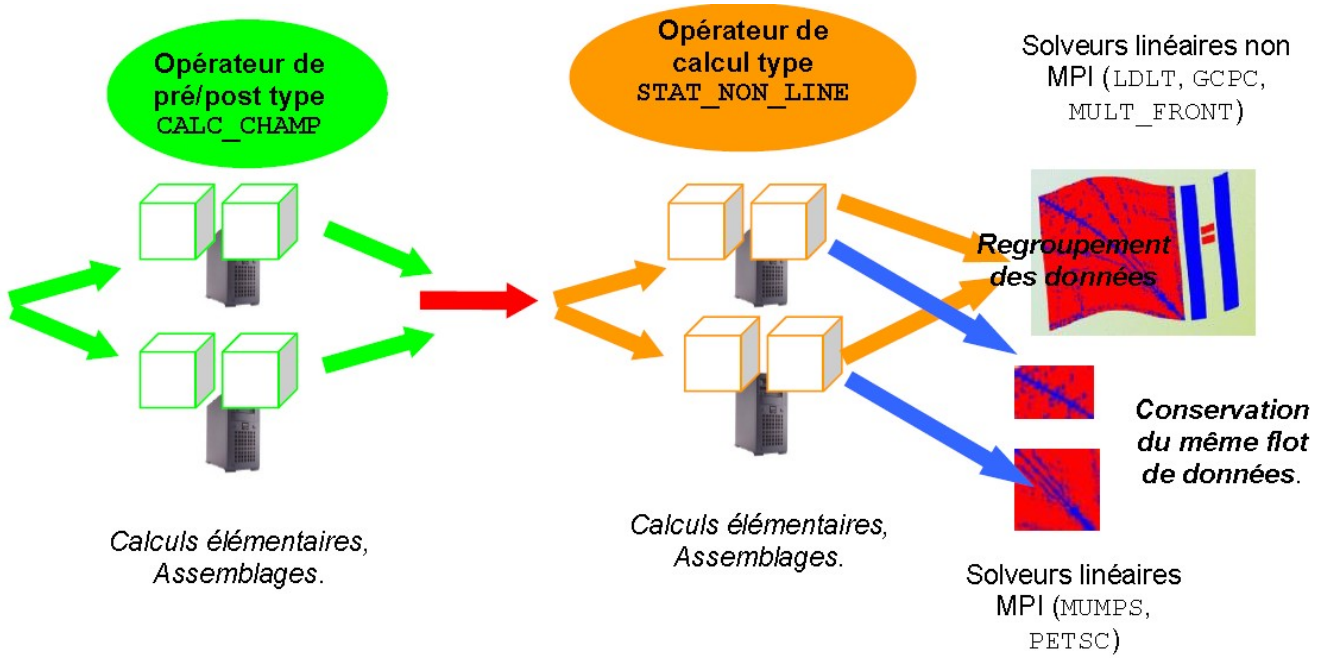


*Figure 2.3.1. _ Organization of parallel diagram MPI of construction and resolution of the linear systems.*

4 Menus Options+ncpus/mpi_nbcpus/mpi_nbnoeud.

In the unfolding of the command file *Code_Aster*, if several processes MPI are activated (parameter mpi_nbcpu), one starts by distributing the frequential meshs and/or sub-bands between the processors (strategies parallel for 1b, 1d, 2b and 2c, cf 2.3.1/2.4.1) figures. This distribution is declined in various manners and it is skeletal in the operators AFFE/MODI_MODELE (to act on the diagram parallels 1b, 2b and 2c), the keyword SOLVEUR (for 2b and 2c) and INFO_MODE/CALC_MODES (for the parallel diagram 1d).

Parallelism more running (strategies 1b+2b, 1b+2c or 1b+2b+2c) is founded on **distribution of the tasks** and of **structures of data** implied in handling of **linear systems** (cf figure 2.3.1) . Because they are these stages of construction and resolution of linear systems which are often more soliciting in computing times and resources memory. They are present in most operators because they are hidden with deepest other algorithms “plus trades”: nonlinear solvor, modal and vibratory calculation, diagram in time…
Once the first stage of distribution of the finite elements of the model carried out between all processes MPI (strategy 1b), each one any more will not manage that the treatments and the data associated with the elements of which it has the load. The construction of the linear systems in *Code_Aster* (elementary calculations, assemblies) is some then accelerated. One often speaks about “ **parallelism in space** ”. It is a parallel diagram rather of a “data-processing” nature.

Once these built portions of linear system (parallel diagram 1b), two cases arise:

- that is to say it **following treatment is naturally sequential** and thus all processes MPI must have access to total information. With this intention these ends of linear systems are gathered and thus the following stage neither will be accelerated, nor will not see lowering its consumption memory. It is generally of an end of operator, a postprocessing or a linear solvor not paralleled in MPI (strategy 1b+2a).
- that is to say it **following treatment accepts parallelism MPI**, it acts then mainly linear solveurs HPC MUMPS (1b+2b), PETSC (1b+2c) or both at the same time (1b+2b+2c). The parallel flood of data builds upstream is then transmitted to them (after some adaptations). These packages of linear algebra reorganize then, in-house, their own parallel diagrams (with a algebraic vision). One speaks then about parallel diagram of a “digital” nature rather. This combination “data-processing parallelism”, on the level of the assembly of the linear system, and, “digital parallelism”, on the level from its resolution, the 2 *via* MPI, is the most current combination.

