Application of OpenMP to weather, wave and ocean codes

Paolo Malfetti
CINECA, via Magnanelli 6/3, I-40033 Casalecchio di Reno, Italy
Tel.: +39 051 6171411; Fax: +39 051 6132198;
E-mail: p.malfetti@cineca.it

Weather forecast limited area models, wave models and ocean models run commonly on vector machines or on MPP systems. Recently shared memory multiprocessor systems with ccNUMA architecture (SMP-ccNUMA) have been shown to deliver very good performances on many applications. It is important to know that the SMP-ccNUMA systems perform and scale well even for the above mentioned models and that a relatively simple effort is needed to parallelize the codes on these systems due to the availability of OpenMP as standard shared memory paradigm. This paper will deal with the implementation on a SGI Origin 2000 of a weather forecast model (LAMBO – Limited Area Model Bologna, the NCEP ETA model adapted to the Italian territory), a wave model (WA.M. – Wave Model, on the Mediterranean Sea and on the Adriatic Sea) and an ocean model (M.O.M. – Modular Ocean Model, used with data assimilation). These three models were written for vector machines, so the paper will describe the technique used to port a vector code to a SMP-ccNUMA architecture. Another aspect covered by this paper are the performances that these models have on these systems.

1. Introduction

In recent years it has been a common perception that only vector processor machines are appropriate for running limited area weather forecast models, wave models and ocean models. Few recent models have been developed on MPP systems, while others have been ported using the message passing paradigm.

This paper will deal with the implementation, the porting and the performances on a SGI Origin 2000 of a weather forecast model, a wave model and an ocean model.

After the description of the memory and communication architecture of the SGI Origin 2000 in Section 1, Section 2 describes a weather forecast model (LAMBO – Limited Area Model Bologna), a wave model (WA.M. – Wave Model) and an ocean model (M.O.M. – Modular Ocean Model). Section 3 describes the parallelization techniques adopted at CINECA to port these array-native codes from a vector to a shared memory multiprocessor machine using OpenMP [5]. In order to evaluate the parallel execution time performance, results are compared with those theoretically predicted. Section 4 also shows how different scalability curves can be experimentally obtained varying the size of input data; this will be done on data coming from a realistic case using three different spatial resolutions. Moreover the paper will point out the effects on the efficiency curve of the cpu upgrade without upgrading the interconnection network. Numerical representation effects on weather forecast will be briefly studied. Finally the paper provides some conclusions.

1.1. Programming environment

The SGI Origin 2000 [9] is a scalable shared-memory multiprocessing architecture. It provides global address spaces for memory and for the I/O subsystem. The communication architecture is much more tightly integrated than in other recent commercial distributed shared memory (DSM) systems, with the stated goal of treating a local access as simply an optimization of a general DSM memory reference. The two processors within a node do not work as a snoopy SMP cluster but operate separately over the single multiplexed physical bus and are governed by the same, on-level directory protocol. Less snooping keeps low both absolute memory latency and the ratio of remote to local latency, and provides remote memory bandwidth equal to local memory bandwidth (380 MB/s each). The two processors within a node share a hardwired coherence controller, called a hub, that implements the directory based cache coherence protocol.

The Origin includes other architectural features for good performance, including support for dynamic page migration and prefetching, a high-performance local and global interconnect design, coherence protocol fea-

Scientific Programming 9 (2001) 99–107
ISSN 1058-9244 / $8.00

tures to minimize latency and bandwidth needs per access, and synchronization primitives like LL/SC and at-memory fetch-and-op to reduce the serialization for highly contended synchronization events.

Within a node each processor has separate 32 KB first-level instruction and data caches (L1 cache) and a unified 4 MB second-level cache (L2 cache) with 2-way associativity on R10000 processor (from now on indicated by R10K); second level cache is 8 MB on R12000 processor (R12K).

2. Applications

2.1. LAMBO

LAMBO, Limited Area Model BOlogna, is a grid-point primitive equations model, based on the 1989 and 1993 versions of the ETA model, operationally used at the National Centre for Environmental Prediction of Washington. The model, originally developed in its former adiabatic version during the early seventies, has been consistently improved during the years, both with regards to numerical schemes, related to the adiabatic part of the model, and also with respect to the parametrization of the physical processes [2,8,12].

LAMBO has been running operationally since 1993 [14] at Agenzia Regionale Prevenzione Ambiente – Servizio Meteorologico Regionale where it has been almost completely reformulated in its pre- and post-processing sections.

