Research Papers
Issue RP0238
December 2014

Advanced Scientific Computing

Performance analysis of the COSMO-CLM model

By **Silvia Mocavero**
CMCC
silvia.mocavero@cmcc.it

Antonella Nigro
University of Salento
anto.nigro90@libero.it

Arianna Resta
University of Salento
resta.arianna@gmail.com

Cinzia Rosato
University of Salento
cinzia.rosato@hotmail.it

Giacomo Sciolti
University of Salento
giacoscio@alice.it

Italo Epicoco
University of Salento & CMCC
italo.epicoco@unisalento.it

and **Giovanni Aloisio**
University of Salento & CMCC
giovanni.aloisio@unisalento.it

SUMMARY COSMO-CLM (or CCLM) is a non-hydrostatic parallel atmospheric model, developed by the CLM-Community starting from the Local Model (LM) of the German Weather Service. Since 2005, it is the reference model used by the german researchers for the climate studies on different temporal scales (from few to hundreds of years) with a spatial resolution from 1 up to 50 kilometers. It is also used and developed from other meteorological research centres belonging to the Consortium for Small-scale Modelling (COSMO).
The present work is focused on the analysis of the CCLM model from the computational point of view. The main aim is to verify if the model can be optimised by means of an appropriate tuning of the input parameters, to identify the performance bottlenecks and to suggest possible approaches for a further code optimisation. We started analysing if the strong scalability (which measures the improvement factor due to the parallelism given a fixed domain size) can be improved acting on some parameters such as the subdomain shape, the number of processes dedicated to the I/O operations, the output frequency and the communication strategies. Then we profiled the code to highlight the bottlenecks to the scalability and finally we performed a detailed performance analysis of the main kernels using the *roofline* model.

INTRODUCTION

CCLM is a unified model, used for the Numerical Weather Prediction and the modelling at regional scale. Model equations are based on rotated geographical coordinates and several physical processes are modelled through different parameterisation schemes. The model configuration can be set changing the following configuration files:

- INPUT_ORG for Setup;
- INPUT_IO for I/O;
- INPUT_DYN for Dynamics;
- INPUT_PHY for Phisics;
- INPUT_DIA for Diagnostics;
- INPUT_ASS for Data Assimilation;
- INPUT_INI for Data Initialization.

The main goal of the present work was the performance analysis of the model. It involved some groups of parameters such as:

- /LMGRID/ which allows to select the domain size and its decomposition;
- /RUNCTL/ which allows to select the execution parameters;
- /TUNING/ which allows to select the tuning variables;
- /IOCTL/ and /GRIBOUT/ which allow to control the I/O parameters.

More details about the available parameters can be found in the COSMO user's guide [2]. The CCLM test configuration taken into consideration covers the region shown in figure 1 corresponding to the alpine region $(780 \cdot 460 km^2)$, with a 2 km resolution.

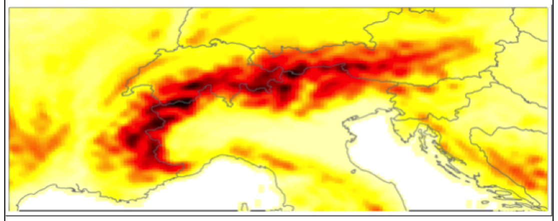


Figure 1:
CCLM test configuration

ANALYSIS OF SCALABILITY

The aim of the analysis is to verify the strong scalability of the model. A set of 4 runs (on 16, 64, 512 and 2048 cores) has been executed on ATHENA, the iDataPlex cluster with Intel Xeon E5-2670 Sandy Bridge processors, available at CMCC. The analysis has been performed acting on different factors which can influence the performance: the domain decomposition, the resources dedicated to the I/O operations, the communication strategies and the I/O frequency. Moreover, the kernel's scalability has been analyzed in detail.

DOMAIN DECOMPOSITION In this first analysis we have verified if and how the shape of the subdomain impacts on the computational performance of the model. Three different domain decompositions have been considered, changing the number of processes along the longitude and the latitude directions:

- Squared subdomains
- Longitudinal bands
- Latitudinal bands

Let $P_{I/O}$ be the number of processes dedicated to I/O operations, P_x and P_y respectively the number of processes along the longitude and the latitude axis. The total number of processes

