

Parallelization of the weather forecast code Mephysto

C. Calonaci¹, P. Malfetti¹, S. Campagna¹, P. Faggian², D. Ronzio²

1. CINECA, via Magnanelli 6/3, Casalecchio di Reno, Italy

2. CESI S.p.A., via Rubattino 54, Milano, Italy

Abstract

This paper describes the parallelization of the weather forecast limited area model named Mephysto. We describe the problems that arose, how we overcame them and we analyze the performance on different computing platforms.

1. Introduction

Mephysto is a numerical limited area model that simulates the time evolution of the weather physical variables in the hydrostatic approximation. This code was first developed by UB/MNC (University of Belgrade and National Meteorological Center), then by CESI (Centro Elettrotecnico Sperimentale Italiano, Milano) who updated it by substituting the original physical schemes used in the operative model at NCEP (National Centers for Environmental Prediction, Washington). CESI runs Mephysto daily in operative mode to compute a 72 hours weather forecast on a grid including most of the Europe and the whole of Italy. The original scalar version of Mephysto takes about 2 hours and 45 minutes on a vector Cray T90 processor.

Initially Mephysto was ported and parallelized on a SGI ORIGIN 3800 equipped with 128 MIPS R14000/500 MHz processors; later the model was run on a IBM SP3 equipped with 128 Power3/375 MHz processors and on a COMPAQ ALPHASC having 32 EV67/833 MHz processors.

The code has been parallelized using the shared memory programming paradigm OpenMP: this choice permits a quite quick parallelization and the possibility of exploiting data structures shared among the processor's memories. In addition no heavy reorganization of the algorithm is required and the original readability is safeguarded, so it is simple to introduce new aspects and to change the parallel version of the code. OpenMP directives are standard *de facto* [4][10], so the code is portable in any shared memory environment.

In this work the techniques used to parallelize the code are described and the experimental performance of the parallel versions is shown. We test a complex OpenMP code, not a simple benchmark made of singular constructs: in this way it is possible to understand on every platform how the implementation on some particular OpenMP directives influences the scalability of a real application.

This paper is organized as follows: the structure of the model and the numerical schemes applied are briefly described in paragraph 2; paragraph 3 depicts the three phases (porting, single processor tuning, and parallelization); parallel architectures involved and experimental performance results are analyzed in paragraph 4; concluding remarks are presented in paragraph 5; finally, in the last paragraph, the bibliography is given.

2. Description of the Mephysto limited area model

The Mephysto (MEteorology for HYdroelectric STORAGE Optimization) model is a tridimensional limited area model, nested in the general circulation model of ECMWF (European Center for Medium Range Weather Forecasts), used to compute short time range weather forecasts.

It solves the equations that describe the dynamics and the thermodynamics of the atmosphere by a finite difference method. Mephysto differs substantially from the original version UB/NMC [5][9], by the use of different radiation and precipitation schemes and because of the different parameterization of the planetary boundary layer.

Use is made of Ritter and Geleyn's radiation scheme [11] is used, which is based on resolution of the radiation transfer equation of the δ -two-stream version. The description of the precipitation processes in convention conditions is based on the Betts and Miller scheme [1][2] but with some significant changes [12] in order to adapt the scheme, which was originally designed for the tropics, to Italy latitudes. The large scale precipitations are described using the schemes by Sundqvist *et al.* [13] and Zhao *et al.* [14].

The turbulent motions are described according to the Mellor-Yamada's scheme [8] with some *realizability conditions*, in cases of strong stability or instability thermodynamic conditions. Loboeki's integral relations [7] are used to describe surface layer turbulent motions.

The model is applied on a horizontal computational domain whose dimensions are [1W,23E] longitudinal degrees and [33N,55N] latitudinal degrees.

In the horizontal, Arakawa's E *semi-staggered* grid is used, in which scalar variables (e.g. pressure, temperature, humidity) are computed in H grid points and wind components in V ones.

In the vertical the η coordinates system is used:

$$\eta = \frac{p - p_T}{p_H - p_T} \eta_s \quad \text{where} \quad \eta_s = \frac{p_{ref}(z_s) - p_T}{p_{ref}(0) - p_T} \quad 0 < \eta_s < 1$$

and $p_{ref}(z)$ is the reference pressure for the standard atmosphere.

η is a profile analytically determined using a non linear function; such a function was chosen to make the thickness between two layers as large as one hundred meters up to the top of the domain, fixed at 30 mb.

