A better diagnostic of the load imbalance in OASIS based coupled systems

E. Maisonnave, L. Coquart, A. Piacentini

TR/CMGC/20/176

Abstract

Coupled computations in separated executables are often performed concurrently. Their respective speeds is a function of their MPI parallelism. Speeds cannot be the same, and computations load balanced, without setting the appropriate MPI domain decomposition, which is unlikely to happen without a comprehensive work. A complete rewriting of the tool for load balancing measurement in OASIS coupled models (`lucia`) becomes necessary, with two main requirements that are to simplify of the delivered information, proposed to non technical staff and to avoid any disk writing at runtime that could perturb the measurement. This implementation proposes an alternate version launched by the OASIS library itself at the end of the simulation. We avoid the production and the post-processing analysis of trace files. We deliver at low cost a larger set of summarised quantities that can help to better understand the origin of any coupling extra cost. Moreover, we provide the timeline, for every process involved in the coupling, of every coupling events, such as coupling field sending or receiving, but also interpolation, file reading or writing, in addition to the coupling initialisation or termination phases. These timelines are available in netCDF format files that can be easily and quickly handled with commonly used visualisation tools. To be able to keep the implementation as simple as possible (~ 700 lines), we limit the analysis to an OASIS component (and not partition) per-basis, that requires additional memory, mainly on each component master process, approximately equal to the size of a single precision array containing the coupling events x number of MPI processes. This means that users will have to restraint the analysis to relatively short simulations or reasonably MPI-parallel models to avoid memory overflow

Table of Contents

1- Rationale

The modularity of the MPI "Multiple Program - Multiple Data" (MPMD) coupled configurations have unfortunately a drawback : it is easy to waste a large part of the computing resources they need. Computations in separated executables are often performed concurrently. Their respective speeds is a function of their MPI parallelism. Speeds cannot be the same, and computations load balanced, without setting respective appropriate MPI domain decompositions, which is unlikely to happen without a comprehensive work. A collection of CMIP5 model simulation analysis [1] shows that up to 62% of the allocated resources can be wasted by this imbalance. Even though such a deep problem has several origins, like the lack of human resources for solving non blocking computing issues, or the popularity of non frugal configurations, e.g. [2], we think that, in the perspective of the reduction of the carbon footprint of our activities, the problem of load imbalance in coupled configuration is well worth some effort of addressing. In addition, to have any chance to actually reduce the waste at a community level, any proposed solution should be simple and portable enough to fit all the community requirements, and this solution must be strongly advertised in the coupler documentation.

A first solution [3] was implemented in 2014 for the OASIS3-MCT based coupling systems. Called "`lucia`", this post processing tool aimed to deliver a simplified evaluation of the load imbalance. It derives from an exploratory tool developed in 2010 for OASIS3 [4]. Since the OASIS3 coupler was a separated executable (OASIS3-MCT is a library included in the model executable), the post-processing tool has to be adapted but some part of the code originally designed for a separated coupler remained inadequate. Included in the official OASIS release, it fitted most of the community requirements during the last decade. But difficulties emerged, that can be categorised following two axes:

- the simulation slow down due to OASIS traces, needed by the post processing tool, and written on disk during the simulation [5] ;
- the irregular duration of model time steps (computation, disk access) which could lead to erroneous interpretation of the `lucia` simplified diagnostic, as observed for the EC-Earth model in [6].

These issues cannot be remedied without a deep modification of our code. This put together with its congenital inadequacy to the lack of central coupler in separate executable, we conclude that a complete rewriting of the load balancing measurement routines is necessary.

This overhaul is constrained by two requirements:

- the simplicity of the delivered information, in direction to a public (all community researchers, technical staff, students) that has few time to spend to the technical setup of a model ;
- no disk writing.

Simplicity is the core advantage of our dedicated tool, in comparison to more comprehensive analysis of the MPI communications of a model, that can be done by a tool like `paraver` [7],

more powerful but less usable (readability/handling of highly parallel configurations, compatibility with MPI MPMD, difficulty to separate OASIS communications from others).

In our previous version of the post processing tool, the in-memory implementation was not considered as mandatory. The opposite choice (disk writing) was useful to avoid a memory congestion, but also to make possible an evaluation of the load imbalance before the end of the simulation. We think it is now possible to rely on the high parallelism of our models to think about a distributed version of the post-processing tool algorithm.

In addition to these two main constraints, one could like to add to the requirements :

- To diagnose the total time of the simulation. Since our measurements exclude the first and last time steps, the duration of the initialisation and termination phases are ignored, though OASIS can be at the origin of slow downs during these phases. A measurement can be done just before the `OASIS_init` and just after the `OASIS_terminate` routines to have a good estimation of this duration ;
- To include the OASIS internal timings ;
- To compute normalised performance parameters like speed in SYPD, cost in CHPSY.

At the opposite, we are strongly unfavourable to try to evaluate the non precisely defined "time of communication", which would also require to precisely identify the communication pattern and would be paid by a complication of our code. Though we do not deny the existence of this coupling extra time, its importance would only be noticed at the limit of the models scalability, a zone where the model internal communications and imbalance costs [8] are probably dominating any coupling extra cost.

2- BSC exploratory work

In the framework of the IS-ENES3 EU project, a team of the Barcelona Supercomputing Centre proposed to enhance the functioning and the quality of the information provided by the load balancing post processing tool `lucia`. To let us testing this solution, they provided a set of OASIS modified routines (`mod_oasis_advance.F90` & `mod_oasis_method.F90`), a new Python tool for post processing (`lucia_lite.py`) and a comprehensive documentation.

OASIS is a community tool, currently in use in dozens of coupled models. For that reason, it is necessary to test any modification of this library (including the load balancing tool) on a large set of representative cases and various supercomputing environments. The final validation would have to be done that way. But for the need of this intermediate evaluation, a single testing with the tutorial[1] example, available in the OASIS release, is done. This test proposes the exchange of 2 fields, during an adjustable number of time steps, between two small FORTRAN programs.

As a first step, a machine has to be selected with the following constraints:

- a Python library, posterior to the 3.6 version, must be installed, together with the Panda 1.0.3 packet ;

1 See examples/tutorial directory of OASIS official sources

- an X server must be accessible to be able to produce the new set of plots which details the load balancing characteristic[2].

