

Study of the overhead
due to the exchanges of data
compared to the computation in a model
in a coupling context

*Damien Declat *- Sophie Valcke * - Reiner Vogelsang #*

draft - February 4th, 2002

The overhead associated to the exchanges of data is certainly one of the most important criteria that could influence the design of the PRISM coupled system. Therefore, it was important to have a first guess of the efficiency of the different techniques of exchange on different machines, to compare them and finally to give a first estimation of the overhead associated to the exchange in the coupling context.

** Global Change and Climate Modelling Team, CERFACS
SGI Germany*

I -The different techniques of exchange

The following list of the different techniques is not an exhaustive one. It just gathers some possibilities judged realistic for the future implementation.

1.1 The "common" technique

The common statement defines a block of main memory storage so that different units can share the same data. In a coupling context, the component models to be coupled are different routines called sequentially and form one unique executable; the coupling fields are global arrays declared in the common statement.

- Advantages:
 - Efficient exchange of coupling information in the memory.
 - If the models are "sequential by construction" (see PRISM glossary), optimal use of the resources and simplest configuration.
- Disadvantages:
 - Loss of flexibility: the coupling fields must be the same global arrays in both components..
 - As both components form one unique executable, there are possible I/O, name space and internal communication conflicts, if no special care is taken.
 - If the components are not "sequential by construction", implies splitting them into a series of sequential sub-components.

In the test performed (see MPI2/model1.f90 or MPI1/test_1/common_model1.f90), the main program represents the first component and the second component consists in one subroutine called by the main. The coupling global arrays are declared in the main program and in the subroutine; they are initialised in the subroutine and verified back in the main program. The timing includes the initialisation of the common block and the verification of the coupling arrays. To have a significant timing for each test, a high number of exchanges are performed in a loop sequence.

An additional test was also performed (see MPI1/test_1_revised/common_model1.f90). In this test, the array of the common block was extended by an additional dimension to implement a so-called circular buffer. For each exchange in the loop, the index of that last new dimension is increased and set back again to one if the index exceeds the last dimension. On the SGI O3800, the last dimension was chosen such that the size of the array exceeds the L2-cachesize three times. The toggling of start addresses ensures that data are taken from the memory which reflects the situation in large application codes. This was not necessarily the case in the original test for which all examples fit into the L2-cache. Even the largest example, ARPEGE chem, declares a common block of only 6,17 MB which is smaller than the L2-cache size of 8 MB. Given the tight iteration loop and the fact that a write-back cache keeps an allocated cacheline as long as a replacement

happens due to a process migration, external invalidation or replacement by another cacheline, unreasonable high transfer rates -which do not reflect the situation of large application codes- were observed in the original test. For the Fujitsu VPP5000 and the NEC SX5 vector processors, there is no cache and therefore no possible cache effect, as shown by the results of the original and revised tests which are almost identical.

1.2 The memory copies

With this technique, the exchange implies that the content of a local variable is copied into a global variable by one subroutine, and then copied from the global variable to another local variable by the other subroutine. The global variable is declared in a common statement or in a F90 module used by both subroutines. In a coupling context, the component models to be coupled are different routines called sequentially and form one same executable (as in I.1 the “common” technique); the coupling fields are declared as local variables.

- Advantages:
 - Less efficient than the “common” technique but may still be more efficient than message passing on some architectures.
 - As for I.1, if the models are "sequential by construction" (see PRISM glossary), optimal use of the resources and simpler configuration.
 - Flexibility: the coupling fields are local variables and the model coupling interface library below each send/receive instruction implemented in the model code could do the copy into the global variable automatically.
- Disadvantages:
 - As for I.1: as both component models form one same executable, there are possible I/O, name space and internal communication conflicts, if no special care is taken.
 - As for I.1: if the components are not "sequential by construction", it implies splitting them into a series of sequential sub-components.
 - Declaration of additional global variables for the exchange requires extra memory.

The test consisted in calling successively two subroutines in the same executable (see MPI2/model1.f90 or MPI1/test_2/mem_copy_model1.f90). A module containing the global variable is defined and used in both subroutines. In the source subroutine, the local coupling field is first initialised and then copied in the common global variable, while in the target subroutine the content of the global variable is copied in a local coupling field and verified. The timing starts before calling the source subroutine and ends after calling the target subroutine and therefore includes the initialisation of the local coupling field in the source subroutine and the verification of the local coupling field in the target subroutine.