The computational grid implemented in the code consists of 48 vertical levels and every horizontal plane has 97x177 grid points.

The subroutines used in the code can be subdivided into several groups: every group is relative to a particular physical phenomenon. The evolution of the meteorological observables is simulated by calling these groups in the proper order.

Table 1 shows the order in which single subroutines (*radiaz*, *vtadv*, *hzadv*) or subsets of functional subroutines (Fix1, Fix2, Physics) are executed for every time step Nt to simulate the weather evolution on the grid.

Nt = 0	1	2	3	4	...	60	61
Init	<i>radiaz</i>				...		<i>radiaz</i>
Outdyn					...		
	Fix1	Fix1	Fix1	Fix1	...	Fix1	Fix1
	<i>vtadv</i>		<i>vtadv</i>		...		<i>vtadv</i>
	Fix2	Fix2	Fix2	Fix2	...	Fix2	Fix2
		<i>hzadv</i>		<i>hzadv</i>		<i>hzadv</i>	
		Physics			...		

Table 1 – Scheme of subroutine calls

The roles of each subroutine or subset of subroutines are as follows:

radiaz describes the radiation processes in the atmosphere;

hzadv computes horizontal advection at the H points of the grid;

vtadv computes vertical advection at the V points of the grid;

pgdtho, *pdte* (Fix1) compute pressure and vertical velocity trends;

pdnew, *bocoh*, *pguv*, *ddamp*, *bocov*, *rdtemp*, *hdiff* (Fix2) update fields and boundary conditions of the meteorological variables and compute horizontal diffusion;

profq2, *surfce*, *gcond*, *profhv*, *cumulus*, *precip* (Physics) describe the physics of meteorological processes, specifically of turbulent motions and precipitation processes.

From the table 1 it is possible to observe that dynamic subroutines (Fix1 and Fix2 subsets) are activated every time step; advection subroutines (*hzadv* and *vtadv*) are activated every two time steps; they are not in time-phase so as to increase the numerical stability of the model.

3. Mephysto Parallelization

Porting and Single Processor Tuning

Mephysto is a scalar Fortran 90 code and had been optimized to run on a vector CRAY T90 processor.

The problems faced in this phase were related to the numerical precision of REAL variables, variable initialization and floating point exceptions removal.

The default dimension of REAL variables is 8 bytes on CRAY platforms and 4 bytes on SGI ones. Using compiler flags it is possible to set the numerical precision of these floating point variables to 8 bytes, but this choice degrades heavily the performances. The solution adopted was to use 4 bytes precision as default and, in the subroutines where 8 bytes precision is required to have numerical correctness, to declare DOUBLE PRECISION (8 bytes) only for the appropriate variables. In this way a good balance between execution speed and correctness of results has been reached.

Then, where necessary, the variables were explicitly initialized. Underflow exceptions were found, but their number has been reduced by an order of magnitude thanks to a reorganization of a code fragment where most of the underflows was generated. The rest of the remaining underflows have been masked with the zero value; this operation is numerically correct for this algorithm.

Single processor tuning has been focused on a careful and effective use of the compiler (i.e. to take advantage of the superscalar functional units) and on the utilization of the fastest math libraries available.

On each platform a tuned executable for the processor hardware has been produced and, for example, on a MIPS R12000/300 MHz the execution time has been decreased from 66 to 17 hours for a 72 hours weather forecast for the IOP-8 episode (see paragraph 4).

Parallelization

An incremental parallelization approach has been chosen: subroutines have been sorted with respect to CPU time required and have been parallelized according to this order; all subroutines with a CPU time percentage greater than 0.3% have been parallelized. Most of the subroutines spend most of the time in double nested loops, computing 2-dimensional matrices. These arrays represent some physical variables (e.g. temperature, relative humidity, pressure) spread in the 3-dimensional domain; the first index refers to a location in the horizontal plane, the second index to the vertical level. Most loops, thanks to the schematization of the physical processes, could be optimally parallelized using an OMP PARALLEL DO directive on the outer loop (parallelizing on vertical levels of the atmosphere) if all dependencies between iterations can be avoided.

The basic idea is the following:

Original:	→	Parallelized
do l=1, vert_levels		!\$OMP PARALLEL DO
do inside_the_layer		do l=1, vert_levels
...		do inside_the_layer
		...

