

On a first use of CWIPI at CERFACS

By F. Duchaine†, T. Morel† and A. Piacentini†

Motivations and objectives

This report deals with the evaluation of CWIPI with CERFACS codes for massively parallel applications. To achieve this, the aerothermal application developed with O-PALM (Duchaine *et al.* 2009) has been realized with CWIPI. The solvers used in this study are AVBP_V6.1 for the computational fluid dynamics and AVTP_V2.0 for the conduction into solids.

In this report, we will first introduce the methodology retained to integrate CWIPI paradigms in the codes. Some tests are then presented to assess the coupling. Finally, performance results are exposed. For all these points, results from the application with O-PALM and from the application with CWIPI are compared.

In this report, (\mathcal{C}_p) designates the coupled configuration realized with O-PALM and presented in (Duchaine *et al.* 2009) and (\mathcal{C}_c) the coupled model with CWIPI introduced here.

Aerothermal application AVBP - AVTP with CWIPI

The integration of CWIPI in the AVBP and AVTP solvers have been made in order to facilitate the source management and maintenance. In this section, the *CWIPIsation* of the AVTP solver is presented. Indeed, as the structure of both codes is similar, their *CWIPIsation* are comparable.

A new folder *CWIPI/* is created in sources of AVTP. This folder contains a directory *DUPL/* in which some sources of AVTP are duplicated with some modifications. Then, the *CWIPI/* directory contains a set of 11 sources files useful for the coupling interface with CWIPI. A new *makefile* called *makefile_cwipi* for the code AVTP interfaced with CWIPI is thus created from the existing one. This *makefile* allows the creation of an executable linked with the CWIPI library.

The first step of the *CWIPIsation* consists in addressing parallelism issues. In AVTP, the parallel instructions are based on the *ipm* macro. As a result, some modifications in the *ipm.inc* file allow to directly prepare the code for the interactions with CWIPI. Note that the *ipm.inc* file is duplicated in *ipm_cwipi.inc* for CWIPI with the following changes. The first modification concerns the communicator `MPLCOMM_WORLD`. As this communicator becomes the global one used by the whole coupled application (AVBP & AVTP) and not only by the code itself, a new local communicator (just for one solver) have to be attributed to the coupled solvers. The fonction *cwipi_init_f* allows this definition:

```
1  define (IPM_INIT ,
2  #
3  c
4  c    - ipm - init -
5  c
6  c    call MPI_INIT ( ipm_info )
7  c    call cwipi_init_f (MPLCOMM_WORLD,"avtp" ,
8  c    +                localcomm)
9  c    ipm_comm = localcomm
10 c    ipm_comt = localcomm
11 c    ipm_comm = MPLCOMM_WORLD
12 c    ipm_comt = MPLCOMM_WORLD
13 .
14 .
```

† CERFACS, 42 avenue G. Coriolis, 31 057 Toulouse Cedex 01, France

```

15 c      - ipm -
16 )
17 )

```

The name of the coupled code (here *avtp* on line 7) is given to CWIPI. This name is useful for the connection with an other solver, as will be presented later. From lines 9 to 12, we see that the `MPLCOMM_WORLD` communicator has been replaced by the *localcomm* communicator given by CWIPI. In order to have an access to the communicator *localcomm* everywhere in the AVTP solver, its declaration is embedded in a module *mod_avtp_cwipi*. The second modification of the *ipm* macro concerns the finalization of the coupled application. In order to properly finalize the coupling procedure, the directive *cwipi_finalize* is called by the code.

```

1  define (IPM.END,
2  #
3  c
4  c      - ipm - end -
5  c
6  c      call cwipi_finalize_f ()
7  c  --- > CWIPI
8  c      call MPINF finalize ( ipm_info )
9  c      if ( ipm_info.ne.MPLSUCCESS ) then
10 c          call MPLABORT ( ipm_comm [,] ipm_info [,] ipm_work )
11 c      endif
12 c  --- > CWIPI
13 c
14 c      - ipm -
15 )

```

It is important to note that for the moment, CWIPI is in charge to finalize the MPI process. As a consequence, the `MPI_FINALIZE` directive of the standalone code on line 8 is commented. Due to the modifications of the *ipm* macro, the source files *kpl_begin.F*, *kpl_end.F* and *master_control.F* have to be duplicated in order to allow a compilation with and without CWIPI. The duplicate source of *kpl_begin.F* integrates a use of the CWIPI library and to the module *mod_avtp_cwipi*. The duplicate sources of *kpl_end.F* and *master_control.F* integrate a use of the CWIPI library. Moreover, the initial source *kpl_update.F* uses MPI directive on the global communicator `MPLCOMM_WORLD` which has to be replaced by the local communicator of CWIPI *localcomm*. This concludes the modifications and duplications concerning the parallelism of the solver.

The following modifications are directly linked to the surface coupling with an other code. The main program of AVTP is modified in the following manner:

```

1  *
2  *      Copyright (c) CERFACS (all rights reserved)
3  *
4  *      =====
5  *      program avtp
6  *
7  *      IPM declarations:
8  *
9  *      IPMDECLAR
10 *
11 *      Begin MPL session:
12 *
13 *      ! -> Interface couplage
14 *      call interf_chdir
15 *
16 *      call kpl_begin( ierror )
17 *
18 *      ! -> Interface couplage
19 *      call interf_init
20 *
21 *      call master
22 *
23 *      call slave
24 *
25 *      End MPL session:
26 *
27 *      ! -> Interface couplage
28 *      call interf_end

```

```

28   call kpl_end( ierror )
29
30   end

```

The subroutines starting with *interf_* are added to the standalone code. In order to facilitate the source management, these instructions are always called by the sources but their refer to empty routine when CWIPI is not used and to coupling instructions when CWIPI is used. The routine *interf_chdir* (line 13) allows to specify a running directory for the AVTP code.

```

1   call chdir("./AVTP01")