As mentioned earlier, LAMBO is a grid-point, primitive equations limited-area model: in such models the only basic approximation, which is well justified by the scale analysis of the vertical component of the momentum equation, is the hydrostatic approximation, which assumes that the pressure at any point is simply equal to the weight of the unit cross-section column of air above that point. In general, a primitive equations model is a model in which, assuming that the atmosphere is in hydrostatic equilibrium, the motion is predicted by applying the principles of conservation of momentum, energy and mass (separately for dry air and moisture) and using the law of ideal gases. Such a set of differential equations constitutes the initial and boundary value problem, the solution of which provides the future state of the atmosphere. The equations of motion are solved in practice using finite difference methods and all model variables are defined on the so-called Arakawa E-type grid. Particular numerical schemes were developed to integrate on the E-grid the part of the equations related to adiabatic processes and precisely to horizontal advection [7] and geostrophic adjustment [11].

2.2. WA.M.

The Wave Model, WA.M., has been developed by a group of international scientists with the aim of producing a tool for the forecast of the waves based only on physical principles.

The WA.M. describes the sea state at a certain time in a certain position as the overlapping of many sinusoidals with different frequencies and directions. The energy distribution on these components is called the "sea spectrum".

The model integrates numerically the "energy balance equation", that expresses the equilibrium between the energy associated to the sea state in a fixed position, its advection energy and the local velocity to produce and dissipate the undulatory motion. This includes the generation from wind, energy exchange between the wave components, dissipation phenomena (as white-capping and sea bottom friction), shoaling, refraction from the bottom and interaction with the streams. The equations are solved on all the grid points and for each spectrum component.

2.3. M.O.M.

Any Ocean Data Assimilation system consists of three components: the dynamical model, the data and quality control procedures and the insertion technique. The numerical model is a modified version of the Modular Ocean Model, M.O.M., implementation in the global ocean [4,16]. M.O.M. solves the primitive equations under hydrostatic, Boussinesq and rigid lid approximations using finite difference methods. All variables are defined on the so called "B-grid" of Arakawa and Lamb [1]. The horizontal resolution is 1x1 degree almost everywhere except in the tropical area where the north-south resolution is increased to 1/3 of a degree. There are 15 levels unevenly spaced down to 3000 meters and the first 11 levels are confined in the first 250 m. The vertical diffusion and horizontal viscosity are parameterized with the Mellor-Yamada [10] turbulence closure scheme and Smagorinsky non-linear viscosity [17], respectively. At the surface the ECMWF atmospheric reanalysis fields are used to compute momentum and heat fluxes with the method implemented by Rosati and Miyakoda [13]. The surface salinity boundary condition is still a relaxation to climatological monthly mean values.

The data set assimilated into the ocean model consists of both XBT and CTD temperature profiles contained in the World Ocean Data Bank-94 [3] and

the Reynolds weekly sea surface temperature analyses [15].

The preassimilation procedure has been implemented and checked in order to ensure the most effective use of the observations.

The assimilation scheme consists of the univariate variational optimal interpolation scheme developed by Derber and Rosati [6].

3. Porting techniques

All the three codes described were running on CINECA CRAY C90 (from now on indicated with C90) and had to be ported on a Origin with 16 R10K processors at 195 MHz, with 8 GB global shared memory (from now on indicated with Origin-R10K). Later CINECA's Origin was upgraded to a 64 R12K processors at 300 MHz, with 32 GB of global shared memory (from now on indicated with Origin-R12K).

The migration of these codes to the parallel Origin system has been structured in four major steps:

- porting;
- single processor tuning;
- parallelization;
- performance analysis.

In order to obtain good MFLOP performance from porting a code written for a vector machine to a RISC processor, it is necessary to use well all memory hierarchies (especially the L1 and L2 caches) to minimize data movement from RAM and feed the cpu registers; for this reason single processor tuning is a crucial step for the scalability of these codes.

The parallel version of these codes has been written in a shared memory programming model, which is by far the most natural and efficient way to implement parallel code on Origin systems. Thus the parallelism has been achieved by the insertion of OpenMP standard directives and by exploiting the auto-parallelizing compiler features.

OpenMP permits the use of different parallelization schemes inside the same code; this flexibility is not present with other programming models (e.g. message passing).

Another advantage given by OpenMP is the possibility of using an incremental code parallelization approach: at the beginning the parallelization effort has been applied to the most time consuming routines, incrementally considering other routines to reach the desired parallelization level. SGI's ssrun tool has been fundamental in recognizing the most time consuming subroutines.

Unless specified the experiments were run using SGI's miser (or equivalent tool), so that the CPUs were dedicated to the application.

3.1. LAMBO

The purpose of this work was to follow two basic criteria:

- the parallel version of LAMBO had to run on the Origin-R10K at least in the same time as the serial C90 version;
- to retain code readability and portability the code modifications had to be kept to a minimum.

The porting process has been straightforward: the only important issue was related to the numerical precision required, due to the different default variable size on C90 (64 bits) and on Origin-R10K (32 bits).

Single processor tuning: in order to run efficiently the LAMBO vector code on the cache-based Origin architecture, aggressive optimization compiler flags had to be turned on, in particular for loop nesting and cache prefetching analysis.