**Note:**

- *Let us note that at the conclusion of the cycle “construction of system linear – resolution of this one”, some is the scenario implemented (sequential linear solvor or parallel MPI), the vector solution is then transmitted, in entirety, with all processes MPI. The cycle can thus continue some is the following configuration.*

Moreover, one can **to superimpose or substitute for this parallelism MPI** (which functions on all the platforms), another level of parallelism managed this time by **OpenMP language**. This one is however limited to the fractions of machine sharing the same memory physically (PC multi-hearts or nodes of waiter of calculation).
It does not make it possible to lower consumption memory but on the other hand it accelerates certain types of calculation and this, with a granularity lower than that of the MPI: it gets a better acceleration even if the flood of data/treatments is not very important. It is a parallel diagram of a “data-processing” nature which intervenes mainly in the basic operations of algorithms of linear algebra (*via* for example bookstore BLAS).
This parallelism can be:

- that is to say complementary to parallelism MPI by accelerating calculations within each process MPI (in the resolution part of system linear with MUMPS, strategy known as “2b/1c”).
- that is to say to replace parallelism MPI by accelerating the resolution of system linear with MULT_FRONT (strategy 2a).

## 2.4 Distribution of modal calculations

When simulation cannot break up into calculations *Aster* independent, but that it remains **dominated** nevertheless **by generalized modal calculations** (operators INFO_MODE and CALC_MODES), one can organize a specific parallel diagram.

It is founded on **distribution of modal calculations** independent: each one being in charge of a sub-band frequential . This parallel diagram of a “data-processing” nature purely gets only savings of time.

It can however cohabit with the preceding parallel diagrams:

- **chaining** with parallelism MPI of construction of the linear systems (in for example CALC_MATR_ELEM, strategy 1b+1d ),
- **office plurality** with parallelism MPI (even OpenMP) in the resolutions of systems linear (if use of MUMPS, strategy on 2 levels of parallelism, 1d/2b even three, 1d/2b/1c ).

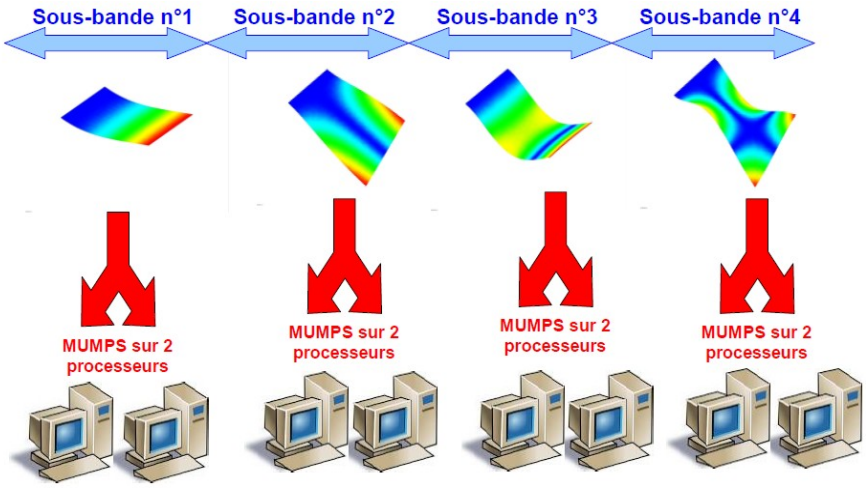


*Figure 2.4.1. _Organization of parallel diagram MPI of distribution of modal calculations and resolutions of the associated linear systems.*

In any event, **any parallel calculation *Aster*** (except of course the scenario 1of a independent calculations) **must respect the paradigms according to** : at the end of the operator of calculation[5], the total bases of each processor are identical[6] and communicator MPI running is the standard communicator (MPI_COMM_WORLD). All the other possible under-transfer MPI must be destroyed.

Because it is not known if the operator who follows in the command file envisaged an incomplete flood of data. It is thus necessary to organize the suitable communications to supplement the possibly incomplete fields.

**Note:**

*Between each order, the user can even change the distribution of the meshs according to the processors. It is enough for him to use the order MODI_MODELE . This mechanism remains licit when one connects various calculations (mode CONTINUATION ). The rule being of course that this new distribution continues, for the model concerned, until the possible next one MODI_MODELE and that this distribution must remain compatible with the parallel parameter setting of calculation (many nodes/processors…).*

5 It is not the case of operators of impression (for example, IMPR_RESU) or of end of calculation (FIN/POURSUITE).

6 And very close to the base generated in sequential mode.

# 3 Some preliminary advices

One formulates here **some advices to help the user to benefit from the strategies of parallel calculation of the code**. But it is necessary well to be conscious, that for any thing, it is initially necessary to optimize and validate its sequential calculation by taking account of advices which swarm in documentations with the orders. With this intention, one uses, if possible, a coarser grid and/or one activates only some steps of time.