```

Then, the *interf_init* routine initialize the coupling with an other code, here AVBP (line 25):

```

1   open(iiunit , file=" ../coupling.choices" , status="old" )
2   read(iiunit ,*) dim_geom
3   read(iiunit ,*) tol_geom
4   read(iiunit ,*) out_freq
5   read(iiunit ,*)
6   read(iiunit ,*) nit_ncpl
7   read(iiunit ,*) n_cpl
8   close(iiunit)
9   !*
10  nit_cpl   = nit_ncpl
11  niter_cpl = nit_cpl * n_cpl
12  i_cpl     = 0
13  !*
14  open(iiunit , file='avtp_lite_out' , status='unknown' )
15  call cwipi_set_output_listing_f(iiunit)
16  write(iiunit ,*)
17  write(iiunit ,*) "dump_apres_initialisation"
18  write(iiunit ,*) "_____ "
19  write(iiunit ,*)
20
21  call cwipi_dump_appli_properties_f
22
23  call cwipi_create_coupling_f(" Test_flo_0" ,
24  +   cwipi_cpl_parallel_with_part ,
25  +   "avbp" ,
26  +   dim_geom ,
27  +   tol_geom ,
28  +   cwipi_static_mesh ,
29  +   cwipi_solver_cell_vertex ,
30  +   out_freq ,
31  +   "Ensignit_Gold" ,
32  +   "text" )
33
34  innodeb_cpl   = 0
35  infaceb_cpl   = 0
36  nnodeb_cpl    = 0
37  nfaceb_cpl    = 0
38  nfaceb_nv_cpl = 0
39  compt_cell    = 0

```

From the primitive *cwipi_create_coupling_f*, we see that the creation of the coupling (line 23) implies to know the name of the code to couple with. This aspect is at the opposite of the O-PALM philosophy. Indeed, a *PALMerisation* of a code is done without any reference to another code: the codes give/receive data to/from the O-PALM application and not to/from a given solver. Nevertheless, a parametrization of this input for CWIPI via a variable given to the code is possible. For example in the case presented here, the solver read some useful parameter for the simulation on lines 2 to 7. The coupling is declared to CWIPI by the instruction *cwipi_create_coupling_f* on line 23. Lines 34 to 39 contain some initializations for the coupling.

The routine *interf_end* allows to delete the coupling (line 4) and to carry out some des-allocation of tabulars used for data exchanges between the codes. Note that this tables are declared in the module *mod_avtp_cwipi* to guaranty an accessibility everywhere in the code.

```

1   use cwipi
2   use mod_avtp_cwipi
3   !*
4   call cwipi_delete_coupling_f(" Test_flo_0" )
5   !*
6   deallocate ( xyz )

```

```

7      deallocate ( tab_send )
8      deallocate ( tab_rec  )
9      deallocate ( elems   )
10     deallocate ( connind  )
11     deallocate ( glo2cpl  )
12     deallocate ( cpl2glo  )
13     deallocate ( cpl2bnd  )
14     deallocate ( T_send   )
15     deallocate ( T_rec    )
16     deallocate ( Flux_rec  )

```

The geometric entities requested by CWIPI to construct the correspondence between the codes are constructed in the routine *metric* of AVTP. In the conduction solver, adding *_CPL* to the type of a boundary condition imply a coupling (need of information as well as availability of data) for this boundary. The first loop (line 2 to 18) allows to dimension (with the instruction *interf_compte_CPL*) the tables for the coupling. The tables are then allocated with the instruction *interf_alloc_CPL* (line 20). Finally, the routine *interf_generate_CPL* (line 36) creates the coupling mesh which is shared with CWIPI in *interf_send_mesh_CPL* (line 42).

```

1  !-----Interface couplage
2      do np=1,npbound
3          nbeg = ibound1(np)
4          nlen = ibound2(np)
5          key = patch_key(np)
6          if (ndim.eq.2) then
7              k = 2
8          else
9              k = 1
10         endif
11         ip_ibfano = ifbound(1,k,np)
12         nfbeg = nfbgbegin(2,1,k,np)
13         nflen = nbface(k,np)
14         call dlongc ( 80,key,nc1,nc2 )
15         if ( key(nc2-3:nc2) .eq. '_CPL') then
16             call interf_compte_CPL(nlen, nflen, nvbf(k))
17         endif
18     enddo
19 !
20     call interf_alloc_CPL(ndim, nnode)
21 !
22     do np=1,npbound
23         nbeg = ibound1(np)
24         nlen = ibound2(np)
25         key = patch_key(np)
26         call dlongc ( 80,key,nc1,nc2 )
27         if (ndim.eq.2) then
28             k = 2
29         else
30             k = 1
31         endif
32         ip_ibfano = ifbound(1,k,np)
33         nfbeg = nfbgbegin(2,1,k,np)
34         nflen = nbface(k,np)
35         if ( key(nc2-3:nc2) .eq. '_CPL') then
36             call interf_generate_CPL(nvbf(k),ndim, nflen, nnode, nlen, nbeg,
37 &                                     ibfano, ibound(nbeg), iibound, x)
38         endif
39     enddo
40 !
41     call interf_send_mesh_CPL
42 !
43 !-----Interface couplage
44

```

The counting operations in *interf_compte_CPL* read:

```

1      nnodeb_cpl = nnodeb_cpl + nlen
2      nfaceb_cpl = nfaceb_cpl + nflen
3      nfaceb_nv_cpl = nfaceb_nv_cpl + nflen * nv

```

where *nnodeb_cpl*, *nfaceb_cpl*, *nfaceb_nv_cpl* stand for the number of coupling node, the number of coupling cells and the total number of vertex, respectively.