The major problem arose when considering the code parallelization: the original version of LAMBO made a large use of equivalenced variables, to save memory, but the presence of an equivalenced variable in a loop inhibits its parallelization. In order to achieve a significant level of parallelism, it has been necessary to remove most of the EQUIVALENCE statements, thus reducing the code readability for the original authors.

Different parallelization schemes have been applied to different subroutines, always choosing the best approach according to the algorithm implemented: as an example, in the horizontal diffusion subroutine, HDIFF, the vertical level outer loop has been chosen for parallelization, while in the vertical advection subroutine, VTADV, the parallelization has been applied to the horizontal inner loop.

In the end it turned out that 10 subroutines were manually parallelized by the insertion of OpenMP directives and 6 were automatically parallelized by the compiler. In the case of the radiation package, the parallelization has been achieved at a higher level, by parallelizing the main loop in the driver routine which calls the other radiation routines.

The experiment was done on a $125 \times 111 \times 31$ grid, with a 60 seconds timestep, for 20 timesteps, and time redistribution between LAMBO subroutines is shown in Table 1.

Table 1
Time redistribution between LAMBO subroutines

Function	Time (s)	%
Hdiff	60,2	22
Hzadv	56,5	21
Pfdht	41,7	15
Vtadv	16,6	7
Profq2	19,1	6
Profs	14,0	5
Cucnvc	7,1	3
Ddamp	6,9	2
Rain	7,9	3
Pargel	6,6	2
Tridi	5,1	2
Qsmth	5,7	2
Pdte	4,7	2
Cloudcov	2,3	1
Rdtemp	1,5	1
Radiaz	2,0	1
Radgel	3,1	1
Comp_sp	1,2	0
Ritem	0,8	0

Table 2
cpu time varying optimization level

Optimization level	Time (s)
O2	1289
O3	1038
Ofast	918
Ofast+IEEE+r12k+lno	878

Table 3
Time redistribution between WA.M. subroutines

Function	Time (s)	%
Snonlin	4056,550	40.2
Implsch	3510,170	34.8
Propags	1562,490	15.5
Wamodel	370,056	3.7
__qerf	184,012	1.8
Stresso	178,986	1.8
__qj1	174,706	1.7
Other	$\sim$ 40,000	0.5

3.2. WA.M.

Porting: this code needed minor modifications, such as the modularization of some PARAMETERs and EQUIVALENCEs and the substitution of some CRAY proprietary subroutines.

This model was tested with a 1/8 of a degree configuration on the Mediterranean Sea, so the grid is made by 337 longitudinal points by 129 latitudinal points; moreover for each grid point 25 frequencies and 12 angles have been considered.

Single processor tuning: this model works with 64 bit numerical precision (both real and integer arithmetic) and doesn't show any numerical instabilities.

For a 6 hour time integration run Table 2 shows that cpu time decreases when the optimization level grows; the code is fastest when the optimization is refined asking for an arithmetic not compliant to the IEEE-754 standard, a code optimized for the R12K processor, with aggressive prefetching and loop fusion.

Time redistribution between WA.M. subroutines for a 72 hours run is summarized in Table 3.

Parallelization: the first four subroutines listed in Table 3 (and six subroutines called by them) were parallelized manually inserting OpenMP directives.

3.3. M.O.M.

Porting: initially the model was run with a 32 bit arithmetic and numerical representation to obtain higher execution speed but losing numerical precision. This test, however, didn't give the expected benefits, because the model has some numerically unstable kernels and it diverges when a 32 bit arithmetic is used. The model instability has been shown also for the 64 bit numerical representation when the high optimization level (that implies a non standard IEEE-754 arithmetic) has been used, in particular some transformations, such as $x/y = x*1/y$, introduced the presence of NaN (Not a Number) quantities.

Single processor tuning: due to numerical instability a non highly aggressive optimization level was selected for the numerical point of view and performance of the model was improved using the software pipelining option. The introduction of a flag that switches on the data caches prefetch (both primary and secondary) has slightly lowered the execution time, while loop fusion and loop fission flags didn't give any substantial benefit; the same happened for the interprocedural analysis option.

Profiling tools have been useful in analyzing the model behaviour and in locating a numerical kernel where most of the execution time is spent. This kernel is a nested do loop containing an instruction similar to:

$$a_{ij} = (b_{ij} * h_{ij} + c_{ij-1} * h_{ij-1} + d_{ij+1} * h_{ij+1} + e_{ij} * (h_{i+1j} + h_{i-1j})) * f_{ij}$$

A floating point analysis shows that this instruction can't exceed the 45 MFLOP/s on the Origin-R12K. The performance tools showed that with all the optimization options turned on the compiler doesn't reach this upper bound.