**The parameter setting by default and postings/alarms of the code propose a balanced and instrumented operation**. One lists below and, in a nonexhaustive way, several questions which it is interesting to be posed when one seeks to parallel his calculation. Of course, certain questions (and answers) are cumulative and can thus apply simultaneously.

## 3.1 Preamble

It is interesting of **to validate**, as a preliminary, **its parallel calculation** by comparing some iterations in sequential mode and parallel mode. This approach also allows **to gauge the profits atteignables maxima** (speed-up theoretical) and thus to avoid "wasting processors too much". Thus, if one notes $f$ the parallel portion of the code (given for example *via* a run sequential precondition), then theoretical speed-up $S_p$ maximum accessible on $p$ processors is calculated according to the formula of Amdhal (cf [R6.01.03] §2.4) :

$$S_p = \frac{1}{1-f+\frac{f}{p}}$$

For example, if parallelism is used MUMPS distributed by default (scenario 1b+2b) and that the stages of construction/resolution of system linear account for 90% of sequential time ($F$=0.90), the theoretical speed-up is limited to the value $S_\infty = \frac{1}{1-0.9+0.9/\infty} = 10$ ! And this, some is the number of allocated processes MPI.

It is interesting **to evaluate the principal stations of consumption (temps/RAM):** in quasi-static mechanics, they are generally the stages of **elementary calculations/assemblies**, of **resolution of system linear** and algorithms of **contact-friction**. But their proportions depend much on the case (characteristic of the contact, cuts problem, complexity of the laws materials…). If elementary calculations are important, they should be paralleled *via* the scenario 1b (scenario by default). If, on the contrary, the linear systems focus the main part of the costs, the scenarios 2b or 2c can be enough. On the other hand, if it is the contact-friction which dimensions calculation, it is necessary to seek to optimize its parameter setting and/or to parallel its linear solvor interns (cf method GCP+MUMPS).
In dynamics, if one carries out a calculation by projection on modal basis, that can be this last stage which proves to be most expensive. To save time, one can then use the operator CALC_MODES with the option 'BAND' cut out in several sub-bands, sequential and, especially, parallel (scenario 1d). This operator displays a distribution of quasi-independent tasks which leads to goods speedups.

To optimize its parallel calculation, the possible ones should be supervised **imbalances of load** flood of data (CPU and memory) and to limit the overcosts due **with report unloadings** (JEVEUX and MUMPS OOC) and with **filings of fields**. On the subject, one will be able to consult the documentation [U1.03.03] "Indicating of performance of a calculation (time/memory)". It indicates the procedure to follow to establish the good diagnoses and it suggests solutions.

For CALC_MODES with the option 'BAND' cut out in several sub-bands, **the respect of the good load balance is crucial for the effectiveness of parallel calculation** : all the sub-bands must comprise a similar number of modes. One thus advises to proceed in three stages:

- Preliminary calibration of the spectral zone by calls to INFO_MODE(if possible in parallel),
- Examination of the results,

- Launching in mode CONTINUATION calculation CALC_MODES, with the option 'BAND' cut out in several paralleled sub-bands.

For more details one will be able to consult the user's documentation of the operator [U4.52.02].

## 3.2 Some empirical figures

One advises to allocate at least 30 to $50.10^3$ ddls by process MPI (scenarios 1b, 2b or 2c). This granularity can go down to a few thousands from ddls by threads OpenMP while remaining effective (scenarios 1c or 2a).
A standard thermomechanical calculation generally profits, on 32 processors, of a profit about ten in time elapsed and of a factor 4 in peak RAM report (by heart).

For CALC_MODES, one advises to break up his calculation into sub-bands of a few tens of modes (for example 20) and, then, to envisage 2.4 even 8 processes MUMPS by sub-band. It is of course necessary to compose with the number of processors available and the peak report required by the problem[7].
One can obtain profits of a factor 30, in time elapsed, on a hundred processors. The profits in memory are more modest (a few tens of for hundred).
The stage of modal calibration *via* INFO_MODE cost it practically nothing in parallel: a few minutes, at most, for problems about the million unknown, paralleled on a hundred processors. The savings of time are of a X70 factor on a hundred processors and until x2 in peak RAM report.

## 3.3 Independent calculations

When simulation that one wishes to carry out breaks up naturally (parametric study, calculation of sensitivity…) or, artificially (particular thermomechanical chaining…), in similar but independent calculations, one can gain much in time calculation thanks to digital parallelism 1a.

## 3.4 Profit in RAM memory

When the factor dimensioning report concerns resolution of linear systems (what is often the case), office plurality of data-processing parallelisms 1b (elementary calculations/assemblies) and digital 2b (distributed linear solvor MUMPS) is very indicated[8] (even 1b+2b/1c).