An additional test to prevent specific cache behaviour similar to what is described above for I.1 was also performed (see MPI1/test_2_revised/mem_copy_model1.f90).

1.3 The point to point MPI communications, using MPI_SSend/ MPI_Recv

With this technique, the exchange is done through synchronous message passing (MPI_SSend /MPI_Recv primitives). In a coupling context, it allows the component models to be different executables running in parallel.

- Advantages:
 - As each component model is a different executable, there are no possible I/O or name space conflicts.
 - With MPI2, no possible internal communication conflicts.
 - Flexibility: the coupling fields are local variables exchanged through message passing
 - Component models are easily interchangeable.
 - Optimal configuration if the components are naturally parallel.
- Disadvantages:
 - If components are "sequential by construction", may imply a waste of resources on some architectures.
 - Harder to load balance than a sequential configuration.
 - With MPI1, internal communication conflicts are possible if no special care is taken.
 - Depending on the MPI implementation, could result in deadlocks if the sequence of exchanges is different on the source side from the one on the target side.

In the test, the component models to be coupled are two separate two executables (see MPI2/model4.f90 and model5.f90, or MPI1/test_4b/atm_model4.f90 and ocean_model5.f90). With MPI2, the models are spawned by the coupler and gathered in the same communicator (using successive MPI_Comm_Spawn) in order to exchange data. (One can notice here that in case of dynamic strategy for the coupler, this technique implies that the communicator will have to be destroyed exactly in the opposite way it has been created). With MPI1, the two models are started initially in parallel with the mpisx command (or equivalent on other architectures). The timing was performed in one model (model4.f90) just before and after a Ping-Pong test: the coupling fields are initialised in the model, sent to the other model which sends them back to the original model where they are verified. In the context of the tests, the difference between the MPI_Send and MPI_SSend was not significant.

1.4 The point to point buffered MPI communications, using MPI_IBSend and MPI_IRecv

With this technique, the exchange is done through buffered asynchronous message passing (MPI_IBSend and MPI_IRecv with corresponding MPI_Wait primitives). The structure of this test is similar to the I.3 test. As in I.3, the component models are two separate executables (see MPI2/model2.f90 and model3.f90, or MPI1/test_3b/atm_model2.f90 and ocean_model3.f90). The advantages and disadvantages of this technique are similar to the ones of I.3, with the additional advantage that it is not subject to deadlocks even if the sending sequence on the source

side is different from the receiving sequence on the target side; this advantage remains true regardless of the implementation as the messages are explicitly buffered. One can note that the possibility to use the spare cycles between issuing the send/recv operation and waiting for the request to finish for extra computation is not exploited: the MPI_Wait calls follows immediately after the send/recv calls.

II - Experiences

The experiences consisted in comparing the four main techniques of exchange described above on a Fujitsu VPP5000, a NEC SX5, and an SGI O3800:

- “common” technique (original and revised test, see I.1)
- memory copy technique (original and revised test, see I.2)
- MPI1 or MPI2 synchronous communications, using MPI_SSend/MPI_Recv
- MPI1 or MPI2 asynchronous buffered communications, using MPI_IBSend/MPI_IRecv

The data are 'realistic' 2D or 3D coupling fields as described just after. To have a significant timing, a high number of exchanges (more than 1000) were performed in a loop for each test.

II.1 - The sets of fields

The tests were realised on realistic fields:

Mozart (atmospheric chemistry)	128x64x34
Mozart/64 * (atmospheric chemistry)	16x8x34
ARPEGE land scheme	500x500
Bio high: MPI Biogeochemistry high resolution	128x211x23
Bio high/8**	64x53x23
Bio low: MPI Biogeochemistry low resolution	120x110x20
Bio low/8***	30x55x20
ARPEGE chem: (atmospheric chemistry)	180x90x50

*a distribution of the first and second dimensions on 8 processes each was supposed.