Loop fission, array automatic padding, array grouping (automatic and manual) inside a common block didn't give any performance gain as well as re-writing

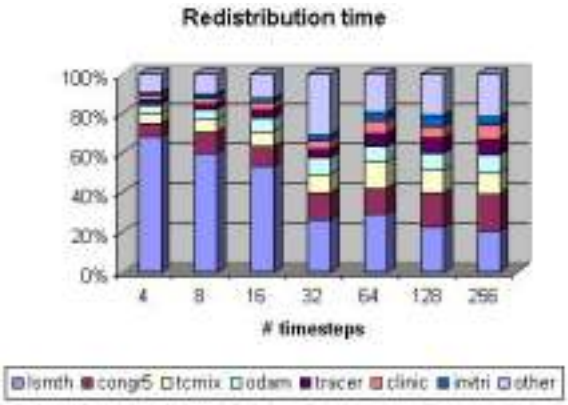


Fig. 1. Time redistribution between M.O.M. subroutines varying the number of timesteps.

the code using FORTRAN 90 for grouping the data structures involved in the loop – so to use better the primary data cache.

The cpu time redistribution of M.O.M. subroutines has been studied varying the number of timesteps, to find out a number of timesteps sufficiently small to be a good and representative sample of the model for long integrations and to minimize the execution time. The behaviour of the seven most time consuming subroutines has been observed for 4, 8, 16, 32, 64, 128 and 256 timesteps configurations.

The redistribution time in the two last configurations is almost the same. For previous configurations only a high evolution (when the configuration passes from 4 to 8, from 8 to 16 and from 16 to 32 timesteps) can be observed, then a smooth evolution (when the configuration passes from 32 to 64 and from 64 to128 timesteps) up to reach an arrangement when the configuration passes from 128 to 256 timesteps, as shown in Fig. 1.

Parallelization: the code has been passed in the Power Fortran Accelerator (pfa) to obtain a parallel version of the code.

4. Experimental results

In order to evaluate the parallel performance, the results are compared with those predicted 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:

Table 4
LAMBO parallel execution: cpu time, theoretical speedup for $f_p = 0.90$, real speedup, theoretical speedup for $f_p = 0.92$

CPUs	Time (s)	S $f_p = 0.90$	S	S $f_p = 0.92$
1	275	1,00	1,00	1,00
2	150	1,82	1,83	1,85
4	87	3,08	3,16	3,23
6	66	4,00	4,17	4,29
8	56	4,71	4,91	5,13
10	50	5,26	5,50	5,81
12	46	5,71	5,98	6,38
14	43	6,09	6,40	6,86
16	40	6,40	6,88	7,27

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

$$S(p) \equiv \frac{T(1)}{T(p)} = \frac{T_s}{T(p)}$$

where $S(p)$ is the speed-up function definition. Speed-up is used to evaluate the parallel performance; in the ideal case, when all the code is perfectly parallel ($fp = 1$), the speed-up function is the linear function $S(p) = p$.

4.1. LAMBO performance analysis

Since only routines that together account for the 96% of the total execution time were considered for parallelization, the Amdhal's curve corresponding to $f_p = 0.92$ should be considered.

Due to imperfect load balancing, cache misses and data contention between processors that fraction is positioned between 90% and 92%.

Table 4 reports the parallel execution time obtained running the experiment described previously in Section 3.1 for 20 timesteps on Origin-R10K.

Table 4 reports also the real speedup between the theoretical speedup calculated for $f_p = 0.90$ and for $f_p = 0.92$.

LAMBO has been operative on CINECA's Origin-R10K since the 1st July 1998. Using 10 R10K processors the first run takes about 5 minutes while the second takes about 32 minutes. This should be compared with the 10 minutes and 50 minutes, respectively required by the previous C90 runs.

4.2. WA.M. performance analysis

Parallel execution time obtained running the experiment described before in Section 3.2 for an integration of 72 hours on Origin-R12K are shown in Table 5.

Table 5
WA.M. parallel execution: cpu time, theoretical speedup for $f_p = 0.90$, real speedup, theoretical speedup for $f_p = 0.92$

CPUs	Time (s)	S fp=.90	S	S fp=.92
1	878,708	1,00	1,00	1,00
2	546,574	1,82	1,61	1,85
4	290,694	3,08	3,02	3,23
8	175,943	4,71	4,99	5,13
16	129,220	6,40	6,80	7,27

Table 6
M.O.M. parallel execution: elapsed time, theoretical speedup for $f_p = 0.90$, real speedup, theoretical speedup for $f_p = 0.92$

CPUs	Time (h)	S fp=.90	S	S fp=.92
1	34:34	1,00	1,00	1,00
8	7:02	4,71	4,91	5,13
10	6.11	5,26	5,59	5,81
12	5.46	5,71	5,99	6,38
14	6.14	6,09	5,55	6,86

Performance tools demonstrate that the subroutines that have been parallelized sum up to the 92% of the serial execution time. The asymptotic behaviour of the two curves is similar so the parallelization can be considered satisfactory.