This latter constraint forbids the use of our domestic supercomputer, the AMD Rome "belenos" machine, recently installed at Météo-France. The first constraint also removes from the list of compatibility all our CERFACS local machines, the CNRS supercomputers jean-zay and irene, and the NERSC machine fram. The only machine (accessible for this test) that fits the requirement is the DKRZ supercomputer mistral. A small number of resources, available in the context of the OASIS dedicated support, also organised within the IS-ENES3 project, were used to proceed to both coupled simulation (compute nodes) and post-processing (interactive node). This shortening of the compatible machine list is the consequence of the use of recent Python functionalities. Even though this issue is supposed to be solved when new machines, and their up-to-date Python libraries, will replace the current ones, it could be good, for the large compatibility needed by the OASIS community tool, to restrain the use of too recent functionalities.

The post-processing analysis gives the expected results on mistral. However, the setting, in the Python code, of 3 out of the 4 parameters was necessary (bounds of coupling steps to take into account, amount of files to read, number of coupling steps to print). We found that the fourth parameter (components that will not be taken into account for the analysis) would better be calculated, e.g. by evaluating to zero their number of coupling fields.

We can summarise the analysis of the results given by the post processing tool in three points:

- The Python tool produces additional plots and gives more information about the coupling, from which an analysis of model dependences (which model is waiting for which one) and a En/Cn/In[3] decomposition on a coupling time step basis ;
- It provides the original load imbalance graphic for each component (oasis.png). For comparison, and thanks to modifications also provided by BSC, we performed an original lucia analysis on the same set of traces. The Python tool numbers slightly differ from the lucia ones, which can be explained by the functional differences emphasised below ;
- For a simple coupled simulation (30 coupling time steps, 2 computing cores and 2 fields exchanged) the post processing analysis takes quite a long time (11s for 22 time steps and 2 processors). We did not try to have a better idea of the scaling property of the tool. The original lucia tool selects a subset of MPI processes, to better save computing performance, during runtime (disk writing) and post processing (disk writing and memory requirements). The Python tool parametrisation allows to perform an analysis based on the whole set of coupling events, but requires that the whole corresponding information must be produced at runtime.

We also noticed a functional regression compared to the original lucia tool: the exclusion of processes not involved in coupling is not possible, unless the master process is the only one involved.

In order to give an En/Cn/In decomposition per "timestep", one of the most interesting additional information produced, the BSC developers introduced a definition of what a

2 We assume that it would be possible to avoid this by producing only files without X11 windows visualisation

3 En : Waiting time for component n, Cn : computing time, In : OASIS interpolation time

timestep is in the context of a coupled system. For each "timestamp", i.e. the current simulated time in second when an operation is performed, the first coupling operation (sending, receiving or interpolating) is identified. Then, for each component, a timestep is defined as the time interval between two consecutive first coupling operations. We will call this timestep "coupling interval" or CI. We did not practically get the value of a CI when calculated with a component that does not include any event during a time stamp, but we think it is equal to zero, which could be somehow misleading. In addition, the CI duration could be irregular.

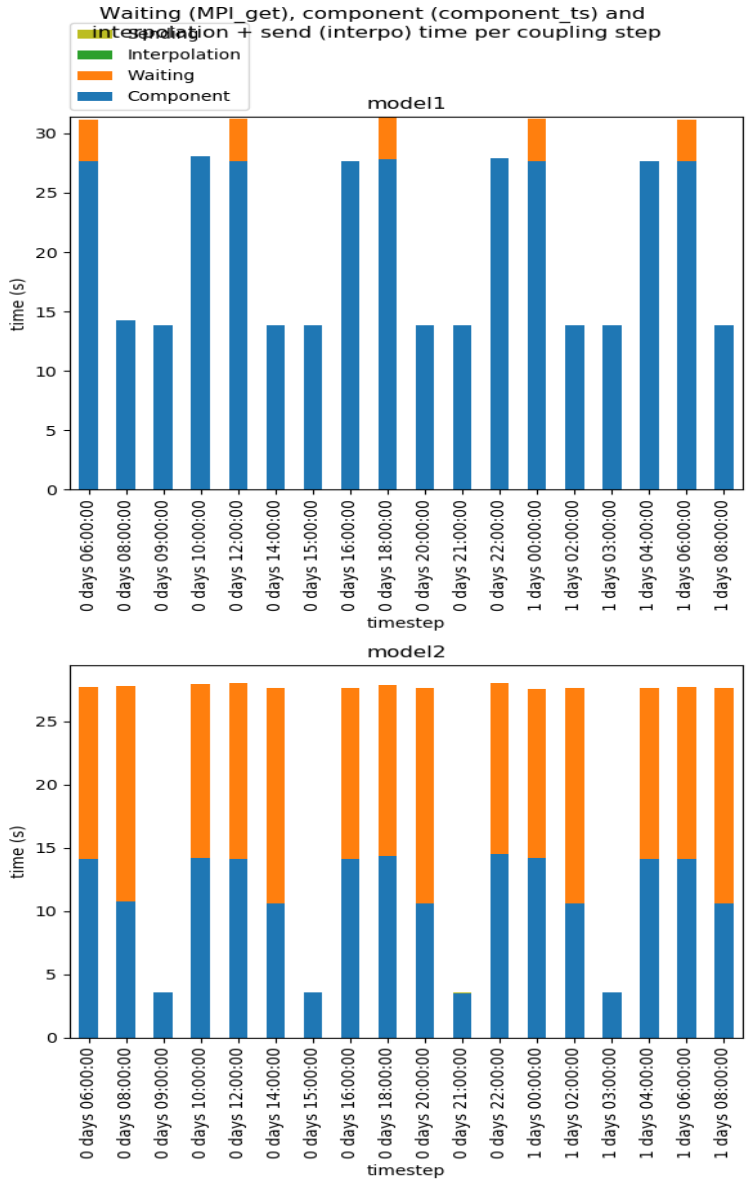

Figure 1: En/Cn/In decomposition per timestep, for selected timesteps of a "tutorial" coupling system run

On figure 1, we show the CI related plot we got with the "tutorial" example. The irregularity of the CI length gives the wrong feeling that the model 1 computation per model timestep is also irregular. A closer look to the plot shows that the computing time is two time smaller when the CI length is also two time smaller. But the same analysis would lead to conclude that model 2 has irregular computing time per model timestep, which is not the case. Actually, this irregularity comes from the definition of CI bounds that could lead to wrongly attribute the computing time of a model timestep to the previous or the next CI. In our Fig 1 example, a close check of the CI content shows that 3 model time steps are included in the CI tagged 8h,

14h, 20h and 2h, but only 1 model time step in the CI tagged 9h, 15h, 21h and 3h. The irregularity is due to the irregular length of a CI, and not to the irregularity of the model time step.