(N_{proc}) will be given by:

$$N_{proc} = P_x P_y + P_{I/O} \tag{1}$$

Since we are interested only to analyse the impact of different subdomain shapes, we can set $P_{I/O} = 0$. The longitudinal or latitudinal bands subdomains are easily obtained setting respectively P_x or P_y to 1. While, to obtain square subdomains, we have to choose P_x and P_y such that

$$\frac{G_x}{P_x} = \frac{G_y}{P_y} \tag{2}$$

where G_x and G_y are the number of grid points respectively along the longitude and the latitude axis, with the following constraints:

$$\begin{gathered} P_x \in \mathbb{N}, P_y \in \mathbb{N} \\ P_x | N_{proc}, P_y | N_{proc} \end{gathered} \tag{3}$$

A rule of thumb for guessing a first approximation of P_x or P_y values is given by equations 4 and 5

$$P_y = \left\lfloor \sqrt{\frac{G_y}{G_x} Nproc} + 0,5 \right\rfloor \tag{4}$$

$$P_x = \left\lfloor \frac{Nproc}{P_y} \right\rfloor \tag{5}$$

In our test case, G_x and G_y, defined into the /LMGRID/ group of the INPUT_ORG file, are respectively 390 and 230. Due to the rounding, the subdomain could be not exactly square; a form factor (ϕ) has been introduced to verify the ratio between the two dimensions of the subdomain. Four different decompositions have been considered as listed below:

CASE 1: $Nproc = 15$
$P_x = 5$ and $P_y = 3$
$\phi = 1.02$

CASE 2: $Nproc = 60$
$P_x = 10$ and $P_y = 6$
$\phi = 1.02$

CASE 3: $Nproc = 510$
$P_x = 30$ and $P_y = 17$
$\phi = 1.04$

CASE 4: $Nproc = 2030$
$P_x = 58$ and $P_y = 35$
$\phi = 1.02$

The first two cases have been considered for applying the longitudinal and the latitudinal bands decomposition, fixing the total number of processes. In particular, for the longitudinal bands decomposition:

CASE 1: $Nproc = 15$
$P_x = 1$ and $P_y = 15$

CASE 2: $Nproc = 60$
$P_x = 1$ and $P_y = 60$

while, for the latitudinal one:

CASE 1: $Nproc = 15$
$P_x = 15$ and $P_y = 1$

CASE 2: $Nproc = 60$
$P_x = 60$ and $P_y = 1$

Figure 2 reports the execution time when the number of core increases, considering the three domain decomposition strategies. The best one is the square decomposition, which minimises the communication time. Indeed, a block decomposition strategy doubles the number of communications w.r.t. the bands decomposition, but the messages length decreases when the number of the processes increases. When it becomes very high, the length of the messages is too short that the block strategy is usually the best one. Moreover, the square decomposition is the best block decomposition because it minimises the perimeter of the subdomain, then the messages length.

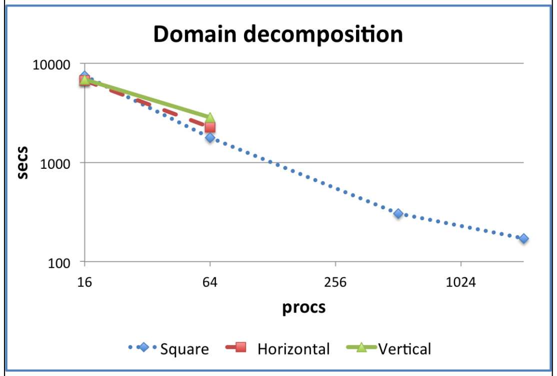


Figure 2:
Scalability analysis with different domain decomposition strategies. The blue line refers to the square domain, while the red and the green lines respectively to the horizontal and the vertical bands decomposition.

I/O RESOURCES In this analysis we are interested to verify the impact of the cores dedicated to the I/O operations on the computational performance. In this case we have fixed the decomposition strategy using square subdomains and changed the number of processes reserved for I/O. Four different cases have been considered as listed below:

CASE 1: $P_{I/O} = 4$
a) $Nproc = 16\ P_x = 4\ P_y = 3\ \phi = 1.27$
b) $Nproc = 64\ P_x = 10\ P_y = 6\ \phi = 1.02$
c) $Nproc = 497\ P_x = 29\ P_y = 17\ \phi = 1.006$
d) $Nproc = 2034\ P_x = 58\ P_y = 35\ \phi = 1.02$

CASE 2: $P_{I/O} = 16$
a) $Nproc = 61\ P_x = 9\ P_y = 5\ \phi = 1.06$
b) $Nproc = 509\ P_x = 29\ P_y = 17\ \phi = 1.006$
c) $Nproc = 2046\ P_x = 58\ P_y = 35\ \phi = 1.02$

CASE 3: $P_{I/O} = 64$
a) $Nproc = 512\ P_x = 28\ P_y = 16\ \phi = 1.03$
b) $Nproc = 2036\ P_x = 58\ P_y = 34\ \phi = 1.006$

CASE 4: $P_{I/O} = 128$
a) $Nproc = 428\ P_x = 25\ P_y = 12\ \phi = 1.23$
b) $Nproc = 2032\ P_x = 56\ P_y = 34\ \phi = 1.03$

Figure 3 reports the scalability curves for the four cases. As we can note, dedicating cores only for I/O does not bring benefits.

