Optimization of the SOR solver for parallel run

Rachid Benshila (LOCEAN-IPSL)
ESOPA team
September 2005

Technical report
Version 1

I. Introduction

This report presents a development to be use for parallel run on a wide range of processors and aims to increase NEMO performance. Running on several processors is necessary to resolve features on a large area with small space scale, but the speed up induced by the number of CPU is moderated by the number of communications and the potential slowness of the network. One tries with this improvement to reduce the number of communications in one of the more expensive routine. Since the performance is a balance between the processor power and the speed of the network, the gain obtained will be strongly computer dependant, and will change on each computer among the number of processors used and the way to divide the global area.

II. Method

When looking at a NEMO flowtrace file generated after a parallel run, the communication routines appear logically to be the more costly (in term of CPU time) with the use of mppsend and mppreceive throught the calls to lib_mpp, particularly with the routine lbclnk which deals with standard boundary treatement.A high number of these calls are required by the barotropic solver to update the boundary conditions at each iteration . In the case of ORCA2_LIM configuration with the SOR solver, one reaches the convergence after 300 iterations, which means 600 calls to lbclnk by time step.

This work deals only with the SOR solver (i.e the solsor module), and aims to reduce the number of its calls to lbclnk and to improve the parallel efficiency of the code.

Solver quick description :

One solves the elliptic equation for the barotropic stream function (free surface case) or the transport divergence (rigid lid case) by using a red-black successive over-relaxation method.

The solution gcx is computed in iterating (on even and odd points) what follows :

```
ztmp =          gcb(ji  ,jj  )  &
       &  - gcp(ji,jj,1) * gcx(ji  ,jj-1)  &
       &  - gcp(ji,jj,2) * gcx(ji-1,jj  )  &
       &  - gcp(ji,jj,3) * gcx(ji+1,jj  )  &
       &  - gcp(ji,jj,4) * gcx(ji  ,jj+1

gcx(ji,jj) = sor * ztmp + (1-sor) * gcx(ji,jj)
```

with gcb the second member of the barotropic system and gcp the extra-diagonal elements of the barotropic matrix.
This computation is performed only on the inside area, because the solution on (i,j) depends of the neigbours (i-1,j), (i+1,j), (i,j-1) (i,j+1), therefore the boundaries have to be updated 2 times at each iteration.

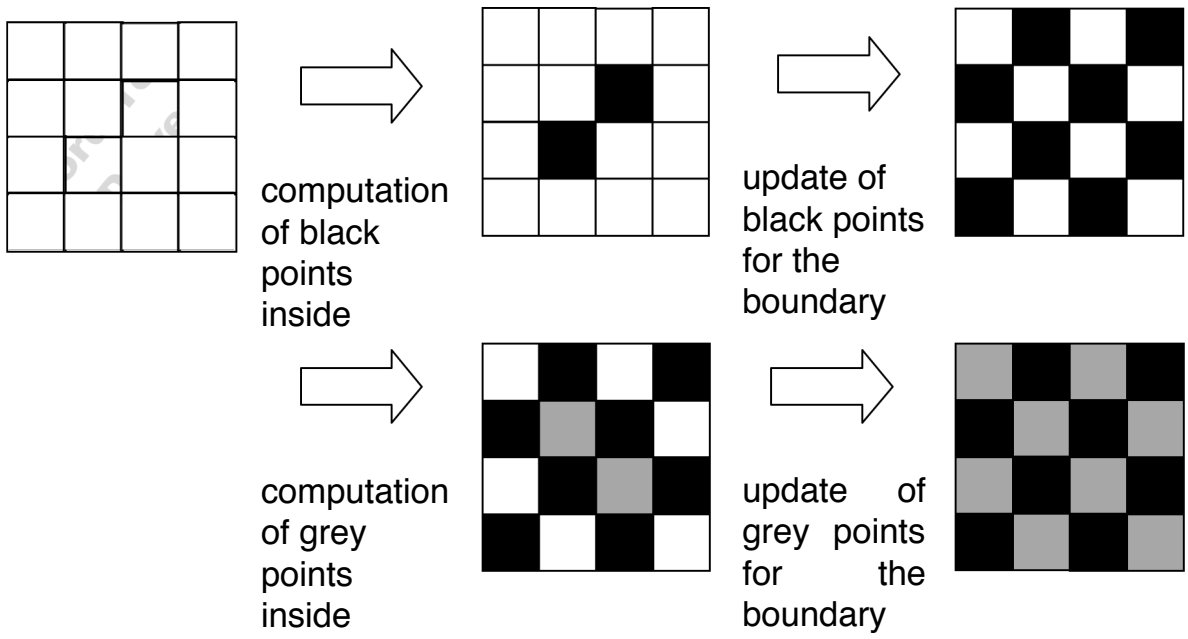

Fig II-1 : SOR solver

In mono-processor mode, boundary update are performed following the east-west periodicity and the north fold condition. This choice depends on parameter jperio.