**a distribution of the first and second dimensions on 2 and 4 processes each was supposed

***a distribution of the first and second dimensions on 4 and 2 processes each was supposed

II.2 - Results

The collected results for the different techniques on the different machines are presented in the Annex I as elapse time in seconds, representing the time needed to initialise, exchange and verify one field, and corresponding initialisation-transfer-verification rate in Mbytes/seconds; the tables also show the ratio of the initialisation-transfer-verification rates for the different techniques.

II.2.1 Remarks on the NEC SX5 results

- For the “common” and the memory copy techniques, the original and revised tests practically lead to the same results, as can be expected on these vector processors with no cache.
- The “common” technique is only 1,2 times more efficient than the MPI synchronous communication (SSend) on average. One has to note that in the MPI synchronous communication tests on the SX5, the two different executables were running on the same node; it shows that “intranode” message passing is very efficiently implemented on the SX5.
- The maximum initialisation-transfer-verification rate observed in our MPI synchronous communication (SSend) tests is about 8 Gbytes/sec. The maximum MPI_Send/MPI_Recv intranode transfer rate is about 18,8 Gbytes/sec for the SX5 (cf. <http://www.ccr1-nece.de/~ritzdorf>), the effective transfer rate depending of course on the size of the message. The difference is due to the size of the fields and to the fact that our measurements also include initialisation and verification of the arrays. For the SX6, the MPI_Send/MPI_Recv intranode transfer rate will be about 14,6 Gbytes/sec.
- The MPI synchronous communication (SSend) is only 1,1 times more efficient than the MPI asynchronous buffered communication.
- The “common” technique is on average 2,1 times more efficient than the memory copy technique.
- The memory copy initialisation-transfer-verification rate is on average 0,6 times the one of the MPI synchronous communication (SSend) or asynchronous buffered communication (IBSend).

II.2.2 Remarks on the Fujitsu VPP5000 results

- For the “common” and the memory copy techniques, the original and revised tests practically lead to the same results, as can be expected on these vector processors with no cache.
- The maximum initialisation-transfer-verification rate obtained in the “common” test is about 16 Gbytes/sec, which represents about 1/5 of the 78 Gbytes/sec memory bandwidth; the difference is due to the fact that our measurements also include initialisation and verification of the arrays.
- The “common” technique is 11,8 times more efficient than the MPI synchronous communication (SSend) and 10,1 times more efficient than the MPI asynchronous buffered communication (IBSend) on average.

- For the MPI communication tests on the VPP5000, the two different executables were necessarily running on different processors communicating via the 1,6 Gbytes/sec cross-bar. For large fields, the maximum initialisation-transfer-verification rate obtained in the tests is quite close to this 1,6 Gbytes/sec, which is reasonable.
- The MPI synchronous communication (SSend) is almost as efficient as the MPI asynchronous buffered communication (IBSend).
- The “common” technique is on average 2,4 times more efficient than the memory copy technique.
- The memory copy technique is on average 5,5 more efficient than the MPI synchronous communication (SSend) , and 4,1 more efficient than the MPI asynchronous buffered communication (IBSend).

II.2.3 Remarks on the SGI O3800 results

- On the SGI O3800, the usage of two separate executables implied that the communication between the two MPI tasks is done via sockets as a default of the current MPI implementation. However, by turning on the capability of the so-called block transfer engine in each compute node MPI is using shared memory as a transport layer (IRIX64 6.5.14f and higher). The MPI environment was set as follows:
 - MPI_XPMEM_ON
 - MPI_XPMEM_THRESHOLD=16384
 - MPI_DSM_MUSTRUN
 - MPI_DSM_VERBOSE
 - SMA_GLOBAL_ALLOC

The next MPI release proposed for April 2002 will not need that additional environment setting anymore.