One could also find misleading to see that, for a given time stamp, the sum of coupling and non coupling tasks is not the same for model 1 and model 2, considering that they are the only two components of the system. These confusing details probably shows that the choice of CI as time slice unit, though convenient, may not be the best to reveal the load imbalance at a finer time scale than the whole simulation duration.

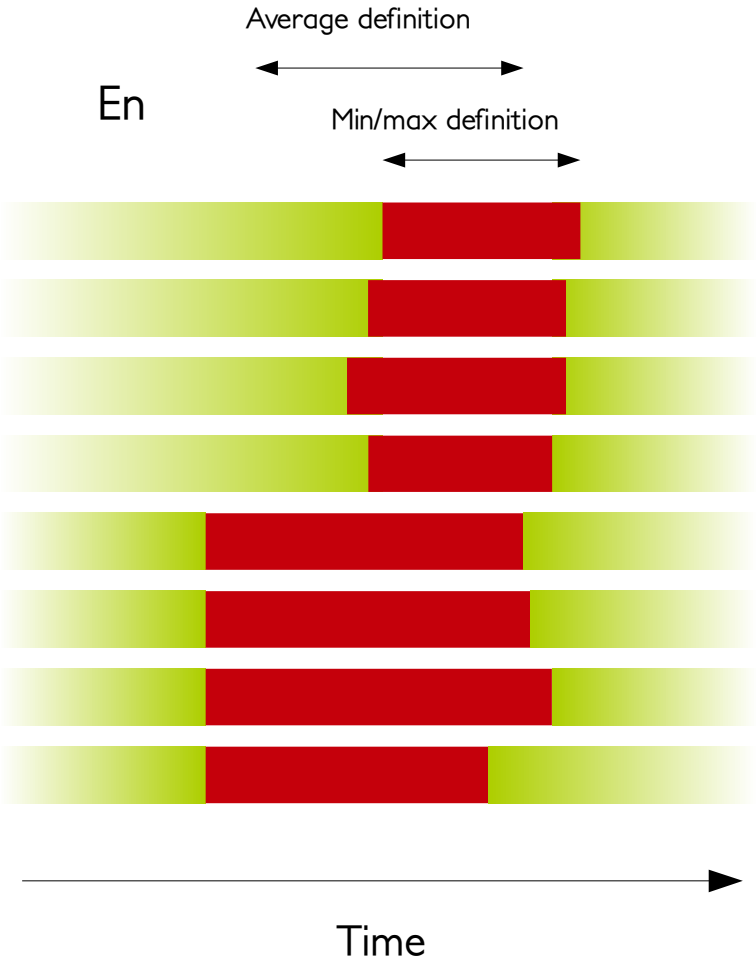

Figure 2: Example of coupling operation, performed on 8 processes, and the corresponding En definitions in original `lucia` (min/max) and new Python tool

The BSC tool innovation which is the less compatible with the needs described in Chapter 1 deals with the definition of En (for each model, time spent waiting for completion of sending and receiving MPI messages) and, consequently, Cn (for each model, the time spent to perform its own calculations and OASIS interpolations). In the previous version of `lucia`, En measures the time spent for each coupling operation, between the latest start of the operation and the latest completion of the operation, among all model MPI processes. These maximum values are preferred to averaged values in order to keep jitter[4] time into calculation times. In the Python

4 Jitter is defined as the time separating the first and the last process when starts a coupling event

tool, average is preferred which means that part of the formerly defined computing processes time can be put back to En. An extreme, but realistic, example shows the difference in Figure 2.

With the BSC definition proposed, a delay in the end of calculation for a set of model MPI processes will lead to include the jitter time, due to the model internal load imbalance, to the waiting time, usually associated to an extra time attributed to the coupling. With the previous min/max definition, En better represents the sum of the model external imbalance (a model is waiting another one), with the coupling field communication time. We keep thinking that this original definition is less confusing.

In addition, the BSC tool removes the jitter evaluation and it is impossible to have an idea of how asynchronous the exchanges are among MPI processes.

Put it all together, the BSC preparatory work proposes to bring new information to the user knowledge. Without jeopardising the simplicity of our tool, we think important to include the model dependency graph and the estimate of load imbalance per coupling time step. As explained in Section 1, we need to produce the diagnostic at the end of the simulation by the coupled model itself, which means that the post processing tool is no longer necessary and has to be rewritten in the OASIS library.

3- Proposed solution

To keep the load balance information simple, starting from a huge amount of raw data (start/end of every coupling operation from every MPI process involved), necessarily implies an intensive processing. The no-disk-access constraint requires to store this huge amount of data and to delay the intensive analysis until the end of the simulation.

A simple evaluation of this data volume can be done that way:

```
(Interpolation_timings + Get_or_Put_timings ) * field_nb *
single_precision_real_size * cpl_time_step_nb * MPI_processes
```

Interpolation or send/receive communication requires 2 float numbers for storage (timing before and after the event). 3D fields are evaluated as a single field. In a distributed memory algorithm, the maximum amount of data to store per node comes from only one model, since it is unlikely to found more than one model per node. In this case, the maximum number of fields exchanged is equal to the number of field sent/received by one model (about 30 in the worst known case). We also evaluate to 1 or less (if OpenMP hybrid parallelism) the number of MPI processes per core (about 200 per node, in the near future compute nodes). For a standard climate simulation, this corresponds to a maximum volume per node of :

```
( 2 + 2 ) * 30 * 4 * 10,000 * 200 ~ 1Gb
```

Given the current memory of our compute nodes (256Gb in the recent AMD Rome based Météo-France supercomputer), it is acceptable but not negligible. It will be mandatory to shorten as much as possible the analysis length to a time slice of the simulation.

In addition, since our algorithm requires minimum/maximum calculations across nodes, the information has to be gathered and communicated to a master process. The cost of such exchange means that, to avoid to perturb the measurement, we need to wait the end of the simulation. One consequence is that only successful simulations could deliver the load imbalance information (functional regression compared to the existing tool).

For gathering, we propose a pure MPI algorithm, and not to ask the users to compile their model with the additional `-qopenmp` option. Considering that the load imbalance analysis is needed during the testing period of a climate model lifetime, complex and fast algorithms are not considered and robustness is preferred, involving global all-to-one communications to one or several active post-processing processes. In the proposed implementation, all MPI processes are sending their data to the component master processes, which build the final synthesis and the corresponding plots. If memory constraints become too tight, we will implement in a second step the analysis splitting on several master processes.

