EXPERIENCES WITH PORTING THE PARALLEL CFD CODE N3S TO FORTRAN 90

J.P. GREGOIRE
D.E.R. Electricité de France
Département MMN
1, avenue du Général de Gaulle
92141 CLAMART CEDEX-France
jean-pierre.gregoire@edf.fr

Abstract
We first present the difficulties we have overcome using the CF90 parallelizer-compiler to obtain a parallel version of the N3S code which is similar to that obtained using the CF77 parallelizer-compiler. We also discuss about the stability of the results produced after this porting. We finally describe the difficulties encountered in controlling the memory consumption when moving from the Fortran 77 static memory allocation to the Fortran 90 dynamic memory allocation.

Introduction

N3S is a finite-element computation code that solves the non-stationary Navier-Stokes equations for incompressible turbulent thermal fluids. The bulk of the computation takes place in a time-marching on-loop until the stationary state of the flow will be reached.
The Electricité de France Research Division (EDF/DER) started the development work on N3S in 1982. Since then, the program processing capacity has grown to the point where it today handles large complex meshes containing one million nodes.
Using the Fortran 77 compiler the code has already undergone advanced vectorization and low level parallelization producing a speedup of 3 in production mode.
Compared to the initial version of N3S (running on a CRAY 1), the recent one (running on a CRAY 98) presents a speedup of 5874 as detailed in table 1.
To prepare the next replacement of the C98 we have ported the code to Fortran 90. This paper presents the difficulties we have encountered during this porting.

1- Compiling with CF90

1-1 Sequential compiling

Among the numerous options of the CF90 compiler, in order to preserve the results we prefer to change the general -Oopt option by the following specific ones :

-Otask0	sequential compiling
-Oscalar1	low scalar optimisation, results can differ from the results obtained when -Oscalar0 is specified, but needed by -Ozeroinc used to force each do loop, in N3S, to be executed at least one time
-Ovector1	only inner loops are vectorized, not all vector reductions are performed

Using these options the CF90 compiler generates some diagnostics due to the fact that it is less permissive than CF77 one.

For example, the double test :

```
if(iter.gt.1.and.rrp.ne.0.) gama=rrx/rrp
```

requires that rrp must be initialised even if iter=1.

After these easy modifications, we ran the code, on a turbulent test case containing 100 ,000 mesh points, and observed, at the first time step, a relative difference of 10^{-13} on the results compared to those computed with the CF77 version of the code. Unfortunately this difference is increasing as the time steps are marching on (this subject is developed latter).

1-2 Parallel compiling

As the code was previously parallelized using !MIC$ directives, we moved from sequential compiling to parallel one, by only changing, in the compiler options, -Otask0 by -Otask1, so !MIC$ directives are recognised and no autotasking is performed .

We don't use -Otask2 to avoid loop nest restructuring.

We submitted the parallelized (by CF77 directives) code to the CF90 compiler. The only diagnostics generated were warning ones of the form :

```
the shared variable ... « may need a guard »
```

These diagnostics, absent using CF77, must be analysed carefully because a guard directive inhibits vectorization and slows down parallelization.

For instance in the parallel algorithm of matrix-vector product Y=A*X, where the permuted matrix A is decomposed in NBLOCK blocks and each block is stored using the jagged-diagonal format :

```
CMIC$       DO ALL  SHARED (...), PRIVATE (...)
        DO 1000  IBLOC = 1,NBLOCK
C        | W = 0.   on the block IBLOCK
         |    IFIN= LONPAR (1, IBLOCK)
         |    DO  I = 1, IFIN
         |    V  W(I) = 0.
         |    END DO
         |
C       W = A * X   on the block  IBLOCK
             NDIAGX = NFRPAR (IBLOCK)
             DO  JDIAG = 1, NDIAGX
                IDECAL = IDEBPAR (JDIAG, IBLOCK) - 1
             $  IFIN = LONPAR (JDIAG, IBLOCK)
Parallel        DO    I = 1, IFIN
                V  W(I) = W(I) + COEF (IDECAL + I) * X (ICOL (IDECAL + I)
                END DO
             END DO
C       Y = permuted W    on the block IBLOCK
             IDEB= IDEBPAR (1, IBLOCK)
C DIR$ IVDEP
             DO  10 I = 1, IFIN
             V  Y (IPERM(IDEB + I)) = W(I)
 10          CONTINUE
 1000 CONTINUE
```

If , as proposed by the compiler, we modify the DO 10 loop by including guard directives :

```
CMIC$ guard
             DO  10 I = 1, IFIN
                Y (IPERM(IDEB + I)) = W(I)
10           CONTINUE
CMIC$ end guard
```

inside each processor, the vector register containing W(I) must go through the guard control process of Y before to be stored. Consequently parallelism is stopped and the vectorization of the DO 10 loop is slowed down.