Once one distributed his calculation *Code_Aster*+MUMPS/PETSC on sufficient processors, RAM consumption of JEVEUX can become prevalent (compared to those of MUMPS that one spread out over the processors). To restore the good hierarchy (the external solvor must support the peak of RAM consumption!) it is necessary to activate, moreover, the option SOLVEUR/MATR_DISTRIBUEE [U4.50.01].

To solve problems borders of very big sizes (> 5M ddls), one can also try the iterative solveurs of PETSC (parallel strategy 2c or 2b+2c).

## 3.5 Saving of time

If the main part of the costs relates to only the resolutions of systems linear of small (N<0.5M ddls) one can be satisfied to use them linear solveurs MUMPS in centralized mode (strategy 2b) or MULT_FRONT (2a). As soon as the construction of the systems becomes considerable (>5%) or that those prove

7 To distribute on more process, the linear solvor MUMPS allows to reduce the peak report. Other arms of lever: operation into Out-Of-Core and change of renumerotor.

8 To lower consumption memory of MUMPS one can also exploit other parameters: OOC, use as a preconditionnor or relieving of the resolutions [U4.50.01].

rather large, it is paramount to extend the parallel perimeter while activating distribution of elementary calculations/assemblies (1b) and while passing to MUMPS distributed (value by default).

On problems borders of big sizes (N>3M ddls), once good selected digital parameters[9] (preconditionnor, relieving… cf [U4.50.01]), the iterative solveurs parallel (2c) can get savings of very appreciable time compared to the generic direct solveurs (2a/2b). Especially if the resolutions are released[10] because, thereafter, they are corrected by a process including (algorithm of Newton of THER/STAT_NON_LINE…).

For the big modal problems (in the face of problem and/or of many modes), it is of course necessary to think of using CALC_MODES with the option 'BAND' cut out in several sub-bands[11].

9 There is on the other hand no universal rule, all the parameters must be adjusted on a case-by-case basis.
10 I.e. one will be less demanding as for the quality of the solution. That can pass by a poor criterion of stop, the calculation of a preconditionnor frustrates and his mutualisation during several resolutions of linear system… Features of the nonlinear and linear solveurs of *Code_Aster* allow to implement this kind of scenarios easily.
11 Balance *via* modal pre-calibrations carried out with INFO_MODE. If possible in parallel.

# 4 Data-processing parallelisms

## 4.1 Slopes of independent calculations

### 4.1.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** independent calculations (parametric, study of sensitivity…).
**Many advised hearts:** limit of the machine/administrative *batch*.
**Profit:** in time CPU.
**Speedup:** proportional to the number of independent cases.
**Type of parallelism:** data processing *via* Shell scripts.
**Scenario:** 1of a §3. Office plurality with all the other strategies of parallelism sells by auction but addressing itself to advanced users (except perimeter of*Astk*).

### 4.1.2 Implementation

The tool ***Astk* allows to carry out a whole series of similar but independent calculations** (into sequential and especially in parallel MPI). One can use an official version of the code or a beforehand built deprived overload. Command files clarifying **calculations are built dynamically starting from a "model" command file and of a mechanism of type "dictionary":** sets of different parameters for each study (keyword `VALE` file.`distr`), valve blocks *Aster*/Python variables (`PRE/POST_CALCUL`)…
The launching of these parallel slopes is carried out with the usual parameters of tender of*Astk*. One can even Re-parameterize the hardware configuration of calculation (list of the nodes, many hearts, total RAM memory by node…) *via* a classical file `.hostfile`.

For more information on the implementation of this parallelism, one will be able to consult documentations [U1.04.00]/[U2.08.07].

**Note:**

- *Before launching such a slope of calculations, it is preferable to optimize as a preliminary on a standard study, the various dimensioning parameters: management memory `JEVEUX`, aspects modal solveurs nonlinear//linear, keyword `FILING`, algorithm of contact-friction… (cf documentations [U1.03.03], [U4.50.01]…).*
- *One can easily collapse a machine while launching too many calculations with respect to the available resources. It is advised to proceed by stages and to get information as for the use potential of means of shared calculations (class batch, large priority jobs…).*

## 4.2 Elementary calculations and assemblies

### 4.2.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** calculations comprising of elementary calculations/expensive assemblies (nonlinear mechanics). Activated by default as soon as the number of processes MPI>1.
**Many advised hearts:** only, between 4 and 8. Chained with the parallelism distributed of MUMPS or of PETSC (value by default), typically 16.32 even 64.
**Profit:** in time even in memory with linear solvor MUMPS/PETSC (if MATR_DISTRIBUEE).
**Speedup:** Variable profits according to the cases (effectiveness parallel[12]>50%). It is necessary a rather large granularity so that this parallelism remains effective: 30 or $50.10^3$ ddls by process MPI.
**Type of parallelism:** data processing *via* language MPI (mpi_nbcpu/mpi_nbnoeud).
**Scenario:** 1b of §2. Natively conceived to chain with digital parallelisms 2b or 2c. Possible with 1c, possible but not very useful chaining with 1d or 2a. No possible office plurality.