In *interf_alloc_CPL*, the allocations are made in order to avoid allocation to zero dimension when

a processor does not contain any coupling node. Moreover, one must take care that with CWIPI, the dimension of the node coordinate table is always three-dimensional (*ndim* on line 1):

```

1      cndim      = 3 ! ndim
2      alloc_xyz  = max(1, nnodeb_cpl*cndim)
3      alloc_tab  = max(1, 3*nnodeb_cpl)
4      alloc_elems = max(1, nfaceb_nv_cpl)
5      alloc_conn = max(1, nfaceb_cpl+1)
6      alloc_cpl2glo = max(1, nnodeb_cpl)
7      alloc_cpl2bnd = max(1, nnodeb_cpl)
8      alloc_T    = max(1, nnodeb_cpl)
9      alloc_Flux = max(1, ndim*nnodeb_cpl)
10
11     !
12     allocate ( xyz(alloc_xyz) )
13     allocate ( tab_send(alloc_tab) )
14     allocate ( tab_rec(alloc_tab) )
15     allocate ( elems(alloc_elems) )
16     allocate ( connind(alloc_conn) )
17     allocate ( glo2cpl(nnode) )
18     allocate ( cpl2glo(alloc_cpl2glo) )
19     allocate ( cpl2bnd(alloc_cpl2bnd) )
20     allocate ( T_send(alloc_T) )
21     allocate ( T_rec(alloc_T) )
22     allocate ( Flux_rec(alloc_Flux) )
23
24     !
25     xyz      = 0.0d0
26     tab_send = 0.0d0
27     tab_rec  = 0.0d0
28     connind  = 0
29     elems    = 0
30     glo2cpl  = -1

```

In *interf_generate_CPL*, the generation of the coupling mesh tables consists of the construction of *connind* (the number of vertex per cell, line12), *xyz* (the node coordinates, line 23) and *elems* (the cell connectivity, line 34):

```

1      do inode=1,nnode
2          iibound(inode) = 0
3      end do
4      do inode=1,nlen
5          n1 = iibound(inode)
6          iibound(n1) = inode
7      end do
8
9      !*
10     !*****
11     !*
12     do icell = 1, nflen
13         connind(infaceb_cpl+icell+1) = connind(infaceb_cpl+icell) + nv
14     enddo
15     infaceb_cpl = infaceb_cpl + nflen
16
17     !*
18     !*****
19     !*
20     do inode = 1, nlen
21         glo2cpl(iibound(inode)) = innodeb_cpl + inode
22         cpl2glo(innodeb_cpl+inode) = iibound(inode)
23         cpl2bnd(innodeb_cpl+inode) = nbeg + inode - 1
24         do idim = 1, ndim
25             xyz((innodeb_cpl+inode-1)*cndim+idim) = x(idim, iibound(inode))
26         enddo
27     enddo
28     innodeb_cpl = innodeb_cpl + nlen
29
30     !*
31     !*****
32     !*
33     do icell = 1, nflen
34         do nn = 1, nv
35             n1 = ibfano(nn, icell)
36             inode = iibound(n1)
37             elems(compt_cell+nn) = glo2cpl(iibound(inode))
38         enddo
39         compt_cell = compt_cell + nv
40     enddo

```

The tables *glo2cpl*, *cpl2glo* and *cpl2bnd* are local indirection tables that allow to easily refer to a

coupling node from a global node on the partition, to a global node from a coupling node and to a boundary node from a coupling, respectively.

The last routine called by *metric* is *interf_send_mesh_CPL*. It allows to share the mesh defined by *nnodeb_cpl*, *nfaceb_cpl*, *connind*, *xyz* and *elems* with CWIPI.

```

1      call cwipi_define_mesh_f(" Test_flo_0" ,
2      +                               nnodeb_cpl ,
3      +                               nfaceb_cpl ,
4      +                               xyz ,
5      +                               connind ,
6      +                               elems )

```

Finally, the physical fields are exchanged during the temporal loop in *slave_temporal*.

```

1      ! -> Interface couplage
2      call interf_niter
3      !
4      do nit=1,niter
5      !
6      .
7      .
8      .
9      !
10     ! -> Interface couplage
11     call interf_exchange
12     !
13     end do

```

The number of iterations of the temporal loop *niter* is first set to the one obtained in the pretreatment (*interf_init*) via the routine *interf_niter* (line 2 of *slave_temporal*):

```

1      niter = niter_cpl

```

The routine *interf_exchange* called on line 11 of *slave_temporal* generates a *rendez-vous* between the codes if the temporal iteration correspond to a meeting point:

```

1      if (nit.eq.nit_cpl) then
2      call interf_send_rec(ndim,neq,neqbnd,nnode,nblength,w,
3      +                   snbound,wbnd_ref)
4      nit_cpl = nit_cpl + nit_ncpl
5      endif

```

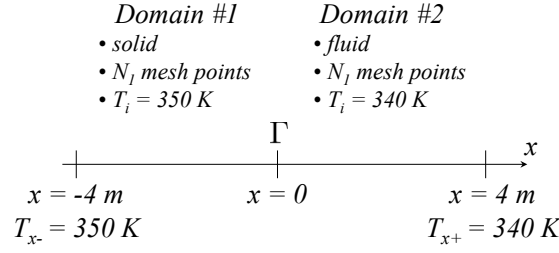
The coupling point is placed in this example at the end of the iteration.

The exchange of the physical quantities correspond to a sending of the temperature and a reception of both a temperature and a thermal flux. As a consequence, temperatures are exchanged with the side-to-side procedure *cwipi_exchange_f* (line 16) while the flux is receive with a *cwipi_receive_f* (line 29):

```

1      T_rec = 0.0d0
2      T_send = 0.0d0
3      Flux_rec = 0.0d0
4      i_cpl = i_cpl + 1
5      !*
6      do inode = 1, nnodeb_cpl
7      T_send(inode) = w(1,cpl2glo(inode)) ! Temperature
8      enddo
9      !*
10     !*****
11     !*
12     dimension = 1
13     numeroex = i_cpl
14     time = real(i_cpl)
15
16     call cwipi_exchange_f (" Test_flo_0" ,
17     +                       "exchangeT" ,
18     +                       dimension ,
19     +                       numeroex ,
20     +                       time ,
21     +                       "Temperature" ,
22     +                       T_send ,
23     +                       "Temperature" ,
24     +                       T_rec ,
25     +                       nNotLocatedPoints ,