4.3. M.O.M. performance analysis

The elapsed time on Origin-R12K for one month integration shown in the Table 6 were obtained with a partially loaded machine and without using miser.

The higher execution time when the number of processor passes from 12 to 14 is probably due to a high machine load. Another cause that can generate such behaviour is a bad process workload distribution or when an higher number of processors is not exploited fully in terms of memory use. In addition, the usage of another router on the communication network can increase the communication overhead because of the growth of the bisection bandwidth, in other words the maximum number of hops required for a message to reach another node grows.

M.O.M. on 12 R12K processors outperforms the 10 hours needed by the C90 run.

4.4. Input size effects

Some experiments taking as input the dataset relating to the south Ticino flood (Sept. 1995) have been done to understand the impact on the model scalability when the resolution is changed.

Under the IRIX 6.5.2 environment LAMBO has been compiled with MIPSpro 7.3 and has been run using miser on Origin-R12K.

The experiments had these configurations:

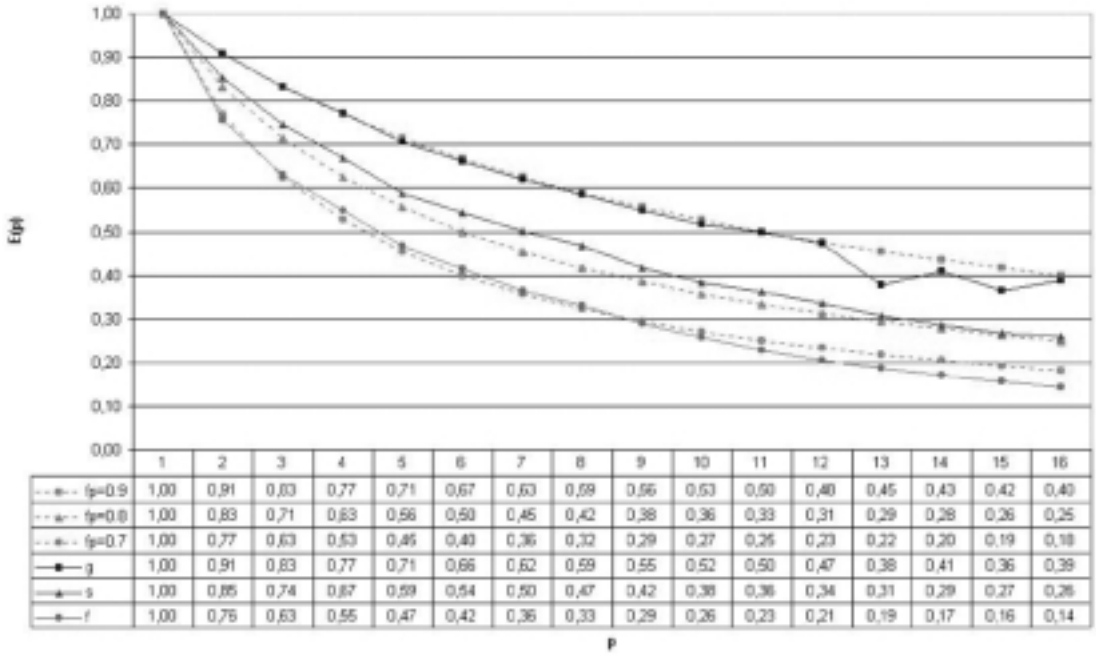
- Father: $65 \times 65 \times 32$ grid, timestep 120 seconds
- Son: $129 \times 129 \times 32$ grid, timestep 60 seconds
- Grandson: $197 \times 197 \times 32$ grid, timestep 30 seconds

Figure 2 summarizes the efficiencies coming from experimental results together with the efficiencies predicted by Amdhal's law when the parallel fraction fp is 70%, 80% and 90%. It's easy to see that LAMBO scalability (and in general all model scalability) is strongly related to the configuration of the experiment or better to the input size. This observation leads to two other considerations: to obtain models that scale well on shared memory machines, data structures have to be large enough to be distributed among the processors that have to be used, otherwise the overhead that comes from remote memory access will bring down the performances; OpenMP implementation overhead (always present) can be percentually reduced increasing the input size so as to enlarge the computational part of the model.

All these three configurations scale up to 16 processors with quite different efficiencies but when more CPUs are added there is no gain in time performance due to overheads (synchronization and remote accesses). LAMBO scales up to 32 processors if the configuration is greater than Grandson but, as mentioned earlier, LAMBO is a hydrostatic model and this kind of model can not be used for very fine grids. It has been noticed that the thread control overhead explodes over 16 processors: the experiment configuration has to be very big so that the computational part hides the overhead.