Some dependencies between iterations in vertical loops are present in the original scalar code, not due to physical reasons but due to the algorithmic implementation. The removal of these dependencies can be done using techniques such as the pre-computing of some variables (otherwise originally computed by a previous iteration), postponing of the update of some matrices with respect to the original point in the code (otherwise a thread could wrongly change data used by another thread), using auxiliary data-structures (to compute at the beginning all the possible values of some arrays otherwise computed by another thread, or to save useful original data when an array is read and updated in a loop), explicit computing by a mathematical expression of some quantities originally computed in different iterations, etc.

Arrays were distributed among processors in column (vertical levels) blocks, so that for most time each CPU computes on the data stored in its own local memory (this implies few communication operations).

Parametrization schemes of the vertical motions introduce some dependencies between horizontal layers for some processes. Consequently, some subroutines have non-removable dependencies between vertical levels due to these physical processes. In this case it has been necessary to parallelize on the horizontal levels and the loop order has been interchanged to have the horizontal loop as the outer one, then this loop could be parallelized (if an OMP PARALLEL DO directive is engaged in the inner loop some performance problems are going to arise due to the high number of involved forks and joins). This technique requires that all the subroutines' local arrays are transposed; otherwise the loops interchange causes a no stride 1 access, with a consequent dramatic increase of cache misses, TLB misses and execution time. The result is the following:

Original:	→	Parallelized
<pre>do l=1, vert_levels do k=1, horiz_level a(k,l) = ...</pre>		<pre>!\$OMP PARALLEL DO do k=1, horiz_level do l=1, vert_levels a(l,k) = ...</pre>

A CRITICAL directive is used to find a minimum for an array during the computation of the precipitation processes and to determine some relative physical variables that are used and shared by all the threads.

The two subroutines computing turbulent phenomena physics show such problems about vertical levels dependencies. The algorithm is restructured by merging two subroutines into one which consists of two different loops on the latitude index: the first loop executes computation of the first of the original subroutines, while the second loop refers to the second subroutine.

Every iteration on latitude is independent, so these two loops can be simply parallelized.

The following scheme shows this technique:

Original	→	Parallelized (a whole subroutine)
<pre>call sub1(...) call sub2(...)</pre>		<pre>!\$OMP PARALLEL DO do ... computation sub1 enddo !\$OMP PARALLEL DO do ... computation sub2 enddo</pre>

This technique has been used during parallelization of the subroutines concerning radiative processes too: the main subroutine has been restructured by inserting three subroutines in the latitudes loop. Every thread computes an iteration in parallel with the others, and it calls the scalar versions of the subroutines. The following scheme shows this technique:

Original	→	Parallelized
<pre> call sub1 call sub2 do i=1, ... call sub3(i) other comp enddo </pre>		<pre> !\$OMP PARALLEL DO do i=1, ... call modified_sub1(i) call modified_sub2(i) call sub3(i) other comp enddo </pre>

At the end of parallelization phase, numerical results have been verified: results of a 72 hours integration were fully consistent with the ones obtained on CRAY T90.

4. Performance analysis

The IOP-8 episode was chosen as case study to verify the model during the parallelization phase. The IOP-8 episode belongs to the MAP Project (the main international research project on Alps meteorology) and represents the meteorological situation which occurred during 20th-21st October 1999; these days were characterized by episodes of Foehn on the north slope of Alps and strong precipitations on the Costa Azzurra and western Italy. The IOP-8 episode has been chosen in order to have simulations in which all the subroutines had a significant role. The horizontal dimensions of the computational domain are [1W,23E] longitudinal degrees and [33N,55N] latitudinal degrees.

Mephysto has been run on the following architectures:

- SGI ORIGIN 3800 equipped with 128 MIPS R14000/500 MHz processors, each node has 4 processors and 4 GB RAM (ORIGIN 3800).
- IBM SP equipped with 128 Power3/375 MHz processors, each node has 16 processors and 16 GB RAM (SP3);
- COMPAQ ALPHASC equipped with 32 EV67/833 MHz processors, each node has 4 processors and 4 GB RAM (ALPHASC).

On SGI architectures every processor can reference both the node's memory (each ORIGIN 3800 node contains 4 processors) and machine's memory [6]; instead on IBM and COMPAQ systems the processor can reference only the node's memory so, on these platforms, it is possible to run a code parallelized with shared memory directives only inside the single node.