```

FIGURE 1. Description of the one dimensional test case used to verify the coupling models (\mathcal{C}_c).

```

26 +                               il_err )
27 !*
28   dimension = ndim
29   call cwipi_receive_f ( "Test_flo_0" ,
30 +                       "exchangeF" ,
31 +                       dimension ,
32 +                       numeroex ,
33 +                       time ,
34 +                       "Flux" ,
35 +                       Flux_rec ,
36 +                       nNotLocatedPoints ,
37 +                       il_err )
38 !*
39   if ( nNotLocatedPoints.ne.0 ) then
40     WRITE(6,*) " _nNotLocatedPoints_=" , nNotLocatedPoints ,
41 +           "sur_AVTP" , nnodeb_cpl
42   endif
43 !*
44 !*****
45 !*
46   do inode = 1, nnodeb_cpl
47     bndflux = 0.0d0
48     do idim = 1, ndim
49       bndflux = bndflux + Flux_rec(ndim*(inode-1)+idim)*
50 +                               sbound(idim, cpl2bnd(inode))
51     enddo
52     wband_ref(1, cpl2bnd(inode)) = T_rec(inode)
53     wband_ref(2, cpl2bnd(inode)) = bndflux
54     wband_ref(3, cpl2bnd(inode)) = T_rec(inode)
55   enddo

```

Lines 39 to 42 allow to inform the user of none located points.

It is important to point out that the aerothermal application developed with CWIPI contains only the codes to couple, AVBP and AVTP, while the O-PALM application consists of:

- the AVBP unit,
- the AVTP unit,
- an interface unit that allows to connect the fluid and solid meshes and to interpolate physical quantities,
- the O-PALM driver.

As the interface and the O-PALM driver are not CPU consuming compared to the solvers, these two executables are launched on the same processor. This processor is not counted for the speed up determinations.

Verification of the coupling

In order to verify the coding of the coupling model (\mathcal{C}_c), a purely diffusive one dimensional test case is used. It consist in two domains joined by an interface Γ (Fig. 1) and sharing the same almost constant thermal properties (Tab. 1). Whereas it is easy to set constant properties in the solid domain, fluid properties like the density ρ depend on the temperature. The domain corresponding to $x \in [-4, 0]$ is treated with the solid solver AVTP and the other part ($x \in [0, 4]$) with the fluid solver AVBP.

Heat capacity	Density	Thermal conductivity	Thermal Diffusivity	Thermal effusivity
$\text{J.kg}^{-1}.\text{K}^{-1}$	kg.m^{-3}	$\text{W.m}^{-1}.\text{K}^{-1}$	$\text{m}^2.\text{s}^{-1}$	$\text{J.K}^{-1}.\text{m}^{-2}.\text{s}^{-1/2}$
C_p	ρ	λ	$D = \frac{\lambda}{\rho C_p}$	$E = \sqrt{\lambda \rho C_p}$
1041.29	1.0038	0.0264	$2.525 \cdot 10^{-5}$	5.2532

TABLE 1. Thermal properties used for the one dimensional problem.

Code	τ_d (s)	Nit^{τ_d}	α	Nit^{cpl}	Nit^{tot}
AVBP	0.6335	14 869 119	0.001	15000	30 000 000
AVTP	0.6335	30 632	0.001	31	62 000

TABLE 2. Description of the coupling frequencies used for the one dimensional test case.

The aim of the present problem is to find the converged temperature in the coupled system when imposing constant temperatures at the ends of the domains: $T_{x-} = 350\text{K}$ for the solid and $T_{x+} = 340\text{K}$ for the fluid. In the case reported here, the initial solution is $T_i = 350\text{K}$ in the solid and $T_i = 340\text{K}$ in the fluid. The CFD and structure models are 2D. Both the fluid and the solid domains contains 1 800 triangular cells (quadrangles split into triangles) and 961 (31×31) nodes. The time steps in the fluid and solid solvers are about $4.26 \cdot 10^{-8}\text{s}$ and $2.07 \cdot 10^{-5}\text{s}$, respectively.

The coupling methodology proposed by (Duchaine *et al.* 2009) to reach a steady thermal state is used. The characteristic diffusion time in both domains are:

$$\tau_d = \frac{L^2}{D} = 0.6335 \text{ s} \quad (0.1)$$

where $L = 4 \text{ m}$ is the size of a domain (Fig. 1) and D is the thermal diffusivity. Table 2 gives the exchanges frequencies of the coupling. Nit^{τ_d} stands for the number of iterations required to simulate the time τ_d . Nit^{cpl} is an integer that approximate the number of iterations required to simulate a portion $\alpha \dagger$ of τ_d . The quantity $\alpha \tau_d$ is the physical time between two coupling event. Finally, Nit^{tot} is the total number of iteration of the simulation.

Figure 2 shows the comparisons of the O-PALM application (\mathcal{C}_p) and CWIPI model (\mathcal{C}_c). For both applications, the convergence to the steady solution is clearly evidenced on Fig. 2 a) and c). Nevertheless, a slight difference exist in the convergence historic of the the two methods. This is simply due to the first exchange of the coupling that is not treated in the same manner for the two applications. What is more important and allows to verify the coupling is that the final converged temperature profiles in the fluid and solid domains are the same for both models (\mathcal{C}_p) and \mathcal{C}_c) (Fig. 2 c).

Performance results

The configuration retained for this first set of performance tests is the NASA-C3X blade (Hylton *et al.* 1983). The aim of the computations showed in this section is to converge toward the thermal steady state of the configuration. Thus, the coupling methodology proposed by (Duchaine *et al.* 2009) is used. The CFD and structure models are 2D. The fluid domain contains 468 780 triangular

† For more information on the α -concept, read (Duchaine *et al.* 2009).