- For the “common” technique, the results show on average a decrease of performance of about a factor of 2,39 for the revised test compared with the cache bound results of the original test.
- For the memory copy, the results show on average a decrease of performance of about a factor of 1,39 for the revised test compared with the cache bound results of the original test. The results show that the effect of forcing the data to and from the memory is more visible for smaller fields: for 16x8x34 fields the effect is quite noticeable, whereas for 180x90x50 fields the effect is nearly visible.
- The two preceding remarks show that the revised tests should be included into the test suite since all cache based scalar RISC processors will show similar results as observed during the current measurements.
- The “common” technique is on average 2,6 times more efficient than the MPI synchronous communication (SSend), and 3,4 times more efficient than the MPI asynchronous buffered communication (IBSend). In general, this advocates for doing things locally as long as possible.
- The MPI synchronous communication (SSend) is 1,3 times more efficient than the MPI asynchronous buffered communications (IBSend). Therefore, if one knows the relationship between processes with regard to the flow of data one should use that knowledge to favour synchronous communication

over buffered or buffered asynchronous schemes if the scheduling of data transfers allows it. However, one may gain overall speedup if asynchronous communication can be overlapped with computation.

- The revised “common” technique is on average 2,5 times more efficient than the revised memory copy technique. In general, this advocates for trying to avoid unnecessary data movement from and to memory.
- The memory copy technique is on average 1,1 more efficient than the MPI synchronous communication (SSend), and 1,3 more efficient than the MPI asynchronous buffered communication (IBSend).
- Unfortunately the tests do not cover one-sided communication like MPI_PUT/GET. However, it seems to be difficult to implement that communication scheme into a coupler context.

III - First estimation of message passing overhead for some particular cases

A first estimation of MPI communication overhead for some particular cases is presented hereafter in Table 1 below. To calculate the overhead, the elapse time needed for the exchanging a given number of fields at a given frequency has to be compared to the elapse time of the models.

III.1 Preliminary remarks

- We supposed here that the different fields are not packed into one MPI message, which could of course still increase the efficiency of the MPI exchange.
- In all these examples, the overhead corresponds to one exchange per time step.
- For each example, the table gives:
 1. the message passing overhead per field for model1 calculated by comparing the additional elapse time required for the exchange of one field to the elapse time of the model1
 2. the message passing overhead per field for model2 calculated by comparing the additional elapse time required for the exchange of one field to the elapse time of the model2
 3. the total message passing overhead per field calculated by comparing the additional elapse time required for the exchange of one field to the cumulated elapse time of model1 and model2
 4. the message passing overhead for model1 for N fields calculated by comparing the additional elapse time required for the exchange of N fields to the elapse time of the model1
 5. the message passing overhead for model2 for N fields calculated by comparing the additional elapse time required for the exchange of N fields to the elapse time of the model2
 6. the total message passing overhead for N fields calculated by comparing the additional elapse time required for the exchange of N fields to the cumulated elapse time of model1 and model2

III.2 Particular remarks for each example

1- MPI high resolution ocean and bgc models – SX5 intranode

- I supposed here that the MPI high resolution bgc run each one on one processor of the same SX5 node.
- The elapse times are extrapolated from SX4 figures, with the hypothesis that the elapse time on the SX5 is 44% the elapse time on the SX4.
- The 16 Gbytes/sec transfer rate is an MPI_IBSend/MPI_Irecv transfer rate I measured for 128x211x23 fields (not including initialisation or verification of the arrays). It is close to the maximum transfer rate of 18,8 Gbytes/sec.

2- MPI high resolution ocean and bgc models on 8 processors – SX5 intranode

- I supposed here that the MPI high resolution ocean model and MPI high resolution bgc run each one on 8 processors of the same SX5 node.

- The fields are 8 times smaller than for 1- (first dimension is divided by 2, second dimension is divided by 4).
- The elapse time is the elapse time from 1- divided by 8.
- The 9 Gbytes/sec transfer rate is the MPI_IBSend/MPI_Irecv transfer rate I measured for 64x53x23 fields (not including initialisation or verification of the arrays).

3- MPI low resolution ocean and bgc models – SX5 intranode

- I supposed here that the MPI low resolution ocean model and MPI low resolution bgc run each one on one processor of the same SX5 node
- The elapse times are extrapolated from SX4 figures, with the hypothesis that the elapse time on the SX5 is 44% the elapse time on the SX4.
- The 11 Gbytes/sec transfer rate is the MPI_IBSend/MPI_Irecv transfer rate I measured for 120x110x20 fields (not including initialisation or verification of the arrays).