### 4.2.2 Implementation

The implementation of this parallel diagram is carried out in a transparent way for the user. *Via* Astk, it initializes by default as soon as one selected a parallel version of *Code_Aster* (noted *** _mpi) as well as a number of process MPI at least equal to 2.
Thus on the centralized waiter *Aster*, it is necessary to parameterize the following fields in the menu Options:

- **mpi_nbcpu=m,** many allocated processes MPI.
- **mpi_nbnoeud=p,** many nodes on which will be distributed these processes MPI.

For example, on the machine centralized Aster5, the nodes are composed of 24 hearts. To allocate 32 processes MPI at a rate of 8 processes per node, it is thus necessary to position mpi_nbcpu with 32 and mpi_nbnoeud to 4.
One advises, in general, of **not to allocate all the hearts of a node in MPI alone** . That can cause of **to slow down simulation** because, even if part of calculations is some accelerated because of its distribution on more hearts, as those divide certain resources memory, L be access to the data , they, are slowed down.
To use more effectively and to 100 % all the allocated resources one advises rather **to blend parallelism MPI and OpenMP** (cf scenarios 1b+2b/1c or 1b+2b/1c+2c).

Once this number of fixed process MPI, one can launch his calculation (in batch on the machine centralized) with the same parameter setting as into sequential. If this parallel diagram is chained with the digital parallelism of MUMPS or that of PETSC (or 2, cf scenarios 2b and 2c), ON can reduce its peak RAM report by activating the option MATR_DISTRIBUEE.

**As soon as several processes MPI are activated**, the assignment of the model in the command file *Aster* (operator AFFE_MODELE) **distribute its meshs between the processors**. *Code_Aster* being a code finite elements, it is a natural distribution of the data (and associated tasks). Thereafter, stages *Aster* elementary calculations and assemblies (matric and vectorial) will be based on this distribution "to dry up" the floods of data/local treatments each processor. Each processor will carry out only calculations associated with the group with mesh of which it has the load.

This **distribution nets/processor** declines itself in various manners and it is skeletal in the operators AFFE_MODELE[U4.41.01]/MODI_MODELE[U4.41.02] *via* values of the keyword DISTRIBUTION/METHODE=:

12 One gains at least a factor 2 (over the times spent by the paralleled stages) by quadrupling the number of processors.

- 'CENTRALIZES': **The meshs are not distributed** (as into sequential). Each processor knows the entirety of the meshs of the model. Parallelism 1b is thus not implemented. This mode of use is useful for the tests of not-regression and some studies where parallelism 1b little even brings back is against-productive (e.g. if one must gather the data items to nourish a linear system not distributed and that necessary communications MPI are too slow). In all the cases where elementary calculations represent a weak share of total time (e.g. in linear elasticity), this option can be sufficient.
- 'GROUP_ELEM' /'MAIL_DISPERSE'/'MAIL_CONTIGU'/'SOUS_DOMAINE' **(defect)**/'SOUS_DOM.OLD': the meshs are distributed while being based on various criteria: by type, by cyclic distribution, of the same packages cuts or following strategies under-fields.
  In the last two scenarios, the distribution is carried out *via* partitionneurs MONGREL (defect) or SCOTCH TAPE (cf keyword PARTITIONNEUR). Those must thus be installed and linkés with the versions *Code_Aster* used. It is obviously done by default on the centralized machine.

**Note:**

- *Between each order, the user can even change the distribution of the meshs according to the processors. It is enough for him to use the order MODI_MODELE . This mechanism remains licit when one connects various calculations (mode CONTINUATION ). The rule being of course that this new distribution must remain compatible with the parallel parameter setting of calculation (many nodes/processors…).*

### 4.2.3 Structures of distributed data

**The distribution of the data which this kind of digital parallelism implies does not decrease consumption memory inevitably JEVEUX**. By preoccupations with legibility/a maintainability, objects *Code_Aster* usual are initialized with the same size as into sequential. Each process MPI "is just satisfied" to partially fill them with the data produced by the meshs whose the load the processor has. With load for the parallel linear solvor used in the continuation of the operator (MUMPS, PETSC or them 2) to assemble these incomplete and distributed data. One thus does not recut generally these structures of data, they comprise many zero values.
This strategy is bearable only as long as the objects JEVEUX principal implied in elementary calculations/assemblies (CHAM_ELEM, RESU_ELEM, MATR_ASSE and CHAM_NO) do not dimension the constraints memory of calculation (cf. §5 of [U1.03.03]).
Normally their occupation memory negligible is compared with that of the linear solvor. But when this last (e.g. MUMPS ) is also Out-Of-Core [13] and paralleled in MPI (with the distribution of the data between processors that implies), this hierarchy is not inevitably any more respected. From where the introduction of **an option** (keyword MATR_DISTRIBUEE cf [U4.50.01]) **allowing truly to recut, with just, the block of matrix *Aster* clean with each process MPI.**

**Note:**

- *In distributed mode, each process MPI handles only incomplete matrices (recut or not). On the other hand, in order to avoid many communications MPI (during the evaluation of the criteria of stop, calculations of residues…), this scenario was not retained for the vectors second members. Their constructions are well paralleled, but, at the conclusion of the assembly, the contributions partial of each process are gathered. Thus, any process MPI entirely knows the vectors (CHAM_NO) implied in calculation.*

13 The Outone (OOC) is a way of managing of the memory which consists in discharging on disc certain objects allocated by the code to release from RAM. Strategy OOC makes it possible to deal with larger problems but these accesses disc slow down calculation. *A contrario*, the mode In-Core (IC) consists in keeping the objects in RAM. That limits the size of the accessible problems but privileges speed.