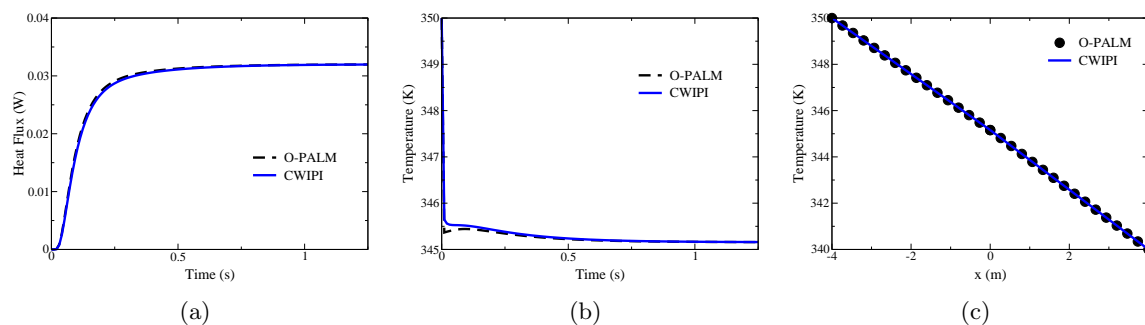


FIGURE 2. Comparison of the coupled models (\mathcal{C}_p) and (\mathcal{C}_c) on the one dimensional test. Convergence evolution of flux (1) and temperature (2) at the interface Γ , converged temperature profile (c).

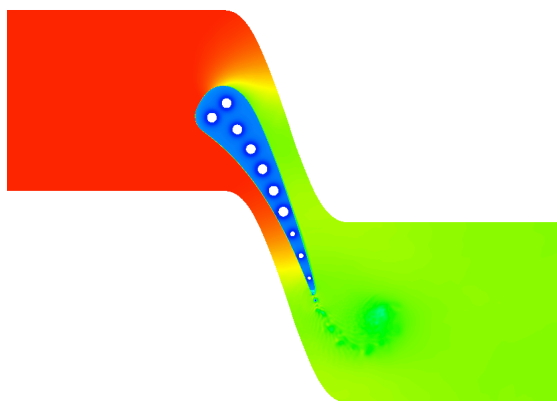


FIGURE 3. NASA-C3X blade.

cells for 237 576 nodes, and the solid model 161 760 cells for 84 468 nodes. The time steps in the fluid and solid solvers are about $1.15 \cdot 10^{-8}$ s and $1.53 \cdot 10^{-5}$ s, respectively. Both domains contain about 6 000 boundary nodes that are shared during the coupling steps.

The performance of three methods is assessed here:

- a basic chained application (\mathcal{C}_s) in which the codes are executed sequentially in a loop,
- the coupled configuration (\mathcal{C}_p) realized with O-PALM and presented in (Duchaine *et al.* 2009),
- the coupled model (\mathcal{C}_c) with CWIPI presented in this paper.

Table 3 presents the 5 tests used to measure the performance of the (\mathcal{C}_s), (\mathcal{C}_p) and (\mathcal{C}_c) schemes when increasing the number of synchronization points between the codes. From case 1 to 5, the number of coupling exchanges (N_{cpl}) increases from 10 to 1000, while maintaining the total number of iterations in the codes (Nit_{avbp}^{tot} for AVBP and Nit_{avtp}^{tot} for AVTP). Thus, the number of iterations between two coupling *rendez-vous* (Nit_{avbp}^{cpl} for AVBP and Nit_{avtp}^{cpl} for AVTP) decreases from case 1 to case 5.

Table 4 gives the number of processors used with the coupling methodologies (\mathcal{C}_p) and (\mathcal{C}_c) to insure an adequate load balancing between AVBP and AVTP. Note that for the computations with the sequential strategy (\mathcal{C}_s), both solvers used the total number of processors. The architecture used for the computations presented in this report is the CERFACS cluster Octopus (IBM IDATAPLEX, a cluster with 82 quad core bi-processors Intel Nehalem 2.67 GHz nodes, 24 GB memory, 85 peak GFlops per node).

Figure 4 shows a typical time historic of the total heat flux on the blade wall obtained with the 5 cases of Tab. 3. The first interesting observation is that all the computations converge

Nit_{avbp}^{tot}	13000				
Nit_{avtp}^{tot}	85000				
Nit_{avbp}^{cpl}	1300	260	130	26	13
Nit_{avtp}^{cpl}	8500	1700	850	170	85
N_{cpl}	10	50	100	500	1000
Case #	Case 1	Case 2	Case 3	Case 4	Case 5

TABLE 3. Description of the coupling frequencies used for the speed up determination.

Total	8	16	24	32	40	48	56	64
AVBP	5	10	15	20	25	30	35	40
AVTP	3	6	9	12	15	18	21	24

TABLE 4. Number of processors used for the speed up determination.

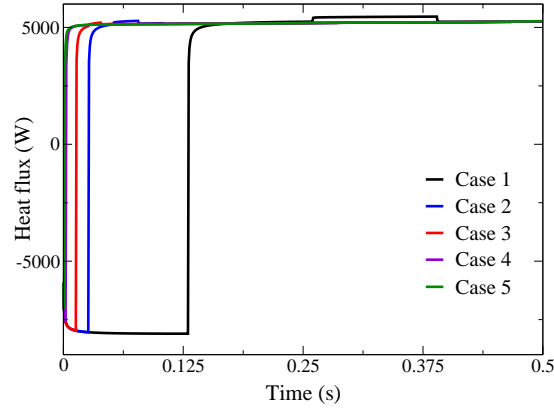


FIGURE 4. Time Historic of the heat flux on the blade wall for the 5 cases.

toward the same heat flux, illustrating the consistency of the methodology. Then, as already shown by (Duchaine *et al.* 2009) and (Wlassow *et al.* 2010), the convergence is accelerated when the codes communicate their boundary conditions with a high frequency.