4- MPI low resolution ocean and bgc models on 8 processors – SX5 intranode

- I supposed here that the MPI low resolution ocean model and MPI low resolution bgc run each one on 8 processors of the same SX5 node.
- The fields are 8 times smaller (first dimension is divided by 4, second dimension is divided by 2).
- The elapse time is the elapse time from 3- divided by 8.
- The 7 Gbytes/sec transfer rate is the MPI_IBSend/MPI_Irecv transfer rate I measured for 30x55x20 fields (not including initialisation or verification of the arrays).

5- MPI high resolution ocean and bgc models – SX5 internode

- I supposed here that the MPI high resolution ocean model and MPI high resolution bgc run each one on one processor of different SX5 nodes
- The elapse times are extrapolated from SX4 figures, with the hypothesis that the elapse time on the SX5 is 44% the elapse time on the SX4.
- The 5,6 Gbytes/sec transfer rate is the maximum internode MPI_Send/MPI_Recv transfer rate (cf H. Ritzdorf).

6- MPI high resolution ocean and bgc models on 8 processors– SX5 internode

- I supposed here that the MPI high resolution ocean model and MPI high resolution bgc run each one on 8 processors of different SX5 nodes.
- The elapse time is the elapse time from 1- divided by 8.
- The fields are not 8 times smaller as whole fields have to transfer from one node to the other at the internode transfer rate of 5,6 Gbytes/sec (maximum internode MPI_Send/MPI_Recv transfer rate, cf H. Ritzdorf).

7- Arpege (500x500x41) and Arpege Land Scheme - VPP5000

- I supposed here that Arpege (500x500x41) runs on one VPP5000 processor, and its Land Scheme runs on another one.
- The elapse time comes from a figure given by F. Bouttier : 5400 sec for 416 time steps for a Arpege 600x300x41 configuration. For 500x500x41, we estimated the cpu time to 18 sec/timestep. The land scheme requires about 10% the cpu time of Arpege.
- The 1,6 Gbytes/sec transfer rate is the MPI_SSend/MPI_Recv transfer rate I measured for 500x500 fields (including or not initialisation or verification of the arrays makes no real differences).

8- Arpege (180x90x50) and MOCAGE – VPP5000

- I supposed here that Arpege (180x90x50) runs on one VPP5000 processor, and the atmospheric chemistry model MOCAGE runs on another one.
- The elapse time for MOCAGE comes from a figure given by V.-H. Peuch : 15 min for 6 hours of simulation.
- A time step of 20 minutes was supposed.
- The 1,44 Gbytes/sec transfer rate is the MPI_SSend/MPI_Recv transfer rate I measured for 180x90x50 fields (including or not initialisation or verification of the arrays makes no real differences).

9- Arpege (180x90x50) and MOCAGE on 8 processors - VPP5000

- I supposed here that Arpege (180x90x50) runs on 8 VPP5000 processor, and the atmospheric chemistry model MOCAGE runs on 8 others.
- The elapse times are the ones from 8- divided by 8.
- The 1,4 Gbytes/sec transfer rate is the MPI_SSend/MPI_Recv transfer rate I measured for 45x45x50 fields (including or not initialisation or verification of the arrays makes no real differences).

10- Arpege (180x90x50) and simplified MOCAGE - VPP5000

- I supposed here that Arpege (180x90x50) runs on one VPP5000 processor, and a simplified version atmospheric chemistry model MOCAGE runs on another one.
- The elapse time for MOCAGE is the one from 8- divided by a factor of 5.
- The 1,4 Gbytes/sec transfer rate is the MPI_SSend/MPI_Recv transfer rate I measured for 180x90x50 fields (including or not initialisation or verification of the arrays makes no real differences).

11- NCAR atmosphere and Mozart – IBM SP3

- The overhead calculated here is the one estimated for coupling the Mozart atmospheric chemistry model to the NCAR atmospheric model every time step.
- The models run on 16 nodes of 4 processors each on an IBM SP-3.
- The resolution of the coupling fields is 128x64x34. Each node has to transfer the equivalent of 32x16x34 fields.
- We supposed that the exchange implies 50 3D fields from the atmosphere to the atmospheric chemistry and back to the atmosphere every time step.
- A total inter-node transfer rate of 50 Mbytes/sec was supposed.