The final information has to be summarised for all components of the coupled system. But the amount of information is already shortened by the first per-component analysis and can be compiled easily on only one process of the coupled system.

To keep the simplicity of the delivered information, we propose to only add to the initial En/Cn/In global information:

- performance parameters (SYPD, CHPSY) ;
- jitter accumulated for all coupling fields ;
- model dependency (En decomposition depending on the waited model).

Finally, the En/Cn/In timing (before/after) for each process of each coupled component is stored in a netCDF file. For the same simplification reasons, this format is preferred to the direct production of a GIF or JPG image because:

- it can be directly produced by the models, since the netCDF library is already used by OASIS and linked with the model executable ;
- the full timeline, including all coupling event (up to 10,000) in all MPI processes (up to 10,000), must be stored in large arrays and visualised by efficient software. Such 2D arrays are already handled in netCDF format ;
- visualisation, zoom in time or space direction and other finer analysis of a netCDF dataset are commonly known by the majority of our community members, and can be performed with a large variety of standard tools like Ferret, ncview, etc.

4- Implementation

The initial implementation starts from a trunk version of OASIS3-MCT v4 (22/07/2020), and a private branch[5] is built to share and possibly correct & enhance this development.

5 `eric_ld`

The load balancing measurement procedure can be basically divided into three parts. In a first step, we declare all the coupling events that will be measured. This phase ends with the allocation of the buffers that will keep in memory the starting/ending timing of each coupling event. Secondly, measurements are done at runtime and saved into these buffers. Thirdly, the information is gathered and processed to produce the diagnostics.

We extend the definition of *coupling event* (CE) to every calling of selected OASIS subroutines that we want to visualise in the timeline. The different kind of CEs are listed in Table 1. For any *coupled* component, at least two CEs are measured: the coupling setup phase (`OASIS_enddef`) during initialisation and the coupling termination. Notice that this termination phase measurement only takes into account the beginning of this phase (`MPI_barrier`) since the main operation performed in this routine is the `MPI_finalize` operation, that forbids further gathering/processing task needed by this tool. A third operation can also be instrumented if applicable: the MPI partitioning declaration.

Event	Send	Receive	Map/interpolate	Output	Read
Kind	`LB_PUT`	`LB_GET`	`LB_MAP`	`LB_OUT`	`LB_READ`
Object	Coupling field	Coupling field	Coupling field	Coupling field	Coupling field

Event	Write restart	Write intermediate restart	Describe MPI partitioning	End coupling definition phase	End coupled simulation
Kind	`LB_RST`	`LB_TRN`	`LB_PART`	`LB_ENDF`	`LB_TERM`
Object	Coupling field	Coupling field	Component	Component	Component

Table 1: List of coupling events (CE), by kind (name of the corresponding parameter used in the FORTRAN code). A CE can be related to a coupling field or to a component

If a component exchanges coupling fields with the coupled system, we measure the time spent during the CEs related to these exchanges. For each coupling field, the sending or receiving can be associated with interpolation (and/or remapping), with the writing of the coupled quantity in a netCDF file (EXPOUT option) and with the writing of this same quantity in one or several restart files, depending on the LAG and time transformation options. The INPUT and OUTPUT operations (array read or write), although not producing any exchange between components, are also instrumented and their timing can also be visualised in the timeline.

If a component is not coupled at all (parameter `coupled` of the `oasis_init` routine set to false), our load balancing measurement procedure ignores it, since it does not call the `oasis_terminate` routine which includes the final analysis procedure. For this reason, the cost measurement of some coupled component such as IO server is impossible with our tool as is.

This analysis can be split into four phases:

1. the information concerning the total time of the simulation (excluding uncoupled components), in each process of the coupled system, is gathered in one master process. We deduce the speed and the cost (global and per component) of the simulation, from

the start of the `oasis_init` to the end of the `oasis_terminate` routines (first and last measurement of all component processes) ;

2. the CE timers are gathered in one master process per component, using the local communicator associated to the component. A timeline netCDF file per component (see Appendix 1) is written and contains (i) the starting/ending times of each CE, on each process, (ii) its kind (see Table 1), (iii) the field number in the OASIS internal numbering table, and (iv) the number of the other component exchanging the coupling field. An example of this timeline (kind of CE) is plotted in Figure 3. The script used to produce it with the Ferret visualisation tool [9] is given in Appendix 2 ;

3. the information in one master process per component is processed to keep the maximum, minimum and average values across processes. This operation is done for three quantities per coupling field and all kind of CE. To be more precise, the first quantity is the difference between the latest and the soonest time, across processes, when starting a CE (also called model "jitter"). The second is the difference between the latest time when ending and the soonest time when starting a CE (mainly used to evaluate the "waiting" time). And the last is the difference between the average values of the ending and starting times of a CE ;

4. a synthesis is performed for all components in one master process and written out in separated ASCII text file.

The tasks name, their purpose, the number of call for each task, the location where the task is performed and the number of processes involved, are provided in Table 2.

The operations performed per component in the second and the third phases relies on the existence of an MPI communicator, local to the component process. This communicator is available in OASIS for all processes declaring the same component name. A recent evolution of the library gives the possibility to declare independent partitions (and associated variables) for a subset of a component processes (see Fig 2.2 of the OASIS documentation [10]). In this case, there is no MPI communicator associated to this subset. Without the creation of new partition wide communicators and a complex treatment of overlapping communicators, it is impossible to provide our load balancing analysis in such case. At this development stage, taking into account the few number of prejudiced users and our wish to keep our algorithm as simple as possible, we prefer to stop the load imbalance measurement procedure in case of multiple different partitioning per component.

The information produced by our tool consists in timelines (netCDF format), saved in one file per component, and in a text file synthesis of coupling related key parameters proposed in Section 3. The header of a simple three components timeline is given in Appendix 2. The full output of the analysis produced for the same three component coupled simulation is proposed in Appendix 3.