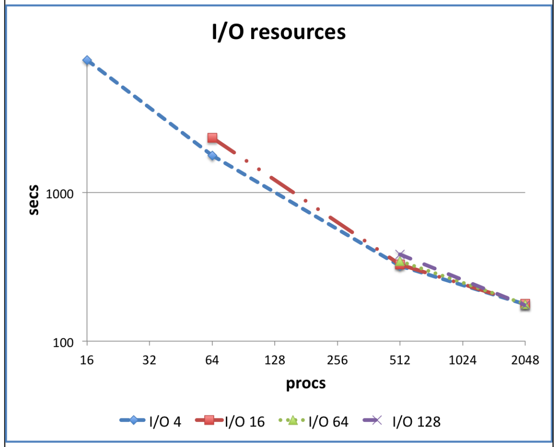


Figure 3:
Scalability analysis with different I/O resources: blue line for 4 cores, red line for 16 cores, green line for 64 cores and purple line for 128 cores.

COMMUNICATIONS The scalability has been evaluated also changing the communications strategy (fixing the square domain decomposition). The $ncomm\_type$ parameter allows to select one of the following communications:

1. Immediate send: begins a nonblocking send;

2. Immediate receive: begins a nonblocking receive;

3. Send-receive: combines in one call the sending of a message to one destination and the receiving of another message, from another process. The two processes can be the same.

Results are shown in figure 4. As we can note, the choice of different communication strategies does not affect the performance.

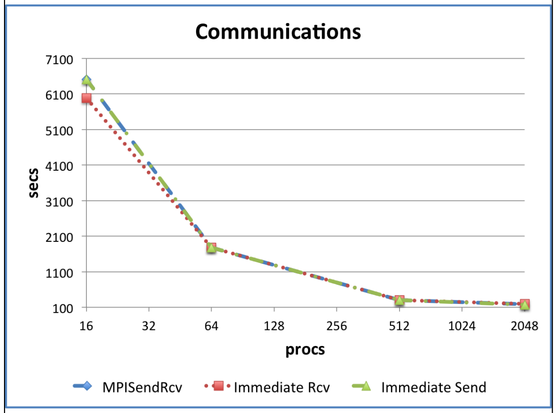


Figure 4:
Scalability analysis with different communication strategies: the green line refers to immediate send, the red line to immediate receive and the blue line to send-receive.

I/O FREQUENCY The objective was to analyse the performance changing the I/O frequency. In the /GRIBOUT/ group, the $hcombper$ parameters indicate the start and end time and the intervals for the output writing. The following cases have been considered:

CASE 1: The file is written once at the end of the run.
CASE 2: The file is written two times.
CASE 3: The file is written four times.

Figure 5 shows a comparison among the three cases. Starting from the results of the analysis carried out changing the dedicated I/O resources and the last test on the I/O frequency, we can deduce that I/O does not represent a bottleneck for the CCLM scalability.