For a multi processor run, they are updated throught communications with neighboring processors (Fig II-2) . For each time step several exchange with adjacent processors must occur. This message passing involves a processor values that have been calculated on the inside area to its neighbours, which will store these values on its boundary.

For example with 2 processors along j :

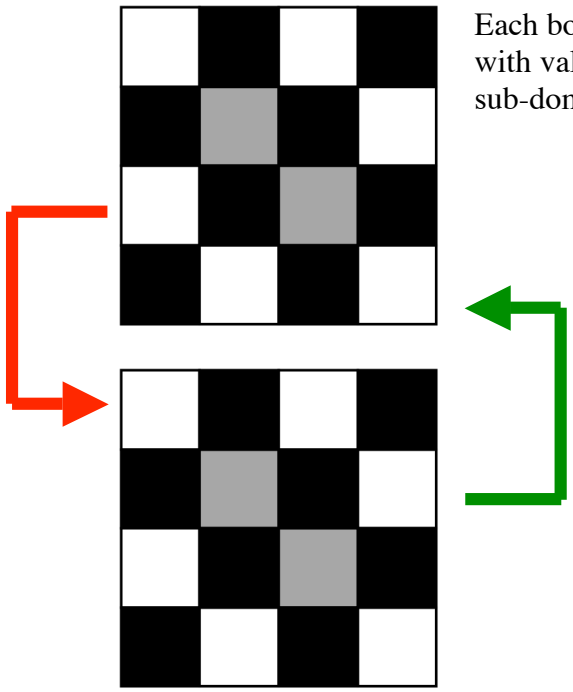

Fig II-2 : boundary update with MPI exchange

For the northern, eastern and western processors, boundary conditions are updated assuming the periodicity choice and the north fold pivot.

With several processors, the cost of these communications can be expensive and are an hindrance to run on more CPU.

Improvment :

To decrease the communication's weight, outer extra outer halo have been added along both i and j. These halos are used only in the solver and allow to decrease the frequency of update. The size of computation for the solver will enlarge for each processor, but the global cost can decrease, since it corresponds to a balance between local computation and communication.

If we consider the case of adding one extra-halo, then two halos can be send and received for each data exchange (the inner on the boundary and the outer one) . When message passing occurs, data from both halos inside is packed and sent to the adjacent processors. With this extra information, a processor can compute a new value and the next time the boundary is already up to date and no message pssing is required.

For example on a 4*4 sub-domain with one outer extra-halo :