Task	Purpose	# calls	Calling routine	Coupled processes involved
Initialisation (oasis_lb_init)	Take measurement of the simulation start	1	oasis_init_comp	all
1st array allocation (oasis_lb_allocate)	Allocate the timer header arrays	1	oasis_coupler_setup or oasis_enddef	all
2nd array allocation (oasis_lb_define)	Allocate the timer arrays depending on the operation # associated to the coupling fields	# coupling fields	oasis_coupler_setup	All processes actually exchanging coupling fields
Performance measurement (oasis_lb_measure)	Take elapsed timing before and after a CE and store in the allocated arrays	# CE	Any routine which hosts a CE	All processes involved in a CE
Total time evaluation (in oasis_lb_print)	Measure of the simulation end and produce speed/cost information per component	1	oasis_terminate	all
Components time line (in oasis_lb_print)	Gather timers information and produce one timeline per component			
Summary per component (in oasis_lb_print)	Produce summary numbers per component			
Global summary (in oasis_lb_print)	Gather component information to produce one global analysis and evaluate the total time spent to perform the load balancing analysis			

Table 2: Summary of the load balancing analysis procedure

The synthesis numbers delivered by this analysis are the following:

- the *total time* (s) of the simulation, from the beginning of the first process calling `oasis_init`, to the end of the last process calling `oasis_terminate` ;
- the corresponding *speed* (SYPD) and *cost* (CHPSY) of the coupled system ;
- the *total time* (s) and corresponding *cost* (CHPSY) per component (same measurement definition) ;
- the time (s) spend in *'waiting'* coupled operations (receiving *and* sending coupling fields) and the *'computing'* time. This time is defined as the difference between the *total loop time* and the waiting time. The total loop time is not the total time, but the time spent in the coupling time loop only, excluding the first and the last coupling time steps.

In addition, detailed information is provided for each component:

- a distribution of the "receive" part of the waiting time (s) per corresponding source components (or *counterparts*) ;
- the spread of the time measurement (s) when sending/receiving operation are started between all processes (component '*jitter*') ;
- the ratio (%) of the waiting time compared to the total time ;

- the ratio (%) of this waiting time plus all other OASIS runtime coupling field operations with averaged value (output, input, mapping/interpolation, restarts writing).

The time spent during these OASIS runtime coupling field operations is also detailed, per component, with average values across component processes, and the corresponding jitter when starting the operations, as follows:

- the mapping/interpolation operation timings (s) ;
- the netCDF output operation timings (s) launched for the `namcouple` options OUTPUT, EXPOUT or during restart writings ;
- the netCDF restart writings timings (s) only.

Finally, two informations are provided to check the performance and the validity of the load balancing analysis itself:

- the duration (s) of the load balancing synthesis phase itself ;
- the spread (s) of time clocks measured after an MPI_barrier call, to check the coherence of the clock time between the computing nodes.

Only one process per component is used to gather the whole timing array and to write the timeline. This arrays is also used to perform maximum/minimum/average operations in a single FORTRAN command. A less memory-consuming (but slower and more complex) implementation will be proposed if this solution does not fit the community requirements.

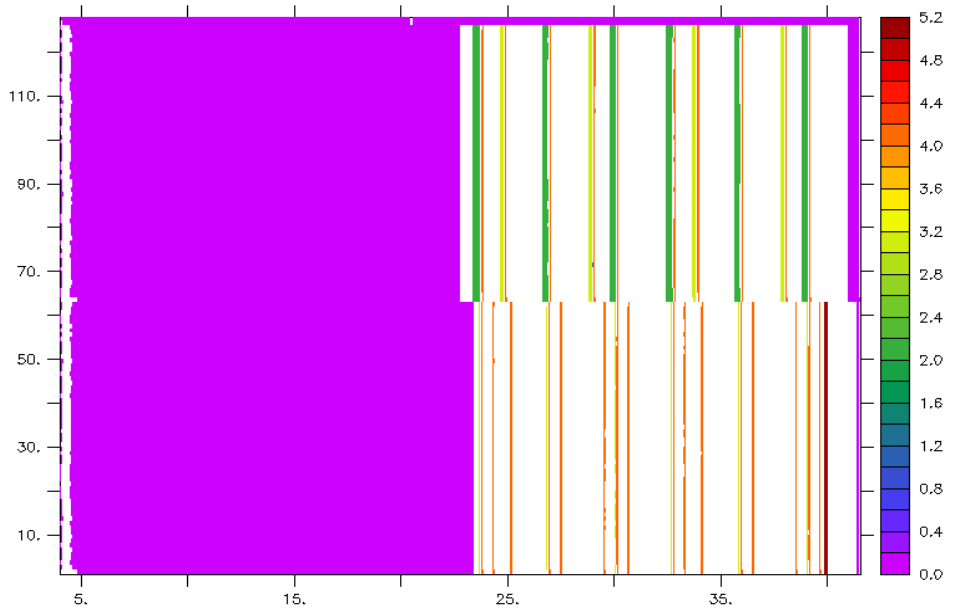

Figure 3: Timeline of kind of CE (x axis for elapsed seconds, y axis for MPI processes), 3 components involved (component 1 : MPI processes 1 to 62; component 2 : MPI processes 63 to 125; component 3, no fields exchanged : MPI processes 126 to 128), initialisation and termination phases in pink, field reception in green, interpolation & mapping in yellow, coupling field netCDF output in orange, netCDF restart file writing in brown

5- Validation

The same tutorial example used at Section 3 is slightly modified to exchange coupling fields on regular global grids and parametrisable resolutions, and to add a third component, which does not exchange any field with the others. The two other components are conserved but included in the same executable. The simulation length is extended to better visualise the exchanges in the timeline. The sequence of send/receive, output and restart operations, together with initialisation and termination phases can be seen in Figure 3.

The corresponding summarised numbers are provided in Appendix 3 and show good agreement with this timeline.

Following the usual procedure [11] to detect a potential side effect of our implementation on other OASIS functions, the load balancing instrumentation is successfully tested on a set of local computers (see Table 3), with and without enabling the load balancing analysis.

Computer	Compiler
Dell Ivy Bridge Core i5-3450	PGI 18.7
Dell Ivy Bridge Core i5-3470	GNUFortran 7.3.1
Lenovo Broadwell Xeon E5-2680-v3	Intel 15.2.164
Lenovo Skylake Xeon Gold 6140	Intel 18.0.1.163

Table 3: List and characteristics of computers used by the automatic OASIS test procedure

The final analysis itself is instrumented to measure its computing performance. Only a fraction of second is necessary to perform the analysis for a single node parallelism (128 processes) of our toy model operated at a global regular ½ degree horizontal resolution and including about 30 CEs per component.

Tests are extended on "`belenos`" to better estimate the scaling properties of our algorithm. In a first step, we increase the parallelism of the coupled components, until using half of the available nodes (512). Figure 4-left clearly shows that the load balancing analysis (i) can be performed without memory or disk access issues using half of one of the most powerful machine in Europe, and (ii) its cost is reasonable (less than a few seconds).