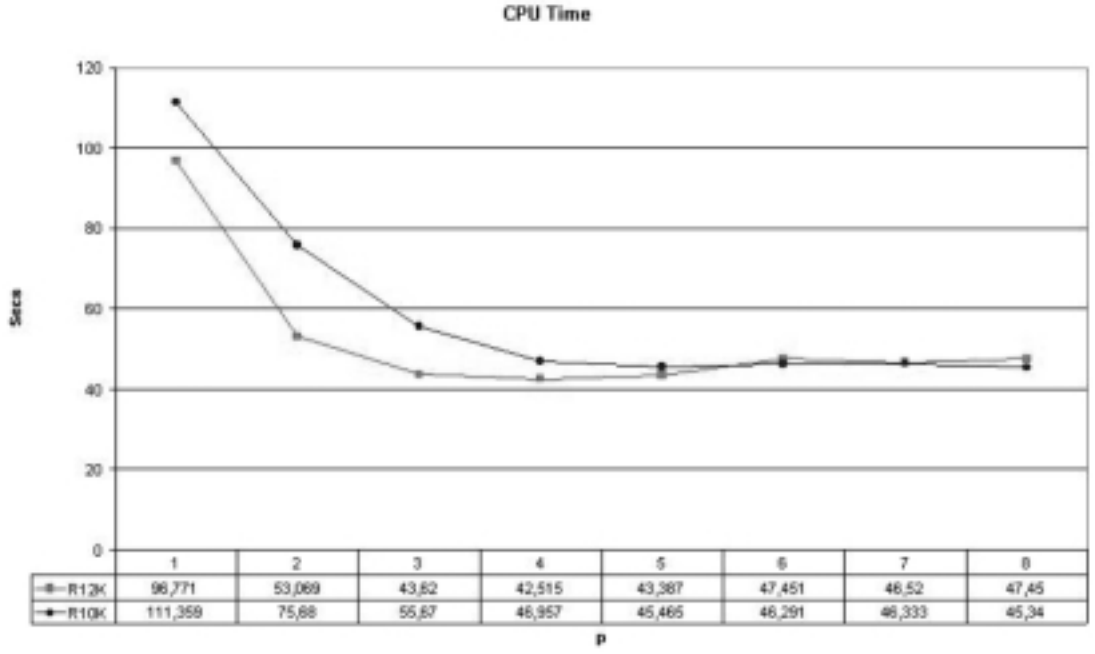
Moreover it is possible to observe that the Son configuration on Origin-R12K is larger than the one that has been used to port the code on the Origin-R10K but the former configuration has a poorer scalability: this behaviour is due to the change of cpu. Passing from R10K, 195 MHz, 4 MB L2 cache to R12K, 300 MHz, 8 MB L2 cache leads the model to run faster on a single cpu and to fit in a secondary data cache using a smaller number of processors.

The same behaviour can be observed in Fig. 3: this picture shows the WA.M. cpu time obtained on the CINECA's Origin-R12K and on CINECA's Onyx equipped with 8 R10K processors, running at 275 MHz, with 64 KB L1 cache, 4 MB L2 cache and 4 GB of memory. The experiment configuration in this case was 1/12 of a degree on the Adriatic Sea represented as a 97×73 grid, for each grid point 25 frequencies and 12 angles have been considered. The model integrates 6 hours forecast. Since the WA.M. for this



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fp=0.9	1,00	0,91	0,03	0,77	0,71	0,67	0,63	0,59	0,56	0,53	0,50	0,48	0,45	0,43	0,42	0,40
fp=0.8	1,00	0,83	0,71	0,63	0,56	0,50	0,45	0,42	0,38	0,36	0,33	0,31	0,29	0,28	0,26	0,25
fp=0.7	1,00	0,77	0,63	0,53	0,45	0,40	0,36	0,32	0,29	0,27	0,25	0,23	0,22	0,20	0,19	0,18
g	1,00	0,91	0,83	0,77	0,71	0,66	0,62	0,59	0,55	0,52	0,50	0,47	0,38	0,41	0,36	0,39
s	1,00	0,85	0,74	0,67	0,59	0,54	0,50	0,47	0,42	0,38	0,36	0,34	0,31	0,29	0,27	0,26
f	1,00	0,76	0,63	0,55	0,47	0,42	0,36	0,33	0,29	0,26	0,23	0,21	0,19	0,17	0,16	0,14

Fig. 2. Father, Son, Grandson and theoretical Amdhal's efficiencies for $f_p = 0.7,\ f_p = 0.8,\ f_p = 0.9$.