In order to evaluate the parallel performance, results are compared with those estimated by the so-called Amdhal's law which represents the parallel execution time $T(p)$ as a function of the number of processor p and of the parallel fraction f_p of the serial time T_s . According to Amdhal's law:

$$T(p) \equiv T_s \left[(1 - f_p) + \frac{f_p}{p} \right]. \quad [1]$$

The metric used to measure the performance of the code is the speedup:

$$S(p) = \frac{T(1)}{T(p)} = \frac{T_s}{T(p)} \quad [2]$$

where $S(p)$ is the speedup function. Speedup is used to evaluate the parallel performance; in the ideal case, when the code is totally parallelized ($f_p=1$), the speed-up function is the linear function $S(p)=p$. In the ideal case, the parallel fraction of each subroutine is equal to 1, but sometimes subroutines have also a sequential computational kernel that is executed only by the master processor, so the parallel fraction for each subroutine is not the ideal one.

Table 2 shows final results relative to ORIGIN 3800 for a 0.5 hours integration for the IOP-8 episode.

Number of threads	Average CPU time	S(p)	Theoretical S(p)
1	194	1,04	1,00
2	109	1,84	1,79
4	65	3,09	2,97
8	46	4,37	4,41

Table 2 – Performance of Mephysto on Origin 3800

The subroutines parallelized sum up to 92.5% but, if scalar kernels belonging to each parallelized subroutine are considered, the effective parallel fraction is 88.4%.

From table 2 it is possible to observe that, on a single processor, the parallel code execution is slightly faster than the scalar ones (in the speedup definition serial time is the execution time of the original scalar code, not of the parallel version running on one processor): probably the OpenMP directives insertion and the consequent different code compilation produce a faster executable.

Running on multiple processors the execution time is less or equal than the time predicted by Amdahl's Law; this behavior depends on two main reasons:

- a single processor execution time lower than the best sequential algorithm time (see above);
- superlinearity effects due to hierarchical memories (caches).

The increase of the number of processors causes the OpenMP directives overhead to increase and removes superlinearity effects, thus worsening the performance.

The computational kernel linearity has been verified also with the integration time: Mephysto takes 1 hour 56 minutes to integrate 72 hours weather forecast on 8 MIPS R14000/500 MHz processors.

Mephysto has been ported on SP3 and ALPHASC to test OpenMP implementation on these platforms: porting to ALPHASC was direct and simple; on IBM there were some problems regarding incompatibility between some optimization levels and OpenMP directives and between OpenMP directives and the correct memory management of some data structures.

Time result for a 1 hour integration run on ORIGIN 3800, SP3 and ALPHASC are showed in table 3 and plotted on fig.1.

	ORIGIN 3800		SP3		ALPHASC	
Num threads	Time (sec)	Speedup	Time (sec)	Speedup	Time (sec)	Speedup
1	387	1	426	1	206	1
2	217	1,78	266	1,6	118	1,75
4	129	3	192	2,22	79	2,62
8	92	4,21	175	2,43		
16	82	4,72	180	2,37		