In a second step, for a given MPI decomposition (512 processes for each two components), we increase the number of CEs (i.e. the simulation length) until more than 140,000, which corresponds to a yearly simulation coupling 15 fields every hour. We see on Figure 4-right that the maximum cost of the load balance analysis is not bigger than previously. Our implementation makes possible the simultaneous processing of the new load balance analysis and the original one (`lucia`), which saves the timing traces on disk at runtime. In our case (Figure 4), the `lucia` cost is not as high, but one must remember than this comparison excludes the `lucia` post-processing phase and that the `lucia` traces are limited to 20 processes per component.

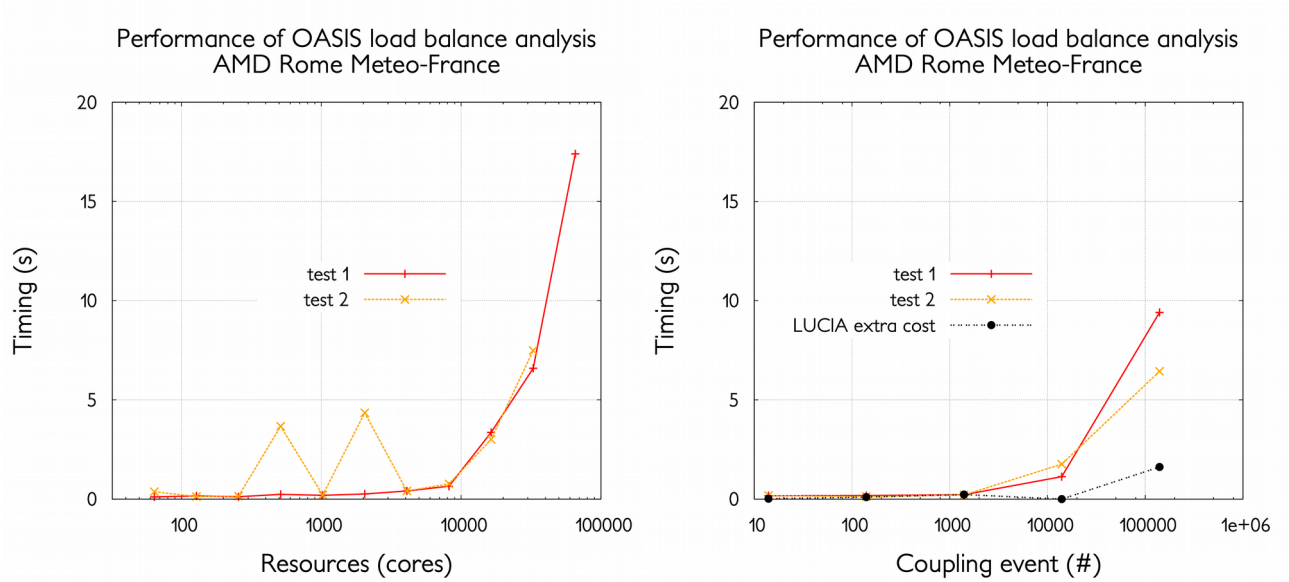

Figure 4: Time to solution (s) of the load balancing algorithm, and the extra time needed when adding the `lucia` analysis, as a function of one component parallelism (# of computing cores) for a simulation of 17 CEs (left), and as a function of the # of simulation CEs for coupled components using 512 cores (4 nodes) (right), measured during 2 test simulations

Another possible blocking issue is the time spent to visualise the timeline. We produced the timeline corresponding to the simulations described in Figure 4-right, using the visualisation tool `Ferret` (without X server, "batch" mode[6]) on our legacy Intel Harpertown (2007) workstation. The image produced is already meaningless with the shortest simulation, because too many information is displayed in the plot. A sub sampling (across processes or CEs) will be mandatory to any user willing to use this visualisation to carefully understand the sequence and the organisation of the CEs. Following this standard requirement, the plot will take less than a few seconds on any up-to-date workstation. Saying that, Figure 5 shows that the time spent to create the plot rapidly increases with the number of CEs, but stays below 10s for plots that already include far too much information.

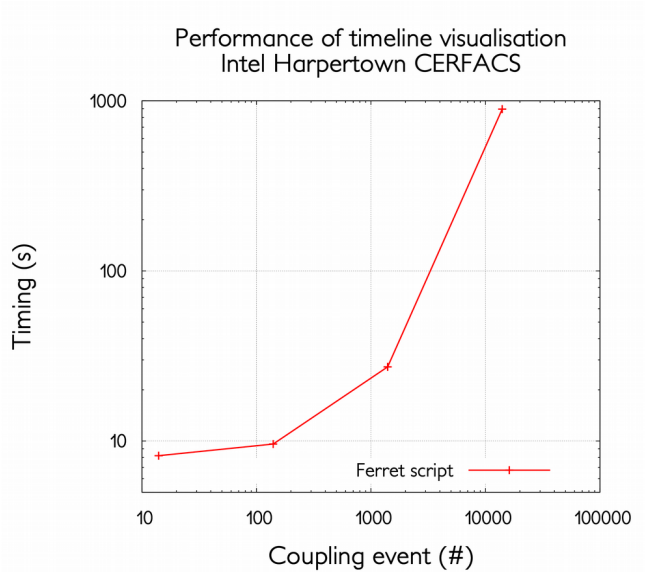

Figure 5: Time needed to visualise the timelines produced with the same simulations than in Figure 4-right

6 `ferret -batch timeline.gif -script timeline_production.jnl`

The Python based `matplotlib` visualisation library has a major advantage compared to `Ferret`: an interactive zoom function (either from the X11 windows displayed, or included in any reader of the vector PDF file that the library can produce). This zoom makes possible the careful examination of one piece of the trace in a large simulation. With our same Harpertown CPU, the production of a PNG file seems slower than the Ferret GIF production. A production in PDF format is even slower[7]. We did not compare X11 display performance with `Ferret` and `matplotlib` but it is clear that, in both case, it can be very painful or even impossible to visualise large series. A pre-selection of time slices seems mandatory to reduce the displaying time. For a detailed documentation of this tool, see [13].