Figure 5 gives the restitution times for the three coupling methodologies (\mathcal{C}_s), (\mathcal{C}_p), and (\mathcal{C}_c) as a function of the number of exchanges between the codes on 8 processors. It is important to note that the speed up informations given latter are based on these times. Not surprisingly, when considering the sequential method (\mathcal{C}_s) the restitution time evolves linearly with the number of exchange. This is directly related to the restarting of the codes (launching of the executable, reading of the mesh, initial solution, boundary condition etc, as well as post-processing steps) at each coupling step. On the other hand, the two coupling strategies (with O-PALM and CWIPI) show a good comportment when the number of exchanges is increased. Indeed, on 8 processors, the restitution time of the coupled applications is independent of the number of exchanges. This mean that in this particular case, the time requested for the exchanges between the codes remains negligible compared to the computational time of the solvers. The superiority of memory exchanges and thus the use of a

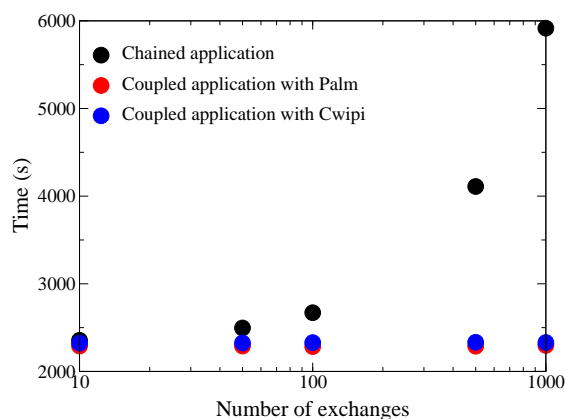


FIGURE 5. Restitution time of the simulations as a function of the number of exchange between the codes on 8 processors for the basic chained application (\mathcal{C}_s), the coupled configuration (\mathcal{C}_p), and the coupled model (\mathcal{C}_c).

Np_{tot}	Np_{avbp}	Np_{avtp}	T_{10}	Su_{10}	T_{50}	Su_{50}	T_{100}	Su_{100}	T_{500}	Su_{500}	T_{1000}	Su_{1000}
8	5	3	2355	1.00	2496	1.00	2670	1.00	4110	1.00	5916	1.00
16	10	6	1189	1.98	1347	1.85	1544	1.73	3133	1.31	5150	1.15
24	15	9	791	2.98	965	2.59	1201	2.22	2996	1.37	5404	1.09
32	20	12	608	3.88	805	3.10	1041	2.57	3021	1.36	5660	1.05
40	25	15	505	4.66	718	3.48	991	2.69	3159	1.30	6004	0.99
48	30	18	440	5.35	670	3.72	971	2.75	3328	1.23	6457	0.92
56	35	21	396	5.95	642	3.89	969	2.75	3598	1.14	6854	0.86
64	40	24	367	6.42	643	3.88	961	2.78	3782	1.09	7353	0.80

TABLE 5. Results of the basic chained application (\mathcal{C}_s). Times are in seconds.

coupler is clearly evidenced here. Compared to a standalone execution, the execution of AVBP or AVTP within the control of CWIPI takes less than 5% more time.

Tables 5 to 7 give the evolution of the restitution times when increasing the number of processors for the 5 tests cases presented in Tab. 3 for the three coupling methodologies. The speed up presented in these table are based on the results obtained for the same number of exchanges on 8 processors. Figure 6 presents a graphic representation of the speed up for the different cases. For case 1 (10 exchanges between the codes during the simulation), the speed up of the sequential coupling (\mathcal{C}_s) remains linear until 32 processors (Fig. 6-a). When the number of exchanges is increased, the linear range of the curve decreases. For a high frequency of exchange, (case 5: 1000 exchanges during the simulation), the computations take more time when increasing the number of processors than with a low number of processors.

The application (\mathcal{C}_p) developed with the O-PALM coupler (Fig. 7) shows a rather good compartment for cases 1 to 3 (Fig. 6-b). Then the increase in the exchange frequency drastically drops the performance of the strategy. Moreover, for 40 processors, we observe a non monotonic compartment of the speed up. This is due to the architecture of the cluster Octopus. When running in mpi1 mode of O-PALM, one must take care of the way to execute the application. For the case on 8 processors, a standard submission with:

```
1 mpirun -f $LOADL_HOSTFILE -np 1 ./palm_main : -np 3 ./main_avtp_mpi :
2 -np 5 ./main_avbp_mpi : -np 1 ./main_interf_aa
```

N_{ptot}	N_{pavbp}	N_{pavtp}	T_{10}	Su_{10}	T_{50}	Su_{50}	T_{100}	Su_{100}	T_{500}	Su_{500}	T_{1000}	Su_{1000}
8	5	3	2290	1.00	2290	1.00	2284	1.00	2288	1.00	2297	1.00
16	10	6	1143	2.00	1163	1.97	1143	2.00	1162	1.97	1167	1.97
24	15	9	752	3.05	763	3.00	770	2.97	820	2.79	887	2.59
32	20	12	575	3.98	575	3.98	591	3.87	671	3.41	999	2.30
40	25	15	515	4.45	816	2.81	844	2.71	911	2.51	996	2.31
48	30	18	382	6.00	409	5.60	433	5.28	706	3.24	722	3.18
56	35	21	334	6.85	360	6.37	415	5.50	600	3.82	635	3.62
64	40	24	575	3.98	575	3.98	591	3.87	671	3.41	999	2.30

TABLE 6. Results of the coupled model (\mathcal{C}_p). Times are in seconds.

leads to a restitution time of the order of 3900 seconds, 60% more than in standalone executions. The explanation is that the all the executables are launched on the first processor of virtual machine, thus slowing down the executions. It is thus necessary to specify to the batch manager the nodes on which the executables have to run. This is done with the following syntax for 8 processors (1 node on the Octopus cluster):

```

1 hostmain='echo $LOADL_PROCESSOR_LIST | awk '{print $1}''
2 hostavtp='echo $LOADL_PROCESSOR_LIST | awk '{print $1}''
3 hostavbp='echo $LOADL_PROCESSOR_LIST | awk '{print $5}''
4 mpdboot -r ssh -f $LOADL_HOSTFILE -n 1
5 mpiexec -n 1 -host $hostmain ./palm_main : -n 3 -host $hostavtp ./main_avtp_mpi :
6 -n 5 -host $hostavbp ./main_avbp_mpi : -n 1 -host $hostmain ./main_interf_aa
7 mpdallexit

```

As mentioned previously, we see from lines 2 and 6 that the O-PALM driver, the interface and the AVTP unit are launched on the same processor.