## 4.3 Distribution of calculations of linear algebra basic

### 4.3.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** calculations comprising of the resolutions of systems linear *via* MUMPS (use direct solvor or preconditionnor of PETSC/GCPC).
**Many advised hearts:** on Aster5, between 2 and 12. Not more than of physical hearts sharing the same physical memory.
**Profit:** in time elapsed (but increase in time CPU).
**Speedup:** Variable profits according to the cases (effectiveness parallel[14]>50%). A low granularity is enough so that this parallelism remains effective: $10.10^3$ ddls by threads OpenMP.
**Type of parallelism:** data processing *via* the OpenMP language (ncpus).
**Scenario:** 1c of §2. A classical use consists in benefitting from a hybrid parallelism MPI+OpenMP to accentuate the performances of MUMPS and to draw part with 100% from the resources machine (2b/1c or (2b/1c) +2c).
Against-productive with 2a, useful with 2b, possible but not very useful office plurality with 2c (only) or 1d.

### 4.3.2 Implementation

The implementation of this parallel diagram is carried out in a transparent way for the user. *Via* Astk, it initializes by default as soon as one selected a parallel version of *Code_Aster* (noted *** _mpi) as well as a number of threads OpenMP at least equal to 2.
Thus on the centralized waiter *Aster*, it is necessary to parameterize the following fields in the menu Options:

- **ncpus=k,** number of threads OpenMP allocated (by process MPI if mpi_nbcpu>1).

This parallel diagram is generally used in conjunction of parallelism MPI of MUMPS. Because one advise, in general, of **not to allocate all the hearts of a node in MPI alone**. That can cause of **to slow down simulation** because, even if part of calculations is some accelerated because of its distribution on more hearts, as those certain resources memory divide, accesses to the data , they, are slowed down. To use more effectively and to 100% all the allocated resources one advises rather **to blend parallelism MPI and OpenMP** (cf scenarios 2b/1c or (2b/1c) +2c).
In this kind of hybrid parallelism (MPI/OpenMP), the tool Astk [U1.04.00] automatically supplements the number of threads according to the resources machines, as soon as the field ncpus is left empty.
For example, on the machine centralized *Aster5*, the nodes are composed of 24 hearts. If one wishes to organize a hybrid parallelism 12MPIx4OpenMP, it is enough to position mpi_nbcpu to 12, mpi_nbnoeud to 2 and ncpus=<vide> (or 4 explicitly).

**Note:**

- *The implementation of this parallelism depends on the data-processing context (material, software) and on the bookstores of linear algebra used. On the centralized machine Aster, one uses threadées BLAS MKL.*

14 One gains at least a factor 2 (over the times spent by the paralleled stages) by quadrupling the number of processors.

## 4.4 Modal calculations of INFO_MODE/CALC_MODES

### 4.4.1 Description

**Use:** general public *via Astk.*
**Perimeter of use:** calculations comprising of expensive searchs for clean modes.
**Many advised hearts:** several tens (for example, many frequential sub-bands X 2.4 or 8).
**Profit:** in time *elapsed v*oire in RAM memory (thanks to the second level of parallelism).
**Speedup:** Variable profits according to the cases: effectiveness of about 70% on the first level of parallelism (on the frequential sub-bands) supplemented by the possible parallelism of the second level (if SOLVEUR=MUMPS, complementary effectiveness about 20%).
**Type of parallelism:** data processing *via* language MPI (mpi_nbcpu/mpi_nbnoeud).
**Scenario:** 1d of §2. Natively conceived to couple itself with parallelism 2b (even 2b/1c).
Possible but not very useful chaining with 1b. Possible but not very useful with 2a and impossible office plurality with 2c (except perimeter).

### 4.4.2 Implementation

The use of CALC_MODES with the option 'BAND' cut out in several sub-bands is to be privileged when with modal problems are dealt **mean sizes or large** (>0.5M ddls) and/or that one seeks one **good part of their spectra** (> 50 modes).
One then cuts out calculation in several frequential sub-bands. On each one of these sub-bands, a modal solvor carries out the associated search for modes. With this intention, this modal solvor uses a linear solvor intensively.
These two bricks of calculation (modal solvor and linear solvor) are them **dimensioning stages** calculation in term of consumption memory and time. It is on them which it should be put the accent if one wants significantly to reduce the costs calculation of this operator.