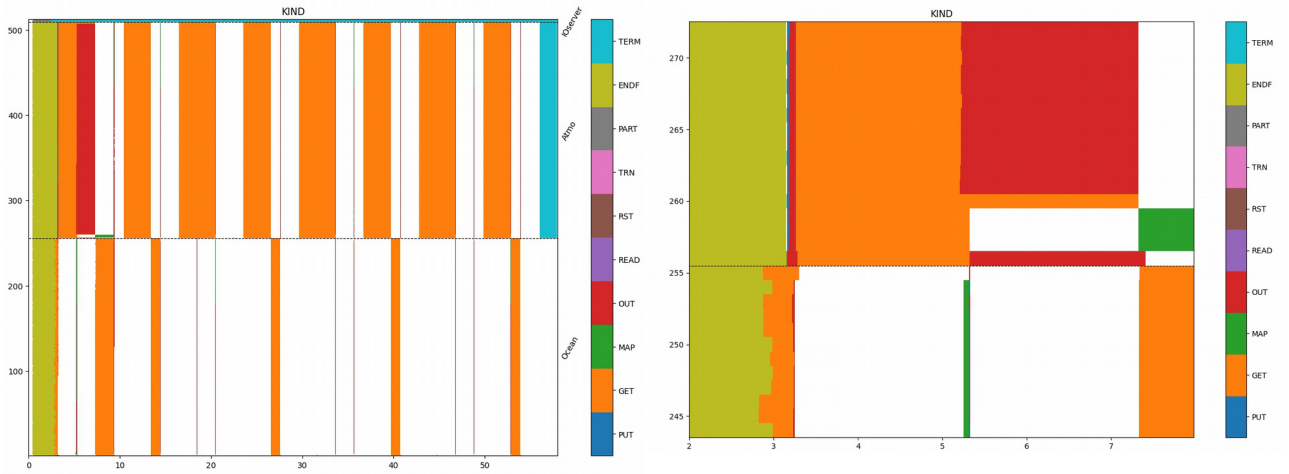

Figure 6 : same kind of timeline than Figure 3, but plotted with `matplotlib`, for the full simulation trace (left) and a process/time slice (right)

6- Conclusion

This implementation proposes an alternate version to the current tool for load balancing measurement in OASIS coupled models (`lucia`). We avoid the production, which could perturb the measurement, and the post-processing analysis of trace files. We deliver at low cost a larger set of summarised quantities that can help to better understand the origin of any coupling extra cost. Moreover, we provide the timeline, for every process involved in the coupling, of every CEs, such as coupling field sending or receiving, but also interpolation, file reading or writing, in addition to the coupling initialisation or termination phases. These timelines are available in a netCDF format file that can be easily and quickly handled with commonly used visualisation tools. To be able to keep the implementation as simple as possible (~ 700 lines), we limit the analysis to a component (and not partition) per-basis, that requires additional memory on each component master process approximately equal to the size of a single precision array containing the number of CE x the number of MPI process. This means that users will have to restrain the analysis to relatively short simulations or reasonably MPI-parallel models to avoid memory overflow.

7 For file production, we were careful not to call the `plt.show matplotlib` function, that is the only one which requires the use of the slowing X11 server. For installation, notice that two extra Python3 packages have to be available : `netcdf4` and, to use the provided Python script, `json` or `yaml`.

Acknowledgements

Computing resources were provided by Météo-France and the German Climate Computing Centre (DKRZ). The authors wish to acknowledge use of the Ferret and matplotlib programs for analysis and graphics in this report. Ferret is a product of NOAA's Pacific Marine Environmental Laboratory. Matplotlib is a Sponsored Project of NumFOCUS, a 501(c)(3) non profit charity in the United States. The project IS-ENES3 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084. The authors are in Sophie Valcke's debt for having reviewed this document.

References

[1] Balaji, V., Maisonnave, E., Zadeh, N., Lawrence, B. N., Biercamp, J., Fladrich, U., Aloisio, G., Benson, R., Caubel, A., Durachta, J., Foujols, M.-A., Lister, G., Mocavero, S., Underwood, S., and Wright, G., 2017: CPMIP: Measurements of Real Computational Performance of Earth System Models in CMIP6 , Geosci. Model Dev., 46, 19-34, doi:10.5194/gmd-10-19-2017

[2] Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., Lüthi, D., Osuna, C., Schär, C., Schulthess, T. C., and Vogt, H., 2018 : Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0, Geosci. Model Dev., 11, 1665-1681, https://doi.org/10.5194/gmd-11-1665-2018

[3] Maisonnave, E. and Caubel, A., 2014: LUCIA, load balancing tool for OASIS coupled systems, Technical Report, TR/CMGC/14/63, SUC au CERFACS, URA CERFACS/CNRS No1875, France

[4] Maisonnave, E. and Valcke, S., 2010: OASIS Dedicated User Support 2009, Annual report, Technical Report, TR/CMGC/10/26, Cerfacs, France

[5] Maisonnave, E., 2020: Ocean/Biogeochemistry macro-task parallelism in NEMO , Working Note, **WN/CMGC/20/31**, CECI, UMR CERFACS/CNRS No5318, France

[6] Maisonnave, E., 2017: IS-ENES2 HighRes ESM performance resulting from OASIS updatesTechnical Report, **TR/CMGC/17/14**, CECI, UMR CERFACS/CNRS No5318, France

[7] Acosta, M., Yepes-Arbos, X., Valcke, S., Maisonnave, E., Serradell, K., Mula-Valls, O. and Doblas-Reyes, F., 2016: Performance Analysis of EC-Earth coupling, Earth Science Dpt, BSC, Spain

[8] Maisonnave, E. and Masson, S., 2019: NEMO 4.0 performance: how to identify and reduce unnecessary communications , Technical Report, **TR/CMGC/19/19**, CECI, UMR CERFACS/CNRS No5318, France

[9] http://ferret.pmel.noaa.gov/Ferret/

[10] Craig, A., Valcke, S. and Coquart, L., 2017: Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0, Geosci. Model Dev., 10, pp 3297-3308, https://doi.org/10.5194/gmd-10-3297-2017

[11] Coquart, L., d'Ast , I. and Valcke, S., 2017: Buildbot : Le logiciel utilisé pour compiler et tester automatiquement les développements réalisés dans le coupleur OASIS3-MCT, Technical Report, **TR-CMGC-17-85**, CECI, UMR CERFACS/CNRS No5318, France

[12] Hunter, J. D., 2007: Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95

[13] Piacentini, A. and Maisonnave, E., 2020: Interactive visualisation of OASIS coupled models load imbalance, TR/CMGC/20/177, CECI, UMR CERFACS/CNRS No5318, France