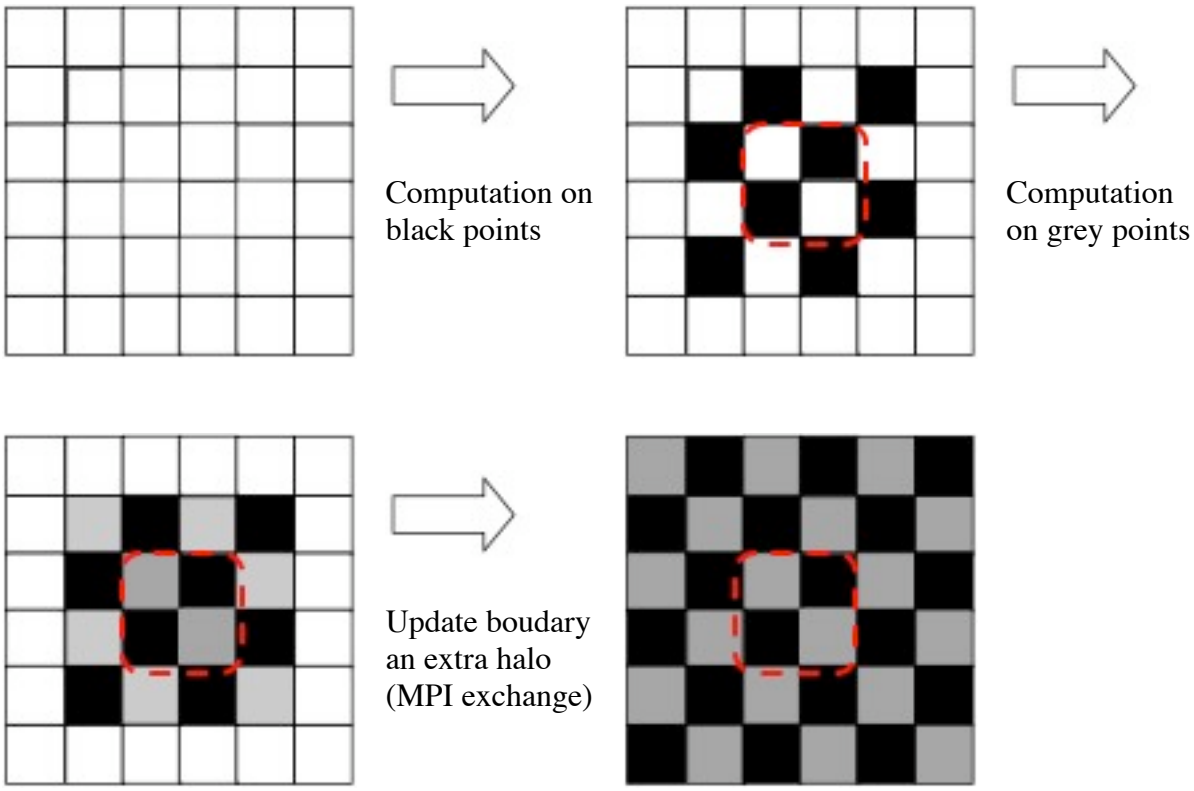

Fig II-3 : SOR solver with extra outer halo

The extra outer halo are updated classicaly with MPI communications with the neighbouring processors. For the initial state, all the matrix used for the solver are computed in the former way and the value on extra halo come from neighbours processors. A new routine mpp_lnk_2d_e (with interface lbc_lnk_e) have been added in the module lib_mpp.
Adding one extra halo allows to divide the number of communications by two in solsor.
Adding three divides the initial number by four , adding 5 by 8 etc …
A saturation point is reached when the decrease of communications can no more correct the local computation size.

We have made the choice to kepp the former SOR solver and to implement this as a new solver (see part III)

North fold case :

Particular operation as been performed for northern processors with ORCA configurations, since those configurations include a T pivot (ORCA2, ORCA025) or a F pivot (ORCA05).