	1	2	3	4	5	6	7	8
R12K	96,771	53,069	43,62	42,515	43,387	47,451	46,52	47,45
R10K	111,359	75,68	55,67	46,957	45,465	46,291	46,333	45,34

Fig. 3. W.A.M. cpu Time on R12K and R10K.

configuration scales up to 4 processors there is no more benefit in adding CPUs infact the model is quite small. WA.M. on the Mediterranean Sea, instead, scales up to 12 processors.

4.5. Effect on numerical representation

Some experiments has been done in order to evaluate the relation between LAMBO output and numerical

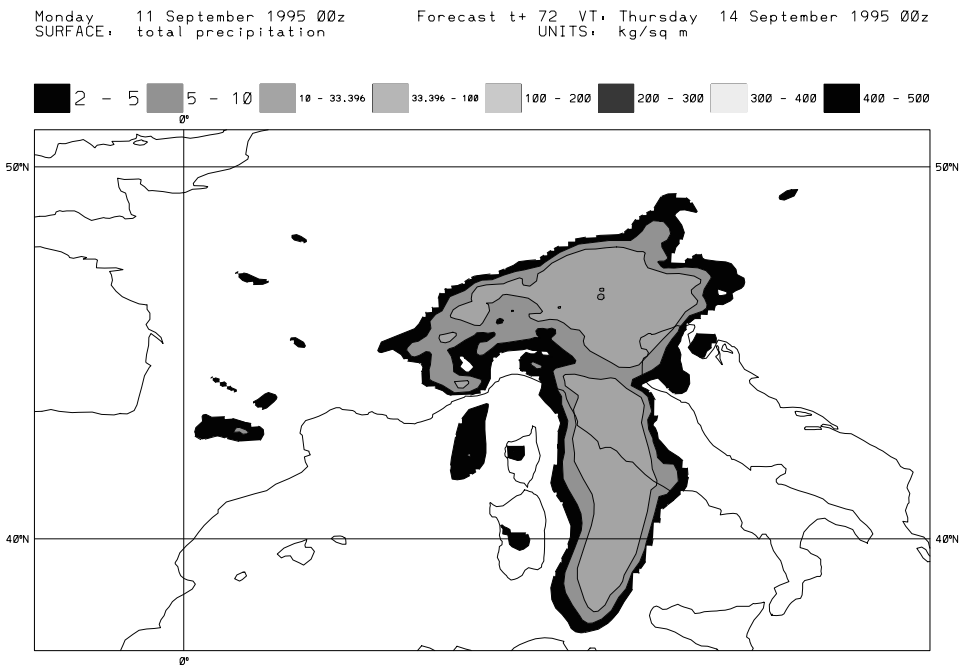


Fig. 4. Total precipitation 72 hours forecast 32 bit numerical representation.

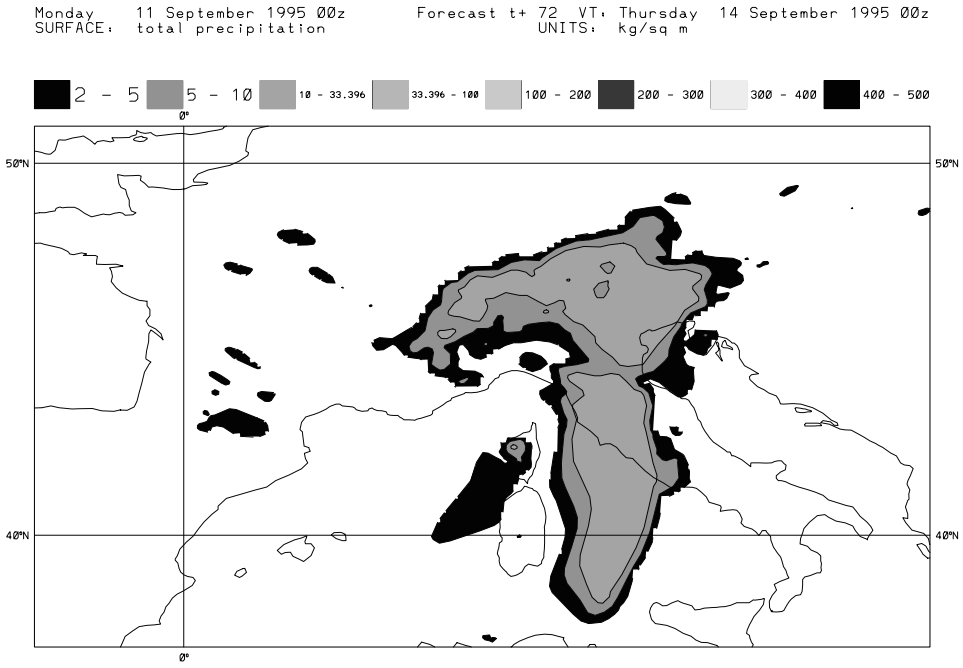


Fig. 5. Total precipitation 72 hours forecast 64 bit numerical representation.

representation. The input dataset chosen is still the one related to the south Ticino flood. This meteorological situation has been chosen because it is characterized by intense phenomena in order to have extreme values for some model variables and to highlight numerical differences and any fatal errors; moreover, with such a situation, the convective scheme has been frequently used during the computation.