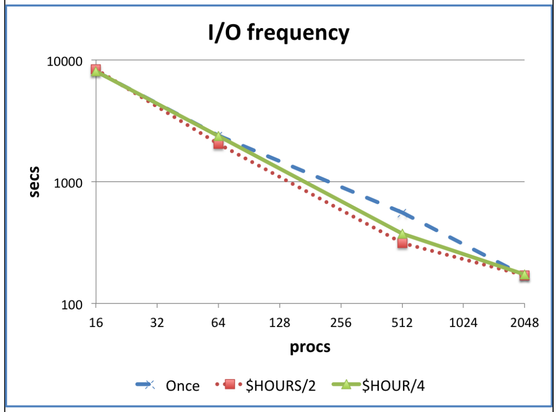
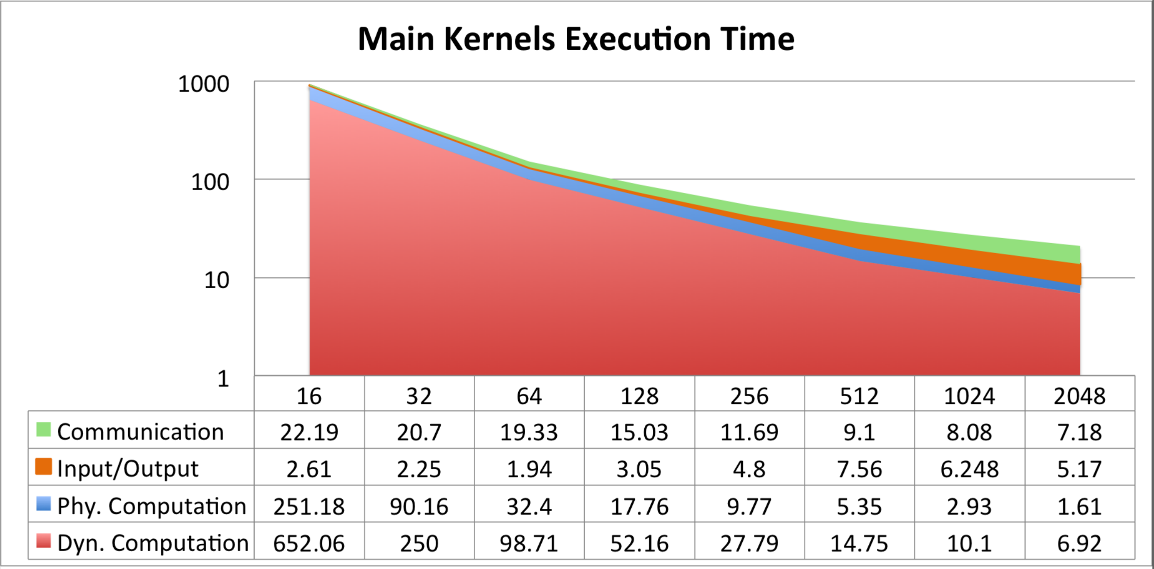


Figure 5:
Scalability analysis changing the I/O frequency: blue line for case 1, red line for case 2 and green line for case 3.

KERNEL'S SCALABILITY The last analysis has been conducted at kernels level. Figures 6 reports the wall-clock time of the fundamental operations (communication, I/O, physics, and dynamics computation) executed in the main kernels (the most expensive ones). Figure 7 reports the I/O, physics and dynamic computation speed-up compared with the ideal trend. The data have been collected from the YUTIMING file created at runtime. The I/O time did not scale with the number of processes since, for these tests, the number of I/O nodes has been kept constant. Figures 8 and 9 report a detailed analysis of the main kernels in terms of execution time and speed-up. Some kernels have a strange behaviour with a super-linear speed-up on 2048 cores and requires a more detailed analysis.



	16	32	64	128	256	512	1024	2048
🟩 Communication	22.19	20.7	19.33	15.03	11.69	9.1	8.08	7.18
🟧 Input/Output	2.61	2.25	1.94	3.05	4.8	7.56	6.248	5.17
🟦 Phy. Computation	251.18	90.16	32.4	17.76	9.77	5.35	2.93	1.61
🟥 Dyn. Computation	652.06	250	98.71	52.16	27.79	14.75	10.1	6.92

Figure 6:
Execution time of the main kernels grouped by the operation type.