The diagnostic of the compiler is correct : on each matrix block the DO 10 loop does not contain any vector dependency but between 2 blocks vector dependencies can occur.

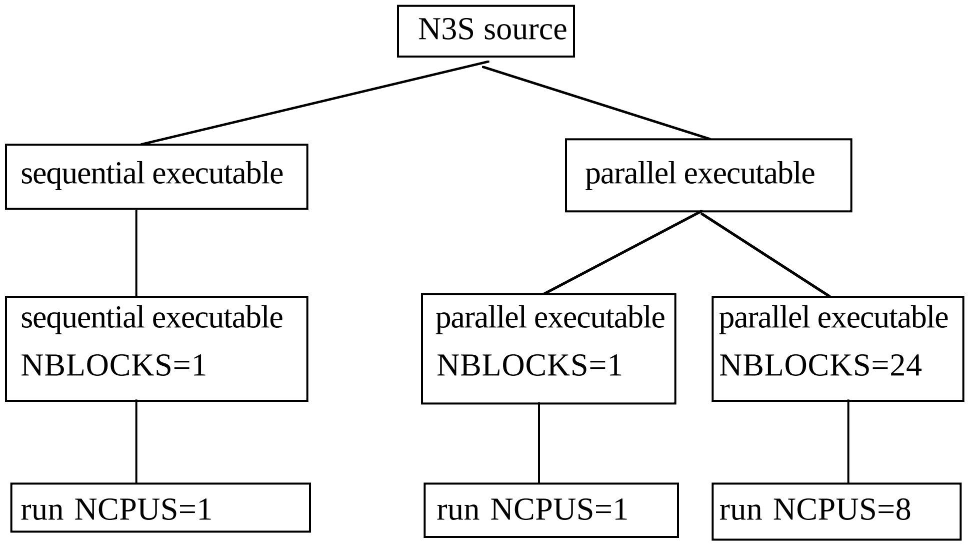
However, we know that in our code no vector dependency occurs and, to preserve computation speed, we neglected this diagnostic.

<u>2- Stability of the results with CF90</u>

<u>Preliminary remark</u>

The N3S code is delivered under quality control based on a set of reference results. Porting the code to Fortran 90, our goal is to obtain results as close as possible to the reference ones.

The parallel version of N3S needs to specify in the source the value of NBLOCKS. As the code is runned in production mode, the number of available processors is varying at each time, so NBLOCKS must be near a multiple of 8 on a C98 : we chose NBLOCKS=24.
In order to validate the parallel version, we have to verify the stability of the results between the 3 following different runs :


N3S source
sequential executable
parallel executable
sequential executable NBLOCKS=1
parallel executable NBLOCKS=1
parallel executable NBLOCKS=24
run NCPUS=1
run NCPUS=1
run NCPUS=8

<u>2-1 Stability of the results of the parallel version</u>
Whatever the number of processors used and NBLOCKS specified we verified that the binary results are identical. This fact proves the well working of CF90 parallelizer.

<u>2-2 Difficulties to obtain the stability of the results when passing from the sequential CF90 to the parallel CF90 compiler</u>
Using the sequential or the parallel CF77 compiler and avoiding the parallelization of dot products we obtained that the binary results were always identical. We tried to obtain the same conclusion using the CF90 compiler.
Unfortunately the results were different.
The localisation of the sources of these differences was difficult because the CF90 compiler does not create the modified-parallel code source.m containing the !MIC$ directives. We can only refer to the compiling listing which is sometimes ambiguous in describing where are the parallel regions handled by the compiler.
So we used numerical printings and measured a relative difference of 10^{-9} on the matrix coefficients after the first matrix assembling.

This important difference indicates that the sequential and the parallel computations of these coefficients were not identical.