Table 3 –Performance of Mephysto on Origin 3800, SP3 and ALPHASC

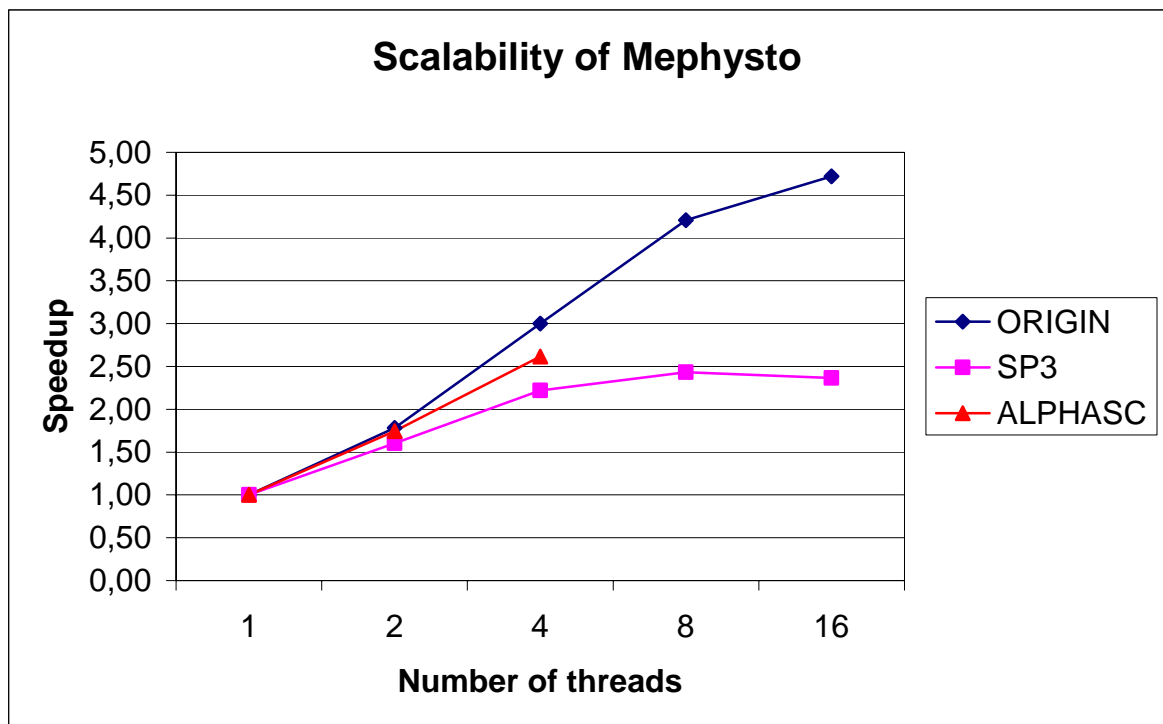


Fig. 1 – Scalability of Mephysto

SP3 does not seem the best platform to run Mephysto, due to a not very efficient optimized scalar version on a Power3 processor and to a poor scalability of the code inside the IBM node.

On a single processor, execution time on Power3/375 MHz is 10% greater than R14000/500 MHz and 107% greater than EV67/833 MHz.

Theoretical peak performances are:

- 1.5 Gflops (375 MHz * 4 FPO per cycle) Power3
- 1.0 Gflops (500 MHz * 2 FPO per cycle) MIPS R14000
- 1.666 Gflops (833 MHz * 2 FPO per cycle) EV67.

SGI/IBM and COMPAQ/IBM scalar execution time ratio are more similar to clock speed ratio than to Gflops ratio:

	SGI/IBM	COMPAQ/IBM
Time ratio	1.10	2.07
Clock ratio	1.33	2.22
Gflops ratio	0.67	1.11

As a consequence, it seems that during the code optimization phase, the IBM compiler is not able to use the two floating point units inside the processor in optimal way.

Inside the IBM node code, scalability is worse than on the other two architectures. OpenMP benchmarking programs developed by J. M. Bull at EPCC [3] show that on the SP3 OpenMP directives introduce overhead times appreciably greater than analogous times on the other platforms.

These benchmarks are able to measure OpenMP directives overhead. Comparing execution times between a scalar section of code and the same section parallelized by OpenMP directives it is possible to understand and to quantify relative overhead.

For these tests we use always the STATIC thread schedule type: in this way, according to our parallelization scheme, the number of communications needed during the computation of domain boundaries is minimum (also experimentally others schedule policies give worst performance).

PARALLEL DO is the most used directive in Mephysto, but some CRITICAL are used too. Fig. 2 and 3 report results obtained using the benchmark program.

Using 4 processors on SP3 CRITICAL overhead is comparable with overhead on ORIGIN 3800 (anyway ALPHASC has the best performance), while using 8 processors on SP3 the overhead is 30% less than on ORIGIN 3800. However, during the computation, CRITICAL sections are much less important in comparison with PARALLEL DO ones, so on SP3 the effects of this best implementation does not affect performance results dominated by PARALLEL DO overhead.

Performance results for PARALLEL DO are different: on 4 processors on SP3 overhead is 2 times compared to ORIGIN 3800 (and ALPHASC), using 8 processor it becomes 3 times compared to ORIGIN 3800.

Increasing the number of threads, from 4 to 8, on SP3 PARALLEL DO overhead time doubles, from 8 to 16 it increases about 4 times. Instead on ORIGIN 3800 from 4 to 8 threads and from 8 to 16 it increases only of a factor 25%.

This considerable increase of PARALLEL DO overhead on SP3 is the reason of the worst scalability of Mephysto and explains the different performance compared to the SGI platform.