Appendix 1: Header of a netCDF timeline file

The described component has produced 36 CEs on 62 processes

```
netcdf timeline_ocean {
dimensions:
     nx = 36 ;
     ny = 62 ;
variables:
     float timer_strt(ny, nx) ;
          timer_strt:long_name = "Start of OASIS event" ;
          timer_strt:units = "seconds since OASIS initialisation" ;
     float timer_stop(ny, nx) ;
          timer_stop:long_name = "End of OASIS event" ;
          timer_stop:units = "seconds since OASIS initialisation" ;
     int kind(nx) ;
          kind:long_name = "Kind of OASIS event" ;
          kind:standard_name = "Kind" ;
          kind:flag_values = "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10" ;
          kind:flag_meanings = "UNDF PUT GET MAP OUT READ RST TRN
PART ENDF TERM" ;
          kind:comment = "undefined send receive map file_output
file_input restart_writing partial_restart_writing partition_def
end_def terminate" ;
     int field(nx) ;
          field:long_name = "OASIS field name ID" ;
          field:standard_name = "Field" ;
          field:comment = "Sequence follows namcouple order" ;
     int component(nx) ;
          component:long_name = "Counterpart coupled component
ID" ;
          component:standard_name = "Component" ;
          component:comment = "Sequence follows component MPI rank
in global communicator" ;

// global attributes:
          :source = "OASIS coupler instrumented for load balancing
analysis" ;
          :title = "OASIS event (nx) timeline on every MPI process
(ny)" ;
          :component_id = 2 ;
}
```

Appendix 2 : FERRET script for timeline visualisation

To visualise three timelines (kind of CE) coming from three components of a coupled system

---

```
use timeline_pam_.nc
use timeline_pim_.nc
use timeline_poum.nc

LET YRECTANGLE = YSEQUENCE({0,0,1,1})

let comm1_size = `timer_strt[d=1],RETURN=jsize`
let comm2_size = `timer_strt[d=2],RETURN=jsize`
let comm3_size = `timer_strt[d=3],RETURN=jsize`
let totcommsize = `comm1_size`+`comm2_size`+`comm3_size`

let YPTS1 = I[d=1,gx=kind]*0
let YPTS2 = I[d=2,gx=kind]*0+`comm1_size`
let YPTS3 = I[d=3,gx=kind]*0+`comm1_size`+`comm2_size`

let dim_1 = YSEQUENCE({1,0,0,1})
let dim_2 = YSEQUENCE({0,1,1,0})

let to_plot1 if kind ge 8 then 0 else kind
let to_plot if to_plot1 eq 7 then 5 else to_plot1

!let to_plot = field
!let to_plot = component

let min2min =
MIN(MIN(timer_strt[i=@min,j=@min,d=1],timer_strt[i=@min,j=@min,d=2]),timer_strt[
i=@min,j=@min,d=3])
let max2max =
MAX(MAX(timer_stop[i=@max,j=@max,d=1],timer_stop[i=@max,j=@max,d=2]),timer_stop[
i=@max,j=@max,d=3])

POLYGON/xlim=`min2min`:`max2max`/ylim=1:`totcommsize`/nolab
timer_strt[d=1,j=1]*dim_1+timer_stop[d=1,j=1]*dim_2, YRECTANGLE+YPTS1+1,
to_plot[d=1]

repeat/RANGE=2:`comm1_size`/NAME=index
( POLYGON/xlim=`min2min`:`max2max`/ylim=1:`totcommsize`/ov/nolab/lev
timer_strt[j=`index`,d=1]*dim_1+timer_stop[j=`index`,d=1]*dim_2,
YRECTANGLE+YPTS1+`index`, to_plot[d=1]; PPL SHASET RESET )

repeat/RANGE=1:`comm2_size`/NAME=index
( POLYGON/xlim=`min2min`:`max2max`/ylim=1:`totcommsize`/ov/nolab/lev
timer_strt[d=2,j=`index`]*dim_1+timer_stop[d=2,j=`index`]*dim_2,
YRECTANGLE+YPTS2+`index`, to_plot[d=2]; PPL SHASET RESET)

repeat/RANGE=1:`comm3_size`/NAME=index
( POLYGON/xlim=`min2min`:`max2max`/ylim=1:`totcommsize`/ov/nolab/lev
timer_strt[j=`index`,d=3]*dim_1+timer_stop[j=`index`,d=3]*dim_2,
YRECTANGLE+YPTS3+`index`, to_plot[d=3]; PPL SHASET RESET)
```

Appendix 3: Load imbalance analysis (summarised text information)

```
 ------------------------------

 Coupled model simulation time (s):   41.527
    (coupled components only)

 ------------------------------

Speed (SYPD) : 379707.221
Cost (CHPSY) :    0.008

Model       ocean simulation time :   41.527
                 cost (CHPSY):    0.004
Model       atmosphere simulation time :   41.525
                 cost (CHPSY):    0.004
Model       ioserver simulation time :   41.522
                 cost (CHPSY):    0.000
 ------------------------------

     Load balance analysis

 ------------------------------

 Model    /   Computing time  /  Waiting time

     ocean /    7.625 /    1.818
     atmosphere /    9.742 /    0.001
     ioserver /    0.000 /    0.000

 ------------------------------
     Additional information
 ------------------------------
 ------------------------------
        ocean
 ------------------------------
 Specific oasis_get time
 (n/a if no oasis_get)
       from model       atmosphere
       :    1.818
 --------------------
Total jitter :    0.135

Partial coupling cost (%) :  19.25
Partial coupling cost including OASIS operations (%) :  34.60

OASIS Operations :
 --------------------
Total mapping/interpolation :    0.642
    with spread :    0.082
Total Netcdf output (OUTPUT+EXPOUT+restart):    0.807
    with spread :    0.192
including restart :    0.000
    with spread :    0.000
------------------------------
         atmosphere
 ------------------------------
 Specific oasis_get time
 (n/a if no oasis_get)
       from model       ocean
       :    0.000
 --------------------
Total jitter :    0.196

Partial coupling cost (%) :   0.01
Partial coupling cost including OASIS operations (%) :  18.33

 OASIS Operations :
 --------------------
Total mapping/interpolation :    0.324
    with spread :    0.062
Total Netcdf output (OUTPUT+EXPOUT+restart):    1.461
    with spread :    0.339
including restart :    0.165
    with spread :    0.007

 ------------------------------
         ioserver
 ------------------------------
 Specific oasis_get time
 (n/a if no oasis_get)
 --------------------
Total jitter :    0.000
Partial coupling cost (%) :   0.00
Partial coupling cost including OASIS operations (%) :   0.00

 OASIS Operations :
 --------------------
Total mapping/interpolation :    0.000
    with spread :    0.000
Total Netcdf output (OUTPUT+EXPOUT+restart):    0.000
    with spread :    0.000
including restart :    0.000
    with spread :    0.000

 ------------------------------

 Total time of this load balancing analysis:
      :    0.228

 ------------------------------
```