The case with 40 processors is badly shaped for this application as it requires 15 processors for AVTP (1 node + 7 processors) and 25 processors for AVBP (3 nodes + 1 processor).

```

1 hostmain = 'echo $LOADL_PROCESSOR_LIST | awk '{print $1}''
2 hostavtp1='echo $LOADL_PROCESSOR_LIST | awk '{print $1}''
3 hostavtp2='echo $LOADL_PROCESSOR_LIST | awk '{print $9}''
4 hostavbp1='echo $LOADL_PROCESSOR_LIST | awk '{print $16}''
5 hostavbp2='echo $LOADL_PROCESSOR_LIST | awk '{print $24}''
6 hostavbp3='echo $LOADL_PROCESSOR_LIST | awk '{print $32}''
7 hostavbp4='echo $LOADL_PROCESSOR_LIST | awk '{print $40}''
8 mpdboot -r ssh -f $LOADL_HOSTFILE -n 5
9 mpiexec -n 1 -host $hostmain ./palm_main : -n 8 -host $hostavtp1 ./main_avtp_mpi :
10 -n 7 -host $hostavtp2 ./main_avtp_mpi : -n 8 -host $hostavbp1 ./main_avbp_mpi :
11 -n 8 -host $hostavbp2 ./main_avbp_mpi : -n 8 -host $hostavbp3 ./main_avbp_mpi :
12 -n 1 -host $hostavbp4 ./main_avbp_mpi : -n 1 -host $hostmain ./main_interf_aa
13 mpdallexit

```

This splitting of node (7 processors for one code and 1 for the other), seem to be the cause the drastic fall of performances.

Finally, the coupling application (\mathcal{C}_c) developed with CWIPI exhibits an excellent comportment on Octopus. Indeed, the speed up remains linear until 64 processors for the 5 cases (Fig. 6-c). After the observations made on the launching of the executables with O-PALM in the mpi1 mode, it is interesting to underline that the batch mode with CWIPI directly support the following command for 8 processors:

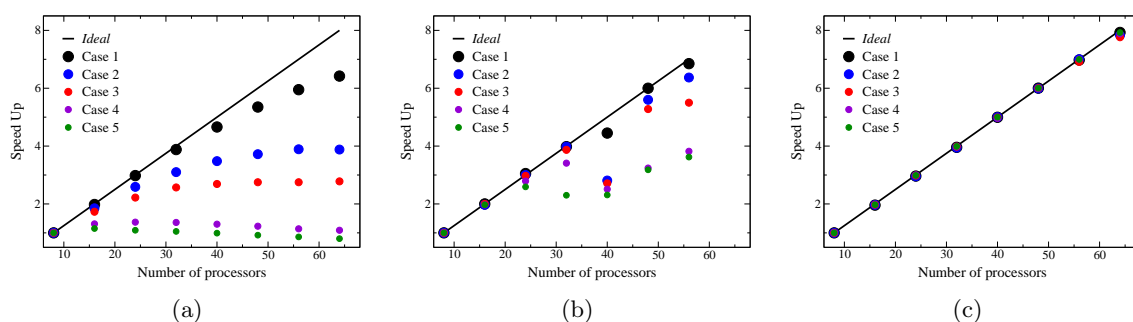
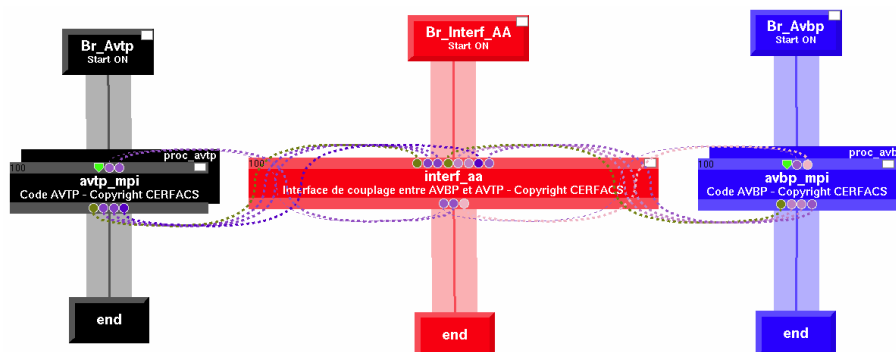
```

1 mpiexec -f $LOADL_HOSTFILE -np 5 avbp_with_cwipi.exe : -np 3 avtp_with_cwipi.exe

```

The O-PALM application (\mathcal{C}_p) presented here have been done with standard communications of the coupler. In order to offer a high level of flexibility to the users, this communication protocol imposes that all the messages transferred from one code to another are managed by the driver of

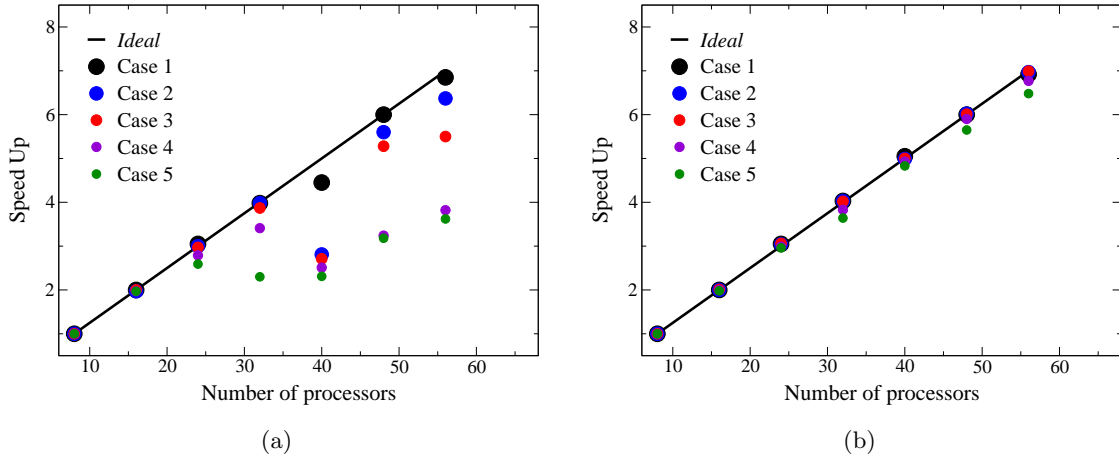
N_{ptot}	N_{pavbp}	N_{pavtp}	T_{10}	Su_{10}	T_{50}	Su_{50}	T_{100}	Su_{100}	T_{500}	Su_{500}	T_{1000}	Su_{1000}
8	5	3	2323	1.00	2325	1.00	2329	1.00	2333	1.00	2330	1.00
16	10	6	1186	1.96	1182	1.97	1180	1.97	1182	1.97	1181	1.97
24	15	9	784	2.96	787	2.95	781	2.98	779	2.99	780	2.99
32	20	12	587	3.96	589	3.95	585	3.98	586	3.98	584	3.99
40	25	15	465	5.00	465	5.00	467	4.99	470	4.96	466	5.00
48	30	18	387	6.00	388	5.99	388	6.00	389	6.00	388	6.01
56	35	21	333	6.98	332	7.00	337	6.91	333	7.01	333	7.00
64	40	24	293	7.93	297	7.83	300	7.76	296	7.88	294	7.93