12- MPI low resolution ocean and bgc models – SGI O3800

- The elapse times for the models are figures measured by Reiner Vogelsang.
- The 131 Mbytes/sec transfer rate is the MPI_IBSend/MPI_Irecv transfer rate measured by Reiner for 120x110x20 fields on the SGI O3800.
- We supposed that 20 fields are exchanged every time step.

III.3 Overhead results –Table 1

	Platform	Model1	Model2	Grid	Total pts	Elapse time model1 (sec)	Elapse time model2	Total elapse time
1-	SX5-Intranode	MPI ocean high	MPI bgc high	128x211x23	621184	0,84	0,94	1,77
2-	SX5-Intranode	MPI ocean high/8	MPI bgc high/8	64x53x23	78016	0,10	0,12	0,22
3-	SX5-Intranode	MPI ocean low	MPI bgc low	120x110x20	264000	0,38	0,47	0,84
4-	SX5-Intranode	MPI ocean low/8	MPI bgc low/8	30x55x20	33000	0,05	0,06	0,11
5-	SX5-Internode	MPI ocean high	MPI bgc high	128x211x23	621184	0,84	0,94	1,77
6-	SX5-Internode	MPI ocean high/8	MPI bgc high/8	128x211x23	621184	0,10	0,12	0,22
7-	VPP5000	Arpege	Land scheme	500x500	250000	18,03	1,80	19,83
8-	VPP5000	Arpege	MOCAGE	180x90x50	810000	2,28	50,00	52,28
9-	VPP5000	Arpege/8	MOCAGE/8	45x45x50	101250	0,28	6,25	6,53
10-	VPP5000	Arpege	MOCAGE simplified	180x90x50	810000	2,28	10,00	12,28
11-	IBM SX3	Atm/16 nodes	Mozart/16 nodes	32x16x34	17408			2,00
12-	SGI O3800	MPI ocean low	MPI bgc low	120x110x20	264000	5,31	6,69	12,00

	Platform	Model1	Model2	Transfer rate (Mbytes/sec)	overhead/field for model1 (%)	overhead/field for model2 (%)	total overhead/field (%)	nbr of field	overhead for model1 (%)	overhead for model2 (%)	total overhead (%)
1-	SX5-Intranode	MPI ocean high	MPI bgc high	16000	0,04	0,03	0,02	20	0,74	0,66	0,35
2-	SX5-Intranode	MPI ocean high/8	MPI bgc high/8	9000	0,07	0,06	0,03	20	1,33	1,18	0,63
3-	SX5-Intranode	MPI ocean low	MPI bgc low	11000	0,05	0,04	0,02	20	1,01	0,82	0,45
4-	SX5-Intranode	MPI ocean low/8	MPI bgc low/8	7000	0,08	0,06	0,04	20	1,59	1,29	0,71
5-	SX5-Internode	MPI ocean high	MPI bgc high	5600	0,11	0,09	0,05	20	2,12	1,89	1,00
6-	SX5-Internode	MPI ocean high/8	MPI bgc high/8	5600	0,85	0,76	0,40	20	16,98	15,15	8,01
7-	VPP5000	Arpege	Land scheme	1600	0,01	0,07	0,01	20	0,14	1,39	0,13
8-	VPP5000	Arpege	MOCAGE	1440	0,20	0,01	0,01	20	3,95	0,18	0,17
9-	VPP5000	Arpege/8	MOCAGE/8	1400	0,20	0,01	0,01	20	4,06	0,19	0,18
10-	VPP5000	Arpege	MOCAGE simplified	1440	0,20	0,05	0,04	20	3,95	0,90	0,73
11-	IBM SX3	Atm/16 nodes	Mozart/16 nodes	50	#DIV/0!	#DIV/0!	0,14	100	#DIV/0!	#DIV/0!	13,93
12-	SGI O3800	MPI ocean low	MPI bgc low	131	0,30	0,24	0,13	20	6,07	4,82	2,69

ANNEXE I

The results collected for the different techniques on the different machines are presented here as elapse time in seconds, representing the time needed to initialise, exchange and verify one field, and corresponding initialisation-transfer-verification rate in Mbytes/seconds; the tables also show the ratio of the initialisation-transfer-verification rates for the different techniques.