We discovered that the matrix assembling of M_GLOB, achieved by the following DO 21 loop, was parallelized while -0task1 CF90 option was specified (only user tasking) :

```
SUBROUTINE  K3DI33
---------------------------------------------------------
CMIC$     DO ALL  SHARED (...), PRIVATE (...)
      DO   IEL = 1,NELEM
      | computation of elementary matrix  M_ELE(IEL,35)
      ENDDO
C assembling of M_GLOB
      DO 21 K=1,35
CDIR$ IVDEP
      |     DO 20 IEL=1,NELEM
      |     |    KK=I_ADD(IEL,K)
      |     |    M_GLOB(KK)=M_GLOB(KK)+M_ELE(IEL,K)
 20   |     CONTINUE
 21   CONTINUE
```

To prove this parallelization we generated the assembly code using -eS option of CF90, and on this later we found the parallel regions, implemented by subroutines, by searching the string of characters : ‘’IDENT _tsk_ K3DI33_..’’

We found 2 parallel regions proving that the assembling of M_GLOB was parallelized.

This parallelization is not correct because I_ADD(IEL,K) does not contain dependencies for a fixed K (as indicated by CDIR$ IVDEP) but certainly contains many dependencies between 2 different K, so parallelism cannot be implemented on K DO loop.

We forced the non-parallelization by including the 2 directives CDIR$ NOTASK and CDIR$ TASK.

After this modification, we measured a relative difference of 10^{-15} on the coefficients of the elementary matrices M_ELE(NELEM,35).

This low difference indicates that CF90 compiles differently a pure vector loop in sequential mode compared to parallel mode.

To avoid this compiling difference we replaced the DO IEL loop by a calling sequence to a subroutine containing the original loop but computing only n*128 elementary matrices.

Using this trick we verified that the binary results were identical when passing from the sequential to the parallel CF90 compiler.

In consequence the sequential version of the code was left and replaced by the parallel version with NBLOCKS=1. So we have to maintain only one version of N3S code.

3- Instability of the results when passing from CF77 to CF90, instability of the code

In order to validate the CF90 version of N3S code we compared, on unitary test cases, the results obtained with this version to those obtained with the official CF77 version.
We discovered a relative difference on each variable, used in the flow modelisation, increasing as the time is marching on :

	time=1	time=2	time=50	time=100
rela.diff. on velocities	3.10^{-12}	8.10^{-10}	2.10^{-3}	4.10^{-3}
rela.diff. on pressure	3.10^{-12}	2.10^{-9}	2.10^{-3}	8.10^{-3}
rela.diff. on K	10^{-15}	5.10^{-12}	3.10^{-2}	5.10^{-2}
rela.diff. on e	10^{-15}	8.10^{-12}	2.10^{-2}	3.10^{-2}

Despite our efforts we could not reduce this difference. N3S code is not enough robust to resist when the order of the floating point operations is changed.
This fact is due the method used to solve the Navier-Stokes equations by splitting these ones in 2 fractional steps : the non linear convection step, in which each variable is transported by the velocity of the previous time step, and the Stokes step solved by implicit solver. A slight difference on the velocity has a large effect in the convection step.
We are correcting this problem by implicitly solving together the convection and the Stokes steps.

4- Computational speed comparison when passing from CF77 to CF90

Using the hardware performance monitor, the hpm command reports the machine performance during the sequential or the parallel executions of the code, always in production mode in our case.
From this report we extracted the following table :

	par. CF77 1cpu	par. CF77 8cpu	seq. CF90	par. CF90 1cpu	par. CF90 8cpu
floating ops	45.84G	45.82G	46.34G	46.73G	46.73G
V floating ops	35.65G	35.65G	36.89G	36.90G	36.90G
mem references	78.38G	78.42G	77.76G	80.75G	82.84G
avg. conflit/ref.	0.69cp	0.93cp	0.61cp	0.60cp	0.69cp
V length	104.79	104.76	111.84	111.40	111.54
MFLOPS	**57.88**	**55.64**	**64.58**	**61.52**	**57.08**
sys. cpu time	52.34s	53.86s	50.35s	46.91s	50.21s
user cpu time	794.41s	825.35s	719.76s	761.84s	820.42s
connec. time	**/**	**293.69s**	**/**	**/**	**292.18s**
concur avg.	**/**	**2.81**	**/**	**/**	**2.81**
price	479.83F	197.01F	436.40F	458.30F	194.03F

(in order to favour parallelism the price in parallel execution is the user cpu time price/concur avg.)

Comments

1- the computational performances are slightly greater with CF90 in sequential and in parallel mode,
2- the resulting parallel speedups (concurrent average of 2.81) are identical,
3- CF77 generates less floating operations and less memory references,
4- but CF90 vectorizes better and produces some MFLOPS more,
5- in the 2 cases, the implementation of the parallelism is attractive, slight increase of the user cpu time compared to a parallel speedup of 2.81 which drops the computational price by the same factor according to our tarification policy.