The variable chosen for the comparison is the total precipitation (which is the most interesting variable for the end users). The comparison has been done on the last snapshot released by the model (after 72 hours integration) because numerical differences between two different computations grow with integration time.

Figures 4 and 5 shows respectively the total precipitation for a run with 32 bit numerical representation and a 64 bit one: the areas where total precipitation is intensive have the same structure in both the figures, while some differences are present where total precipitation is light. This qualitative analysis has been completed with a statistical analysis and also other model variables have been examined (relative humidity at 850 hPa and mean sea level pressure).

A similar comparison has been done between a scalar and a parallel run in order to test the numerical impact of parallelization on the output. Also in this case there were not quantitative and qualitative differences in test variables.

5. Conclusions

During this paper it has been shown that:

- these kinds of models do not seem to be suited only for vector machines: it has been observed that cpu time performances for all these three configurations can be better than the respective ones obtained on a CRAY C90;
- the possibility of using an incremental code parallelization approach and of using different parallelization schemes inside the same code are two big advantages given by OpenMP. A relatively simple parallelization effort has been done for porting these codes: a much greater effort would have been necessary in the case of a message passing implementation;
- single processor tuning is a crucial step: the porting of a vector code requires particular attention, especially loop nesting, prefetching and cache optimization;
- scalability is strictly dependent on the input size, the processor, the L1 and L2 cache sizes;
- these models can perform well on scalable shared memory parallel computers providing satisfactory operational forecasts also with 32 bit numerical representation.

Future work: to experiment with new commercial systems. The LAMBO serial version has been run on IBM Power3 processor at 200 MHz with 128 KB of L1 cache and 4 MB of L2 cache: a 6 hour integration takes 1823 and 4500 seconds respectively for the south Ticino flood Son and Grandson configurations, instead of 2207 and 7247 seconds were necessary for the same configurations on a R12K.

References

[1] A. Arakawa and V.R. Lamb, Computational design of the basic dynamical processes of the UCLA general circulation model, *Methods in Computational Physics* **17**, Academic Press, 1977, pp. 174–265.

[2] T.L. Black, The step mountain, eta coordinate regional model: a documentation, NOAA/NWS/NMC, 1988.

[3] T.P. Boyer and S. Levitus, *NOAA Technical Report NESDIS* **81** (1994), 65.

[4] M.D. Cox, *GFDL Ocean Group Tech. Rep.* **1** (1984), 143.

[5] L. Dagum and R. Menon, OpenMP: An Industry-Standard API for Shared-Memory Programming, *Computational Science and Engineering* **5**(1) (1998).

[6] J. Derber and A. Rosati, *J. Phys. Oceanogr.* **19** (1989), 1333.

[7] Z. Janjic, Non linear advection schemes and energy cascade on semistaggered grids, *Mon. Wea. Rev.* **112** (1984), 1234–1245.

[8] Z. Janjic, The step-mountain coordinate: physical package, *Mon. Wea. Rev.* **118** (1990), 1429–1443.

[9] J. Laudon and D. Lenoski, The SGI Origin: a ccNUMA highly scalable server, in *Proceeedings of the 24th Annual International Symposium on Computer Architecture,* 1997.

[10] G.L. Mellor and T. Yamada, *Rev. Geophys. Space Phys.* **20** (1982), 851.

[11] F. Mesinger, A method for construction of second-order accuracy difference schemes permitting no false two-grid interval wave in the height field, *Tellus* **25** (1973), 444–458.

[12] F. Mesinger, Z.I. Janjic, S. Nickovic, D. Gavrilov and D.G. Deaven, The step-mountain coordinate: Model description and performance for cases of Alpine lee cyclogenesis and for a case of Appalachian redevelopment, *Mon. Wea. Rev.* **116** (1988), 1493–1518.

[13] K. Miyakoda, A. Rosati and R.G. Gudgel, Prediction of Internal Climate Variations, NATO-ASI series, 16, Springer-Verlag, Berlin, 1997, pp. 125.

[14] T. Paccagnella, Operativo un Modello ad Area Limitata Presso il Servizio Meteorologico Regionale dell'Emilia-Romagna. AER available at Regional Meteorological Service of Emilia-Romagna, 1994.

[15] R.W. Reynolds and T.M. Smith, *J. Climate* **7** (1994), 929.

[16] A. Rosati and K. Miyakoda, *J. Phys. Oceanogr.* **18** (1988), 1601.

[17] J. Smagorinsky, *Large Eddy Simulation of Complex Engineering and Geophysical Flows,* Cambridge University Press, 1997.