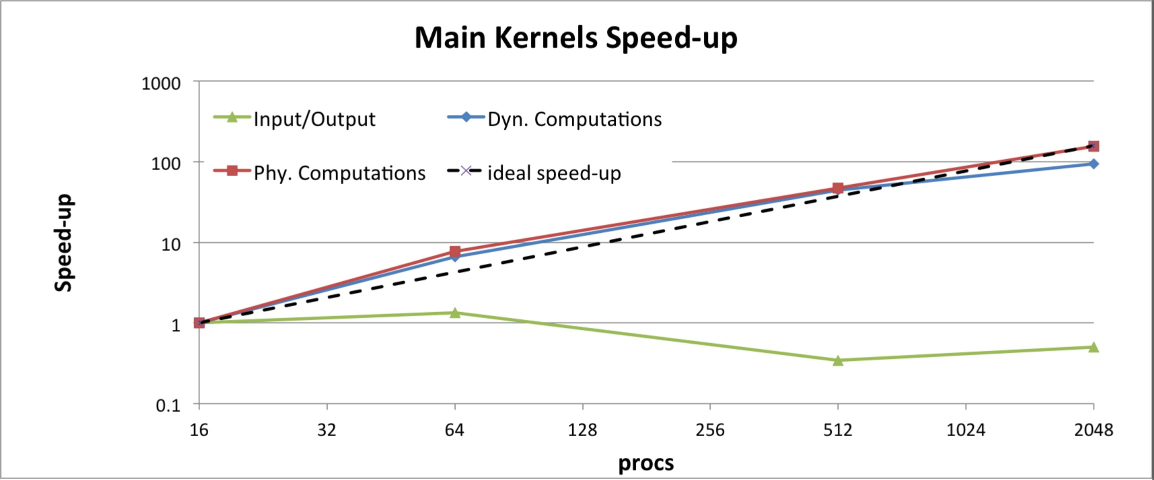
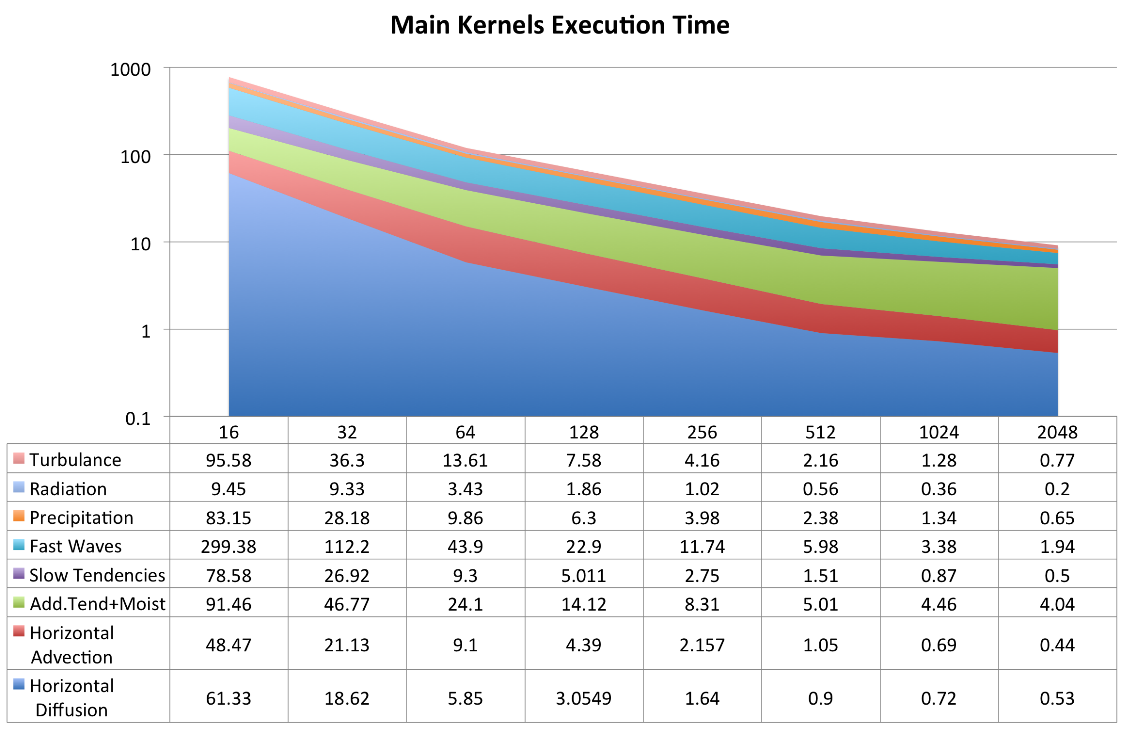


Figure 7:
Speed-up of the main kernels grouped by the operation type.



	16	32	64	128	256	512	1024	2048
🟥 Turbulance	95.58	36.3	13.61	7.58	4.16	2.16	1.28	0.77
🟦 Radiation	9.45	9.33	3.43	1.86	1.02	0.56	0.36	0.2
🟧 Precipitation	83.15	28.18	9.86	6.3	3.98	2.38	1.34	0.65
🟦 Fast Waves	299.38	112.2	43.9	22.9	11.74	5.98	3.38	1.94
🟪 Slow Tendencies	78.58	26.92	9.3	5.011	2.75	1.51	0.87	0.5
🟩 Add.Tend+Moist	91.46	46.77	24.1	14.12	8.31	5.01	4.46	4.04
🟥 Horizontal Advection	48.47	21.13	9.1	4.39	2.157	1.05	0.69	0.44
🟦 Horizontal Diffusion	61.33	18.62	5.85	3.0549	1.64	0.9	0.72	0.53

Figure 8:
Execution time of the main kernels.