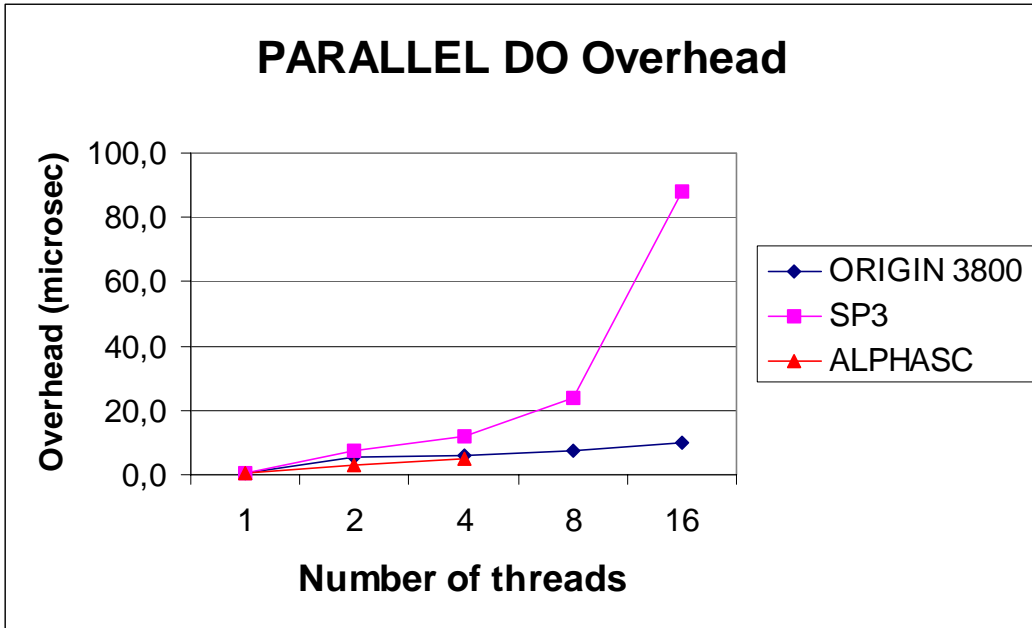


Fig. 2 – A comparison between PARALLEL DO overhead on the three platforms

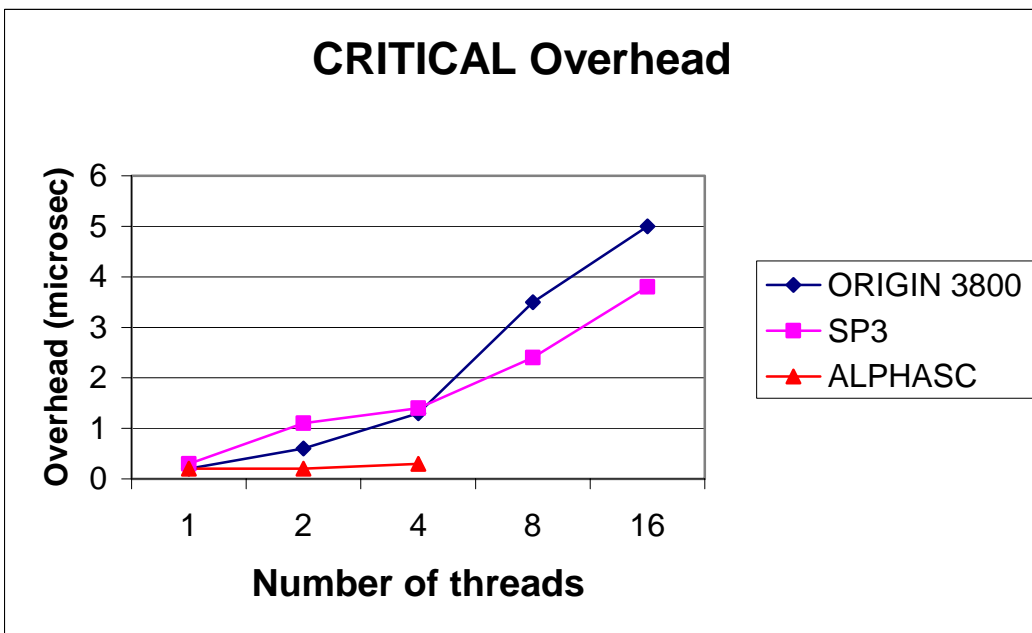


Fig. 3 – A comparison between CRITICAL overhead on the three platforms

Observing speedup trends between the remaining two platforms, the best scalability has been obtained on ORIGIN 3800: this fact can be read as a sign of the good OpenMP PARALLEL DO directive implementation on this architecture.