 TABLE 7. Results of the coupled model (\mathcal{C}_c). Times are in seconds.

 FIGURE 6. Speed up for (a) the basic chained application (\mathcal{C}_s), (b) the coupled configuration (\mathcal{C}_p), and (c) the coupled model (\mathcal{C}_c).

 FIGURE 7. Prepaln caneva of the coupled application (\mathcal{C}_p).

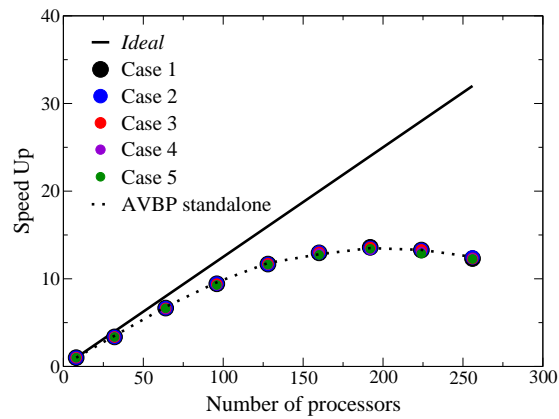
O-PALM. As a result, when the codes communicate with a high frequency and/or when they are launch on a high number of processors, the driver has a lot of work and can be saturated. The latest version of O-PALM (V4.0.0) offers the possibility to overcome this non-optimal behavior. The optimization consists in defining blocking communications between the codes that does not require the intervention of the driver. The routing of the informations is defined by the driver at the beginning of the coupled execution. During the simulation, the codes communicate directly. In the present O-PALM application (Fig. 7), this option is activated on the communications concerning the exchange of physical quantities in the temporal loop of the solvers (i.e. between AVBP and the interface as well as between AVTP and the interface). Table 8 and Fig. 8 illustrate the improvement of this optimization.

The CWIPI coupling application (\mathcal{C}_c) has been ported on the HP Proliant Corail. On Corail,

Np_{tot}	Np_{avbp}	Np_{avtp}	T_{10}	Su_{10}	T_{50}	Su_{50}	T_{100}	Su_{100}	T_{500}	Su_{500}	T_{1000}	Su_{1000}
8	5	3	2267	1.00	2263	1.00	2281	1.00	2266	1.00	2259	1.00
16	10	6	1135	2.00	1134	2.00	1134	2.01	1134	2.00	1138	1.98
24	15	9	743	3.05	745	3.04	746	3.06	757	2.99	762	2.96
32	20	12	562	4.03	562	4.03	567	4.02	592	3.83	620	3.64
40	25	15	449	5.05	454	4.99	456	5.00	460	4.93	468	4.83
48	30	18	377	6.00	376	6.02	379	6.01	384	5.90	400	5.65
56	35	21	328	6.92	325	6.97	326	6.99	335	6.77	349	6.48

TABLE 8. Results of the optimized coupled model (C_p). Times are in seconds.FIGURE 8. Speed up for the coupled configuration (C_p): (a) non optimized communication protocol, (b) optimized communications.

every node has 12 2.2 Ghz AMD core (with 4 flop per cycle per node, i.e. 211 peak GFlops per node, 32 GO memory). Figure 9 shows the performance of CWIPI on this architecture until 256 cores. It is important to note that for these computations, the AVBP setup has not been optimized to reach a perfect speed up. Indeed, Fig. 9 illustrates that the speed up rapidly becomes non linear. Nevertheless, we see that for all the exchanges frequencies (cases 1 to 5), the coupled application offer a perfect relative speed up comparing to AVBP standalone computations.

FIGURE 9. Speed up of the coupled model (C_c) on corail.

Conclusions

Based on an existing aerothermal application composed of the AVBP and AVTP solvers coupled with O-PALM, we have moved to a similar application with CWIPI in a couple of days. We have shown the non regression of the coupled aerothermal results on very academic configurations. Finally, speed up tests have shown that CWIPI has a very good comportment until 256 processors. Further diagnostics have done on bigger configurations allowing to use larger numbers of processors and exchanging more data.

REFERENCES

- DUCHAINE, F., CORPRON, A., PONS, L., MOUREAU, V., NICLOUD, F. & POINSOT, T. 2009 Development and assessment of a coupled strategy for conjugate heat transfer with large eddy simulation: Application to a cooled turbine blade. *Int. J. Heat Fluid Flow* **30**, 1129–1141.
- HYLTON, L., MIHELIC, M., TURNER, E., NEALY, D. & YORK, R. 1983 Analytical and experimental evaluation of the heat transfer distribution over the surfaces of turbine vanes. *Tech. Rep.* CR 168015. NASA.
- WLASSOW, F., DUCHAINE, F., LEROY, G. & GOURDAIN, N. 2010 3d simulation of coupled fluid flow and solid heat conduction for the calculation of blade wall temperature in a turbine stage. In *ASME Turbo expo* (ed. GT2010-22513). Glasgow, UK.