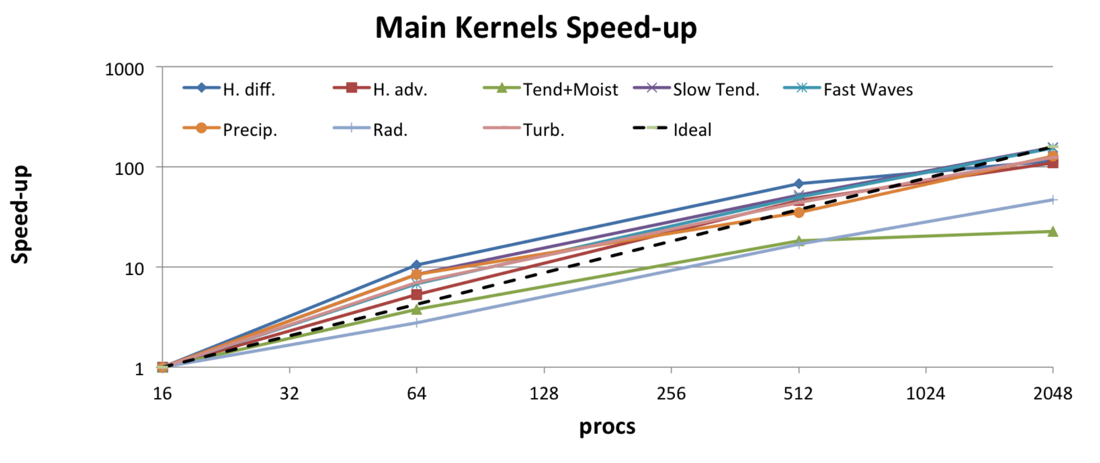


Figure 9:
Speed-up of the main kernels.

PROFILING

The code profiling is an important step for the performance analysis of a model since it allows to trace the call tree and to identify the most expensive routines (the bottlenecks during the code execution). We used the $gprof$ tool to profile the code.
Tables 1 and 2 show the analysis results filtered on the most expensive functions. In particular, for each of these, tables 1 and 2 report:

- the percentage on the total execution time
- the total time spent by the current function
- the number of times the routine is called
- the time spent for each call
- the function name

respectively with and without the O3 optimisation flag. The total execution time was reduced of about 82% just acting at compile time on the optimisation flag.
The reduction can be differently appreciated on the main routines (i.e. after the optimisation, the percentage of the $advection$ routine increases compared with the other ones). Figures 10 and 11 show the routines execution time with and without activating the compiler optimisation.

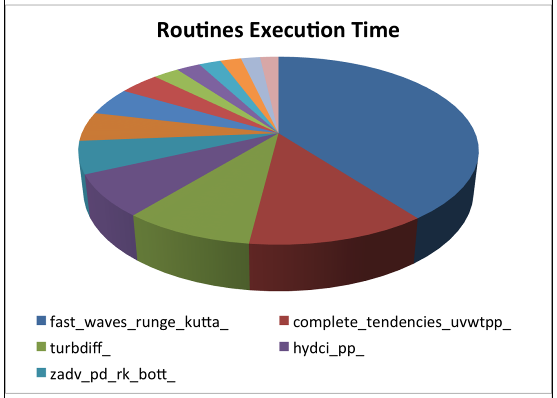


Figure 10:
Distribution of routines execution time activating compiler optimisation.

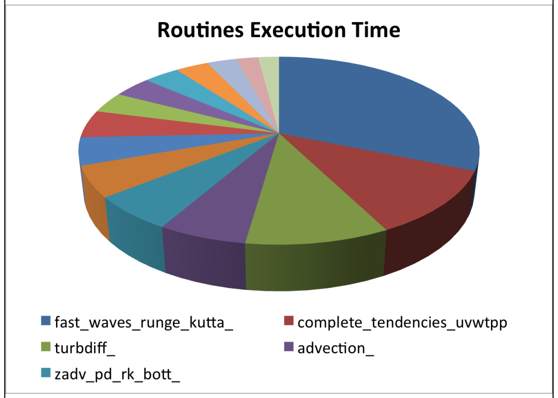


Figure 11:
Distribution of routines execution time without activating compiler optimisation.

Table 1

Gprof results: main kernels profiling activating compiler optimisation.