Despite this fact, thanks to the combination of a faster code execution and a good code optimization, a 72 hours forecast can be computed faster on 4 EV67/833 MHz processors than on 8 MIPS R14000/500 MHz processors.

On ORIGIN 3800 the 16 processors configuration does not obtain good efficiencies on the test case due to the fact that each processor compute only three vertical levels,

so per processor computation decreases and OpenMP directives overhead increases compared to the eight processors configuration; consequently the ratio computation/overhead decreases.

5. Conclusions

In this paper problems related to the porting of a weather forecast hydrostatic code from a vector to a shared memory parallel version have been discussed: if physical processes are independent along atmosphere vertical levels, this parallelization scheme is natural. Other physical processes introduce dependencies between levels and a good parallel performance can be obtained only by restructuring the subroutines computing these effects and using different schemes to avoid computational dependencies.

On an ORIGIN 3800 architecture the parallelization scheme obtains a good performance according to Amdahl's law, thanks to a good OpenMP implementation on this platform.

ALPHASC is an excellent choice to run Mephysto: thanks to the high EV67 clock speed, good compiler optimizations and OpenMP implementation it is possible to run the model on ALPHASC with a half of the processors needed on the ORIGIN 3800.

On a SP3 scalability is poor: OpenMP implementation on this architecture, at least for this kind of codes, is not competitive with the others two platforms. An OpenMP benchmarking code shows that PARALLEL DO overhead is appreciably higher compared to the others two platforms, and that this overhead increases significantly from 4 to 8 processors. PARALLEL DO overhead conditions the scalability of the entire code.

Every day OpenMP Mephysto model runs operatively on 8 processors of the ORIGIN 3800 platform available at CINECA.

Bibliografy

- [1] Betts A. K., 1986. A new convective adjustment scheme: part I, *Q.J.R.M.S.*, 112, 677-691.
- [2] Betts A. K. and Miller M. J., 1986. A new convective adjustment scheme: part II, *Q.J.R.M.S.*, 112, 693-709.
- [3] Bull J. M., 1999, "Measuring Synchronisation and Scheduling Overheads in OpenMP", in *Proceedings of First European Workshop on OpenMP*, Lund, Sweden, Sept. 1999, pp. 99-105.
- [4] Dagum L. and Menon R., 1998, "OpenMP: An Industry standard API for Shared-Memory Programming," *IEEE Computational Science & Engineering*, Vol. 5, No. 1.

- [5] Janjic, 1990: The step-mountain eta coordinate model: Further developments of the convection, viscous sublayer, and the turbulence closure scheme, *Mon. Wea.Rev.* 122, 927-945.
- [6] Laudon J. And D. Lenoski, 1997, "The SGI Origin: a ccNUMA highly scalable server", in *Proceedings of the 24th Annual International Symposium on Computer Architecture*
- [7] Loboeki L., 1993: A Procedure for the Derivation of Surface-Layer Bulk Relationships from Simplified Second-Order Closure Models. *J. Appl. Meteor.* 32,126-138.
- [8] Mellor G.L. and Yamada T., 1982: Development for a turbulence closure model for geophysical fluid problems. *Rev. Geophy. Space Phys.*, 20, 851-875.
- [9] Mesinger F, Janjic Z.I., Nickovic S., Garilov, D.Deaven D.G., 1988: The step-mountain coordinate: Model description and performance for cases of Alpine lee cyclogenesis and for a case of an Appalachian redevelopment. *Mon.Wea.Rev.*, 116, 1493-1518.
- [10] OpenMP Fortran Application Program Interfaces, Version 1.1. Technical report, November 1999.
<http://www.openmp.org/specs/mp-documents/fspec11.pdf>
- [11] Ritter and Geleyn, 1992: A comprehensive radiation scheme for numerical weather prediction models with potential applications in climate simulation, *Mon.Wea.Rev.*, 120, 303-325.
- [12] Ronzio D.A., 1997: Convezione e cumuli. *CESI Report*
- [13] Sundqvist H., Berge E., Kristjansson J.E., 1989: Condensation and Cloud Parameterization Studies with a Mesoscale Numerical Weather Prediction Model. *Mon. Wea. Rev.*, 117, 1641-1657.
- [14] Zhao Q., Carr F.H., 1997: A Prognostic Cloud Scheme for Operational NWP Models, *Mon. Wea. Rev.*, 125, 1931-1953.