SGI O3800

Model	Grid	Total pts	Common	Common revised	Memory copy	Memory copy revised	SSend-Recv	IBSend-Irecv
Initialisation-transfer-verification elapse time (seconds):								
Arpege chem	180x90x50	810000	7,22E-03	1,33E-02	5,01E-02	5,13E-02	3,78E-02	5,40E-02
Bio high	128x211x23	621184	5,39E-03	1,12E-02	3,91E-02	4,07E-02	2,73E-02	4,13E-02
Mozart	128x64x34	278528	2,45E-03	6,11E-03	9,67E-03	1,26E-02	1,23E-02	1,69E-02
Bio low	120x110x20	264000	1,94E-03	4,97E-03	7,20E-03	1,09E-02	1,25E-02	1,61E-02
Arpege land	500x500	250000	1,81E-03	4,31E-03	6,48E-03	9,88E-03	1,18E-02	1,59E-02
Bio high/8	64x53x23	78016	7,03E-04	1,59E-03	2,36E-03	3,31E-03	4,11E-03	4,94E-03
Bio low/8	30x55x20	33000	2,50E-04	6,09E-04	7,81E-04	1,19E-03	1,57E-03	2,17E-03
Mozart/64	16x8x34	4352	3,13E-05	1,09E-04	1,25E-04	1,88E-04	2,97E-04	3,20E-04
Initialisation-transfer-verification rate (Mbytes/sec):								
Arpege chem	180x90x50	810000	898	489	129	126	171	120
Bio high	128x211x23	621184	922	443	127	122	182	120
Mozart	128x64x34	278528	908	365	230	177	181	132
Bio low	120x110x20	264000	1090	425	293	194	169	131
Arpege land	500x500	250000	1103	464	308	203	169	126
Bio high/8	64x53x23	78016	888	392	265	188	152	126
Bio low/8	30x55x20	33000	1056	433	338	222	168	122
Mozart/64	16x8x34	4352	1114	318	279	186	117	109
Ratio:			Comm rev/ Mem copy rev	Comm rev/ SSend	Comm rev/ IBSend	Mem copy rev/ SSend	Mem copy rev/ IBSend	SSend/ IBSend
			3,9	2,9	4,1	0,7	1,1	1,4
			3,6	2,4	3,7	0,7	1,0	1,5
			2,1	2,0	2,8	1,0	1,3	1,4
			2,2	2,5	3,2	1,1	1,5	1,3
			2,3	2,7	3,7	1,2	1,6	1,3
			2,1	2,6	3,1	1,2	1,5	1,2

	1,9	2,6	3,6	1,3	1,8	1,4
	1,7	2,7	2,9	1,6	1,7	1,1
Average ratio:	2,5	2,6	3,4	1,1	1,4	1,3

NEC SX5

Model	Grid	Total pts	Common	Common revised	Memory copies	Memory copies revised	SSend-Recv	IBSend-Irecv
Initialisation-transfer-verification elapse time (seconds):								
Arpege chem	180x90x50	810000	6,14E-04	6,14E-04	1,37E-03	1,31E-03	8,12E-04	1,00E-03
Bio high	128x211x23	621184	6,07E-04	6,04E-04	1,25E-03	1,30E-03	7,50E-04	8,75E-04
Mozart	128x64x34	278528	2,71E-04	2,71E-04	5,00E-04	5,73E-04	3,12E-04	3,75E-04
Bio low	120x110x20	264000	2,75E-04	2,74E-04	5,00E-04	5,72E-04	3,75E-04	3,75E-04
Arpege land	500x500	250000	1,39E-04	1,37E-04	3,00E-04	3,32E-04	2,50E-04	2,50E-04
Bio high/8	64x53x23	78016	1,53E-04	1,53E-04	3,25E-04	3,13E-04	1,75E-04	1,75E-04
Bio low/8	30x55x20	33000	1,37E-04	1,38E-04	3,00E-04	2,79E-04	1,13E-04	1,38E-04
Mozart/64	16x8x34	4352	3,50E-05	3,48E-05	7,50E-05	7,07E-05	3,75E-05	4,00E-05

Initialisation-transfer-verification rate (Mbytes/sec):