% time (self)	self seconds	calls	self s/call	name
28.59	194.85	12963	0.015031243	fast_waves_rk_mp_fast_waves_runge_kutta_
8.82	46.2	12963	0.00356399	src_slow_tendencies_rk_mp_complete_tendencies_uvwtpp_
6.43	44.78	4321	0.010363342	src_turbdiff_mp_turbdiff_
5.20	35.30	4321	0.008169405	src_gscp_mp_hydci_pp_
4.01	25.86	25926	0.000997454	numeric_utilities_rk_mp_zadv_pd_rk_bott_
3.62	19.59	4321	0.004533673	src_runge_kutta_mp_org_runge_kutta_
3.46	18.88	1037040	1.82057E-05	numeric_utilities_rk_mp_yadv_pd_rk_bott_
2.82	18.84	1037040	1.81671E-05	numeric_utilities_rk_mp_xadv_pd_rk_bott_
1.79	14.06	12963	0.001084625	src_advection_rk_mp_advection_
1.66	12.57	4321	0.002909049	src_slow_tendencies_rk_mp_implicit_vert_diffusion_uvwt_
1.53	9.26	4321	0.002143022	src_slow_tendencies_rk_mp_complete_tendencies_qvqcqi_tke_
1.46	8.29	518480	1.5989E-05	meteo_utilities_mp_satad_
1.33	7.23	4321	0.001673224	src_soil_multlay_mp_terra_multlay_
1.28	6.22	4321	0.001439482	src_slow_tendencies_rk_mp_complete_tendencies_init_

Table 2

Gprof results: main kernels profiling without activating compiler optimisation.

% time (self)	self seconds	calls	self s/call	name
25.36	983.90	12963	0.07590064	fast_waves_rk_mp_fast_waves_runge_kutta_
9.03	350.19	12963	0.02701458	src_slow_tendencies_rk_mp_complete_tendencies_uvwtpp_
7.91	306.74	4321	0.070988197	src_turbdiff_mp_turbdiff_
4.90	190.02	12963	0.014658644	src_advection_rk_mp_advection_
4.81	186.48	25926	0.007192779	numeric_utilities_rk_mp_zadv_pd_rk_bott_
4.30	166.67	4321	0.03857209	src_gscp_mp_hydci_pp_
3.85	149.30	82099	0.001818536	src_advection_rk_mp_adv_upwind3_lat
3.85	149.20	82099	0.001817318	src_advection_rk_mp_adv_upwind3_lon
2.96	114.96	1037040	0.000110854	numeric_utilities_rk_mp_xadv_pd_rk_bott_
2.96	114.65	1037040	0.000110555	numeric_utilities_rk_mp_yadv_pd_rk_bott_
2.79	108.43	4321	0.025093728	src_runge_kutta_mp_org_runge_kutta_
2.59	100.49	67200	0.001495387	src_radiation_mp_inv_th
2.31	89.76	4321	0.020772969	src_slow_tendencies_rk_mp_implicit_vert_diffusion_uvwt_
1.73	67.10	4321	0.015528813	src_relaxation_mp_sardass_
1.62	62.90	4321	0.014556816	src_slow_tendencies_rk_mp_complete_tendencies_qvqcqi_tke_

PERFORMANCE ANALYSIS

The qualitative analysis of the model performance has been carried out using Paraver [1], a tool for the performance analysis which allows to understand the behaviour of an application through a visual inspection. Paraver has been used to evaluate the communications weight and the load balancing. It allows to trace the most relevant performance indicators during the job execution and provides several views to analyse the trace. The performance indicators can be shown in two ways:

- *Timeline Display* - to visualise the temporal evolution of a performance indicator during the execution of the application (a line for each process)
- *Statistic Display* - to perform a statistical analysis along a group of processes, a time window or a kind of indicators

Both the views can be applied to a time window or to a group of processes selected by the user, to better focus the analysis. For example, our test have been executed on 256 processes, but only the first 32 processes have been reported in the figures.
Figure 12 and 13 report a *timeline display* of the trace for the CCLM model. Most of the execution time is dedicated to communications (orange lines in figure 12). Using the filters provided by the tool, the running (blue) and the idle (light blue) states have been highlighted (figure 13). Moreover, a zoom on the paraver trace can be done to better capture the status changes (figure 14). For the statistic analysis, the histogram utility has been used giving a numerical report of the filtered trace. Figure 15 reports the idle and running time for each process; the ratio between the running and the total time gives an estimation on how efficiently the computational cores have been used. Considering the average time over all the processes, we have an efficiency of 25%. Figure 16 reports the statistic information along all of the processes. Considering the ration between the average and the maximum value, we can have an estimation of the load balancing among the processes.