However, the organization of modal calculation on distinct sub-bands offers here an ideal framework of parallelism: **distribution of large almost independent calculations**. Its parallelism makes it possible to gain much in time but at the cost of a overcost in memory[15].
If one has a sufficient number of processors (> with the number of nonempty sub-bands), one can then engage one **second level of parallelism *via* the linear solvor** (if one has chooses METHODE=' MUMPS'). This one will make it possible to continue to gain in time but especially, it will make it possible to compensate for the overcost report of the first level to even decrease the peak sequential report notably.

For one **optimal use** of CALC_MODES with the option 'BAND' cut out in several paralleled sub-bands, he is thus advised of:

- **To build relatively balanced sub-bands of calculation**. With this intention, one can thus, as a preliminary, gauge the studied spectrum *via* a call to INFO_MODE [U4.52.01] (if possible in parallel). Then to launch calculation CALC_MODES with the option 'BAND' cut out in several sub-bands paralleled according to the number of selected sub-bands and amongst processors available.

- Of **to take sub-bands** finer than into sequential, **between 10 and 20 modes** instead of 40 to 80 modes into sequential. The quality of the modes and the robustness of calculation will be some increased. The peak report will be decreased by it. It however remains to have sufficient processors available (and with enough memory).

- **To select a number of processors** who is a multiple amongst sub-bands (not vacuums). Thus, one reduces the déséquilibrages of loads which harm the performances.

15 Because of buffers MPI required by the clean communications of vectors in the end of MODE_ITER_SIMULT.

For more details one will be able to consult the user's documentation of the operator [U4.52.02].

# 5 Digital parallelisms

## 5.1 Direct Solvor MULT_FRONT

### 5.1.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** calculations comprising of the resolutions of linear systems expensive (in general STAT/DYNA_NON_LINE, MECA_STATIQUE...).
**Many advised hearts:** 2 or 4.
**Profit:** in time elapsed.
**Speedup:** Variable profits according to the cases (effectiveness parallel≈50%). It is necessary a good granularity so that this parallelism remains effective: at least $50.10^3$ ddls by heart.
**Type of parallelism:** digital *via* the OpenMP language (ncpus).
**Scenario:** 2of a §2. Possible but not very useful chaining with 1b, 2b or 2c. Against-productive with 1c, possible but not very useful office plurality with 1d.

### 5.1.2 Implementation

This method in-house multifrontale developed (cf [R6.02.02] or [U4.50.01] §3.5) is used *via* the keyword SOLVEUR/METHODE=' MULT_FRONT'. It is the linear solvor (historical and self-supporting) recommended by default into sequential on the models of small or average size (<0.5M ddls).
The implementation of this parallelism is carried out in a transparent way for the user. It initializes by default as soon as a calculation is launched *via Astk* (finely Options) using several threads OpenMP.
Thus on the centralized waiter *Aster*, it is necessary to parameterize the following fields :

- ncpus=n, number of threads OpenMP allocated.

Once this number of threads fixed one can launch his calculation (in batch on the machine centralized) with the same parameter setting as into sequential. One can of course lower the specifications in time of calculation.

## 5.2 Package MUMPS

### 5.2.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** calculations comprising of the resolutions of linear systems expensive (in general STAT/DYNA_NON_LINE, MECA_STATIQUE...).
**Many advised hearts:** chained with distributed parallelism calculations elementary/assemblies typically 16.32 even 64.
**Profit:** in time CPU and RAM memory.
**Speedup:** Variable profits according to the cases (effectiveness parallel≈30%). It is necessary an average granularity so that this parallelism remains effective: between 30 and $50.10^3$ ddls by process MPI.
**Type of parallelism:** digital *via* language MPI.
**Scenario:** 2b of §3. Natively conceived to chain itself with parallelisms 1b or 2c. Possible but not very useful chaining with 2a. Very useful coupling with 1c or 1d (even 2).

### 5.2.2 Implementation

**This method multifrontale is pressed on external product MUMPS** (cf [R6.02.03] or [U4.50.01] §3.7) is used either as a direct solvor (keyword SOLVEUR/METHODE=' MUMPS'), that is to say as a preconditionnor of the iterative solveurs PETSC or GCPC (keyword SOLVEUR/PRE_COND=' LDLT_SP').
**It is package HPC advised to fully exploit the profits CPU/RAM which parallelism can get**. This kind of parallelism is performing (especially when it is chained with 1b and is coupled with 1c) while remaining generic, robust and general public.

The implementation of this parallel diagram is carried out in a transparent way for the user. *Via* Astk, it initializes by default as soon as one selected a parallel version of *Code_Aster* (noted *** _mpi) as well as a number of process MPI at least equal to 2.
Thus on the centralized waiter *Aster*, it is necessary to parameterize the following fields in the menu Options:

- **mpi_nbcpu=m,** many allocated processes MPI.
- **mpi_nbnoeud=p,** many nodes on which will be distributed these processes MPI.