Arpege chem	180x90x50	810000	10554	10554	4730	4947	7980	6480
Bio high	128x211x23	621184	8187	8228	3976	3823	6626	5679
Mozart	128x64x34	278528	8222	8222	4456	3889	7142	5942
Bio low	120x110x20	264000	7680	7708	4224	3692	5632	5632
Arpege land	500x500	250000	14388	14599	6667	6024	8000	8000
Bio high/8	64x53x23	78016	4079	4079	1920	1994	3566	3566
Bio low/8	30x55x20	33000	1927	1913	880	946	2336	1913
Mozart/64	16x8x34	4352	995	1000	464	492	928	870

Ratio:	Comm rev/ Mem copy rev	Comm rev/ SSend	Comm rev/ IBSend	Mem copy rev/ SSend	Mem copy rev/ IBSend	SSend/ IBSend
	2,1	1,3	1,6	0,6	0,8	1,2
	2,2	1,2	1,4	0,6	0,7	1,2
	2,1	1,2	1,4	0,5	0,7	1,2
	2,1	1,4	1,4	0,7	0,7	1,0
	2,4	1,8	1,8	0,8	0,8	1,0
	2,0	1,1	1,1	0,6	0,6	1,0
	2,0	0,8	1,0	0,4	0,5	1,2
	2,0	1,1	1,1	0,5	0,6	1,1

Average ratio:	2,1	1,2	1,4	0,6	0,6	1,1
-----------------------	-----	-----	-----	-----	-----	-----

VPP5000

Model	Grid	Total pts	Common	Common revised	Memory copies	Memory copies revised	SSend-Recv	IBSend-Irecv
Initialisation-transfer-verification elapse time (seconds):								
Arpege chem	180x90x50	810000	4,27E-04	4,28E-04	1,14E-03	1,15E-03	4,50E-03	5,06E-03
Bio high	128x211x23	621184	3,07E-04	3,04E-04	8,40E-04	8,47E-04	3,50E-03	3,83E-03
Mozart	128x64x34	278528	1,44E-04	1,44E-04	4,03E-04	3,97E-04	1,75E-03	1,73E-03
Bio low	120x110x20	264000	1,41E-04	1,42E-04	3,71E-04	3,74E-04	1,50E-03	1,64E-03
Arpege land	500x500	250000	1,23E-04	1,21E-04	3,44E-04	3,42E-04	1,25E-03	1,55E-03
Bio high/8	64x53x23	78016	5,13E-05	5,09E-05	1,19E-04	1,20E-04	6,56E-04	5,05E-04
Bio low/8	30x55x20	33000	3,34E-05	3,27E-05	6,28E-05	6,32E-05	2,50E-04	2,30E-04
Mozart/64	16x8x34	4352	1,64E-05	1,69E-05	2,13E-05	2,13E-05	3,12E-04	5,30E-05
Initialisation-transfer-verification rate (Mbytes/sec):								
Arpege chem	180x90x50	810000	15176	15140	5684	5635	1440	1281
Bio high	128x211x23	621184	16187	16347	5916	5867	1420	1298
Mozart	128x64x34	278528	15474	15474	5529	5613	1273	1288
Bio low	120x110x20	264000	14979	14873	5693	5647	1408	1288
Arpege land	500x500	250000	16260	16529	5814	5848	1600	1290
Bio high/8	64x53x23	78016	12166	12262	5245	5201	951	1236
Bio low/8	30x55x20	33000	7904	8073	4204	4177	1056	1148
Mozart/64	16x8x34	4352	2123	2060	1635	1635	112	657
Ratio:			Comm rev/ Mem copy rev	Comm rev/ SSend	Comm rev/ IBSend	Mem copy rev/ SSend	Mem copy rev/ IBSend	SSend/ IBSend
			2,7	10,5	11,8	3,9	4,4	1,1
			2,8	11,5	12,6	4,1	4,5	1,1
			2,8	12,2	12,0	4,4	4,4	1,0
			2,6	10,6	11,5	4,0	4,4	1,1
			2,8	10,3	12,8	3,7	4,5	1,2
			2,4	12,9	9,9	5,5	4,2	0,8
			1,9	7,6	7,0	4,0	3,6	0,9
			1,3	18,5	3,1	14,6	2,5	0,2
Average ratio:			2,4	11,8	10,1	5,5	4,1	0,9