5- Difficulties encountered in controlling the memory consumption in CF90

In the previous version the memory allocation was static : 2 large vectors (an integer and a real one) was defined before the compilation in order to contain all the arrays in the code.
Now these 2 vectors are dynamicaly allocated and deallocated twice : at the first time a small allocation for the data input and then, with the correct size, a large allocation for the calculation.
We also transformed the data input using the object-oriented programming, available in CF90, which contains dynamic memory allocation.
After these 2 changes we discovered that the memory consumption was increased by a factor of 50%.
We traced the memory allocation, without recompilation, by adding the ‘’malloc’’ library at the link and submiting the run with ‘’MEMTRON=1 ; export MEMTRON’’.

It appeared that the objects are mixed with specific zones generated by internal reads (converting characters to integer or real). These zones are not freed by the system and that pollutes the memory.
For instance, suppose that the memory is of the form :

allocated	Z1	free

The following internal WRITE :

```
                WRITE(LIGNE,'(I8)')I
```

implies the allocation by the system of the W zone to copy LIGNE :

allocated	Z1	W	free

If then the program allocates Z2 :

allocated	Z1	W	Z2	free

and deallocates Z1 et Z2 to collect the memory, we obtain :

allocated	free	W	free

The system does not deallocate the W zone, in order to reuse it later, so the free zone is segmented in 2 parts.
As the program will allocate now a large array, the first free zone is wasted.
A solution to this problem may be to change our array management on 2 large vectors to a dynamic allocation of each array in order to reuse the small free zones.

<u>Conclusion</u>

We have encountered difficulties in porting the parallel N3S code from CF77 to CF90 mainly because pure ascendant compatibility is not maintained between these 2 compilers. Another reason is that CF90 does not generate the modified-parallel code ‘’source.m’’ containing the parallel diectives.
These difficulties were overcame and we have now a new ‘’source.m’’ which may be easily compiled without automatic parallelization on any platform.
Computational performances of the code generated by the 2 compilers are similar with a slight advantage to CF90.
By modifying some parallel routines, the results using CF90 are identical in sequential or parallel mode.
We could not reproduice this stability of the results when porting the code from CF70 to CF90. This problem is due to the numerical sensitivity of the convection solver.
In the next future this problem will be corrected by using a more robust Navier-Stokes solver.

References

[1] J.P. GREGOIRE, B. NITROSSO, G. POT.
‘’Conjugate gradient performances enhanced in the C.F.D. code N3S by using a better vectorization of matrix vector product’’, IMAC'S 91, Dublin, July 1991.

[2] J.P. GREGOIRE, G. POT.
‘’Speed-up of CFD codes using analytical FE calculations’’, 14 th. Int. Conference on Numerical Methods for Fluids Dynamics’’, Bangalore, India, July 1994.

[3] J.P. GREGOIRE, B. NITROSSO, Y. SOUFFEZ, G. ROTH.
‘’Speedup ol parallelized N3S code on CRAY C98 in production mode’’, IX International Conference on FINITE ELEMENTS IN FLUIDS, Venezia, October 1995.

[4] J.P. GREGOIRE› J.D. MATTEI, G. SIMEONI ·
‘’Parallelization of ESTET code on CRAY C98’’, HPCN 97, Vienna, April 997.

[5] J.P. GREGOIRE, B. THOMAS.
‘’Speed-up comparison between FE and FD parallel CFD codes on C98’’, CUG97, San Jose, June 1997.

	algori. evolution	**code optimization (**speedup **S)**	**accum. speedup**	**computer**
1979	N3S code		**S=1**	CRAY 1 (1Mw)
1983	tetrahedron FE advection 1-order UZAWA algori.	mat. assembling CG solver (S=3) →	**S=3**	
1985		CG optimization (S=e) mat. reformation (S=8) →	**S=24** **S=96** ←	CRAY XMP 2(4Mw) (hardware gath-scat) (S=4)
1987	gene. UZAWA alg .			
1989	CRAY France price	CG vectorization (S=4) →	**S=384**	CRAY YMP 4(64Mw)
1990	advection 2-order	projected CG		
1992	variable density	analy. FE calcula.(S=3) →	**S=1152**	CRAY YMP 4(128Mw)
1994			**S=1958** ←	CRAY C90 8(256Mw) (S=1.7)
1995		parallelization batch mode (S=3) →	**S=5874**	CRAY C90 8(512Mw)
1998		FORTRAN 90 (S= e) →	**S=5874**	

EVOLUTION AND OPTIMIZATION OF N3S CODE