For example, on the machine centralized Aster5, the nodes are composed of 24 hearts. To allocate 32 processes MPI at a rate of 8 processes per node, it is thus necessary to position mpi_nbcpu with 32 and mpi_nbnoeud to 4.
One advises, in general, of **not to allocate all the hearts of a node in MPI alone** . That can cause of **to slow down simulation** because, even if part of calculations is some accelerated because of its distribution on more hearts, as those divide certain resources memory, L be access to the data , they, are slowed down.
To use more effectively and to 100 % all the allocated resources one advises rather **to blend parallelism MPI and OpenMP** (cf scenarios 1b+2b/1c or 1b+2b/1c+2c).

Ideally, this linear solvor HPC must be used in distributed parallel mode (DISTRIBUTION/METHODE=' GROUP_ELEM'/'MAIL_DISPERSE'/'MAIL_CONTIGU'/'SOUS_DOMAINE'/'SOUS_DOM.OLD'). I.e. it is necessary to have initiated this linear solvor upstream, within the procedures of calculations elementary/assemblies, floods of data/treatments distributed (parallel scenario 1b). MUMPS accept as starter these incomplete data and it gathers them in-house. One thus does not waste time (as it is the case for the other linear solveurs) to supplement the data resulting from each processor. This operating process is activated by default in the orders AFFE/MODI_MODELE (cf. §4.2).
In centralized mode (CENTRALIZE), the phase upstream of construction of the linear systems is not paralleled (each processor proceeds as into sequential). MUMPS account holds then only data from the main processor.

In the first case, the code is parallel construction of the linear system until its resolution (chaining of parallelisms 1b+2b), in the second case, one exploits parallelism MPI only on the part resolution (parallelism 2b).

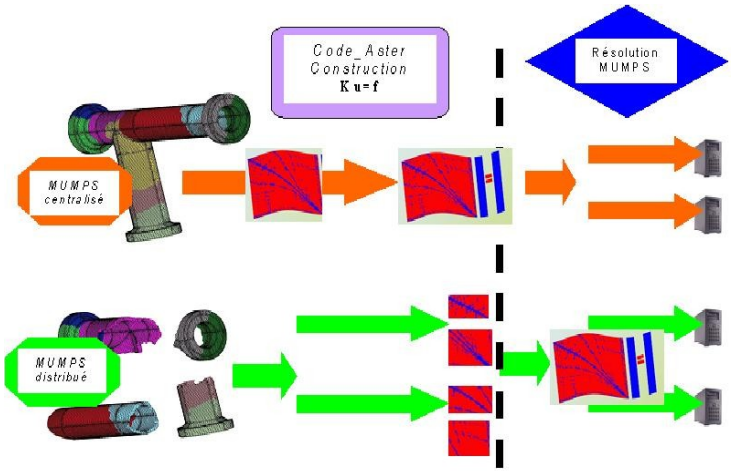


*Figure 5.2.1. _Floods of parallel data/treatments of the Code_Aster+ coupling MUMPS according to the mode of use: centralized or distributed.*

**Note:**

*When the share of calculation devoted to the construction of the linear system is weak (<5%), the two modes of use (centralized or distributed) display savings of similar time. On the other hand, only the distributed approach gets, moreover, of the profits on RAM consumption.*

## 5.3 Iterative Solvor PETSC

### 5.3.1 Description

**Use:** general public *via Astk*.
**Perimeter of use:** calculations comprising of the resolutions of linear systems expensive (in general STAT/DYNA_NON_LINE, MECA_STATIQUE…). Rather nonlinear problems of big sizes.
**Many advised hearts:** chained with distributed parallelism calculations elementary/assemblies (1b), even that of preconditionnor MUMPS (2b), typically 16.32 even 64.
**Profit:** in time CPU and RAM memory (according to the préconditionneurs).
**Speedup:** variable profits according to the cases (effectiveness parallel50%). It is necessary an average granularity so that this parallelism remains effective: $50.10^3$ ddls by process MPI.
**Type of parallelism:** digital *via* language MPI.
**Scenario:** 2c of §3. Natively conceived to chain itself with parallelisms 1b or 2b; possible but not very useful chaining with 2a. Possible but not very useful office plurality with 1c, out-perimeter with 1d.

### 5.3.2 Implementation

This library of iterative solveurs (cf [R6.01.02] or [U4.50.01] §3.9) is used *via* the keyword SOLVEUR/METHODE=' PETSC'. This kind of linear solvor is advised to treat, either of the problems borders of very big size (>5M ddls), or into nonlinear, to fully draw part of the mutualisation from the preconditionnor between various steps of Newton**.**

The implementation of this parallelism is carried out as for package MUMPS (cf. §5.2).

**Note:**

- *Contrary to the direct parallel solveurs (MUMPS, MULT_FRONT), the iterative ones are not universal (they cannot be to use into modal) and always robust. They can be very competitive (in time and especially in memory), but it is necessary to find the point of operation (algorithm, preconditionnor…) adapted to the problem. However, on this last point, the generalized use (and parameterized by default) of MUMPS single precision as preconditionnor (PRE_COND=' LDLT_SP') improved the things considerably.*