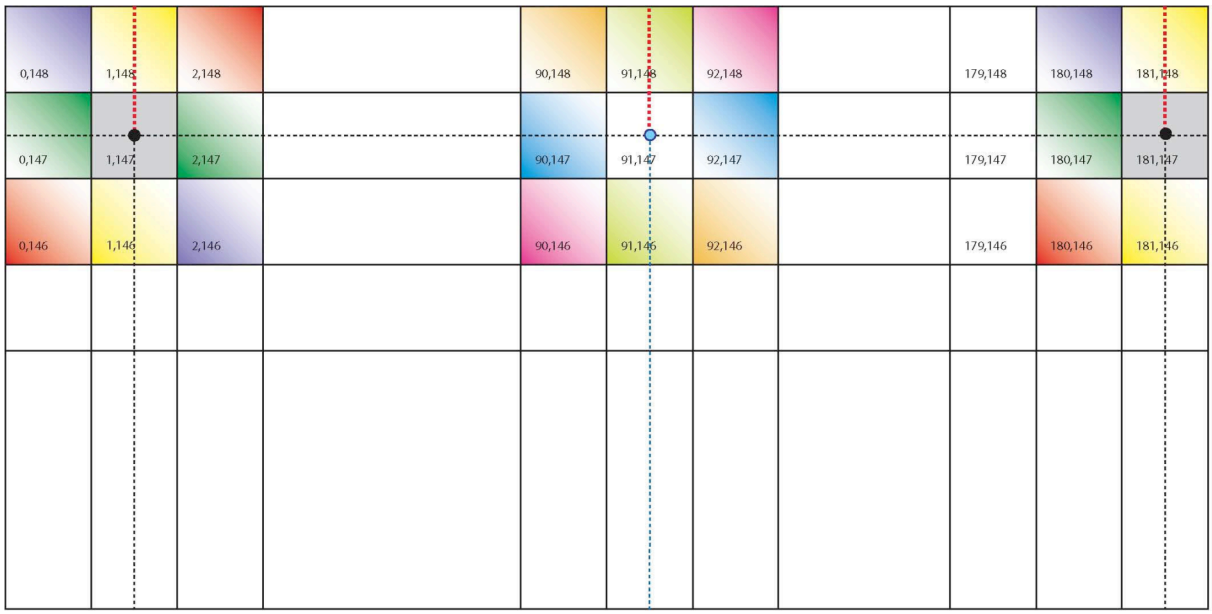

Fig II-4 : orca 2 grid with T pivot

A rotation is performed to update the northern line. The northern extra halo is updated in the same way though all the neighbours orientation is reversed. For the northern lines (last one and extra halo), the computation of the solution :

```
ztmp =          gcb(ji  ,jj  )  &
       &  - gcp(ji,jj,1) * gcx(ji  ,jj-1)  &
       &  - gcp(ji,jj,2) * gcx(ji-1,jj  )  &
       &  - gcp(ji,jj,3) * gcx(ji+1,jj  )  &
       &  - gcp(ji,jj,4) * gcx(ji  ,jj+1)
```

must be :

```
ztmp =          gcb(ji  ,jj  )  &
       &  - gcp(ji,jj,1) * gcx(ji  ,jj+1)  &
       &  - gcp(ji,jj,2) * gcx(ji+1,jj  )  &
       &  - gcp(ji,jj,3) * gcx(ji-1,jj  )  &
       &  - gcp(ji,jj,4) * gcx(ji  ,jj-1)
```

Even if gcp boundaries have been initialy updated, one have to perform a circular permutation just after the update. This has been done in a new routine sol_exd in module solmat, instead of distinguish different cases in the solver (almost for readibility reasons)

Reproductibility issue :

With NEMO and the SOR solver, a classical way to know if the reproductibility mono/multi is ensured is to compare the solver.stat file, which include the residu and the norme of the solution. The new solver with the extra outer halos set to zero corresponds exactly to the former one and gives exactly the same results. This reproductibility is conserved with a closed configuration and any value of extra halo.
When dealing with an ORCA configuration, the rotation induced by the north fold pivot change the order of computation in the solver for the northern lines. We perform :
d+c+b+a
Instead of
a+b+c+d

The reproductibility is losen but can be check by changing the solver for this specific case.

III. How to use

These changes have been done in a new module called solsor_e.F90, very close from the initial SOR solver. It corresponds to the namelist parameter :

nsolv =4

The size of outer extra halo must be changed in par_oce.F90, for example ;

jpr2di =2
jpr2dj =2

for two halos in each direction. We must have :
jpr2di *2 =< nlci
jpr2dj *2 =< nlcj

Take care that with a cyclic east-west condition, jpr2di and jpr2dj must be equal, and so with several processors along i.

These parameters must be changed only when using nsolv=4 with key_mpp_mpi activated.

IV. Perspectives

This development introduced as a new solver could and should be merged with the former SOR solver, since both are equivalent without extra-halo.

Such a technical feature could be test on the preconditionned conjugate gradient solver too, but the gain obtianed does not appear so clearly.

V. Modified routines

Here are described the modifications done on the reference version of NEMO (from CVS tag nemo_v1_05) :

par_oce.F90 : add parameters jpr2di and jpr2dj corresponding to the size of extra outer halo

lib_mpp.F90 : two routines added
mpp_lnk_2d_e : general boundary conditions with outer halo
mpp_north_2d_e :particular case of the north fold with more than one processor along i direction

lbclnk.F90 : interface for mpp_lnk_2d_e
restart.F90 and **restart_dimg.h90** : take into account new dimension of the solution

solver.F90 : add the new solver (nsolv=4) when reading the namelist

sol_oce.F90 : add jpr2di and jpr2dj to the dimension of the matrix

solmat.F90 : update boudary condition for the matrix in case of nsolv=4,
add a new routine to deal with particular case at north fold

solsor_e.F90 : new solver corresponding to the classical SOR with outer extra halo

dynspg_fsc.F90, dynspg_fsc_atsk.F90, **dynspg_rl.F90** :
add call to the new solver
update gcb boundary condition
norme computation only inside the domain