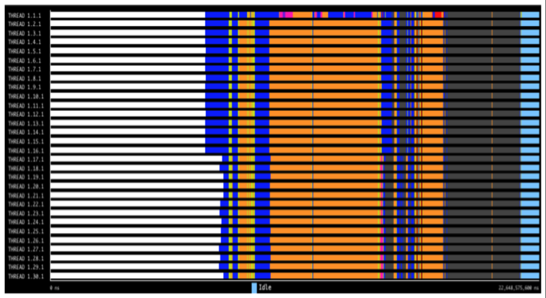


Figure 12:
Paraver tracer highlighting the different events at runtime.

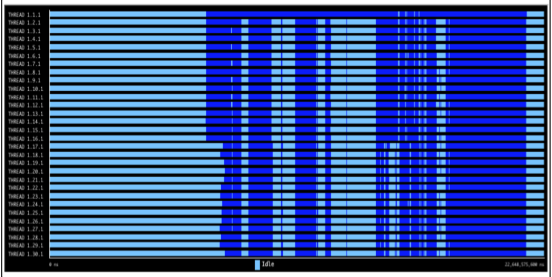
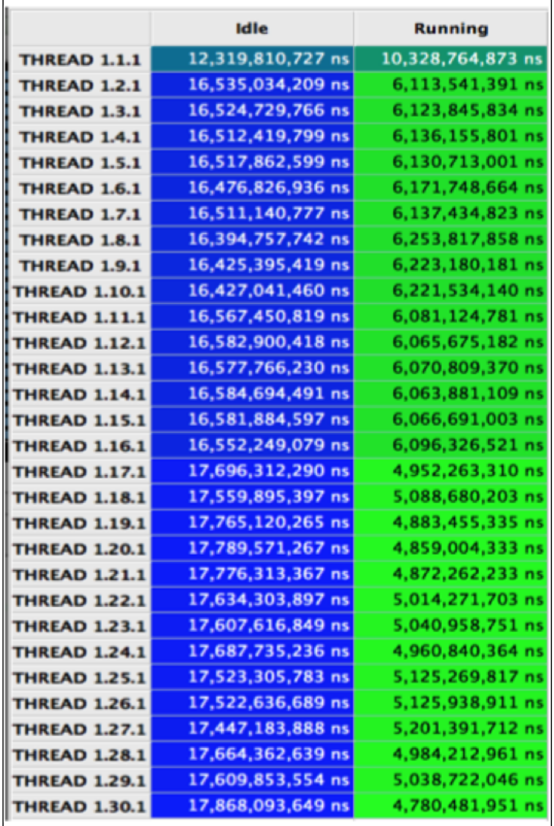


Figure 13:
Paraver tracer highlighting the comparison between the running time and the idle time.

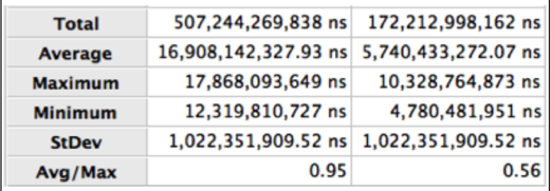
ROOFLINE MODEL

The roofline model [3] allows to measure and compare the performance of one or more computational kernels executed on different hardware architectures. It aims at providing an insight into the performance on many core architecture; it does not need to be a perfect performance model but just insightful. The basic idea



	Idle	Running
THREAD 1.1.1	12,319,810,727 ns	10,328,764,873 ns
THREAD 1.2.1	16,535,034,209 ns	6,113,541,391 ns
THREAD 1.3.1	16,524,729,766 ns	6,123,845,834 ns
THREAD 1.4.1	16,512,419,799 ns	6,136,155,801 ns
THREAD 1.5.1	16,517,862,599 ns	6,130,713,001 ns
THREAD 1.6.1	16,476,826,936 ns	6,171,748,664 ns
THREAD 1.7.1	16,511,140,777 ns	6,137,434,823 ns
THREAD 1.8.1	16,394,757,742 ns	6,253,817,858 ns
THREAD 1.9.1	16,425,395,419 ns	6,223,180,181 ns
THREAD 1.10.1	16,427,041,460 ns	6,221,534,140 ns
THREAD 1.11.1	16,567,450,819 ns	6,081,124,781 ns
THREAD 1.12.1	16,582,900,418 ns	6,065,675,182 ns
THREAD 1.13.1	16,577,766,230 ns	6,070,809,370 ns
THREAD 1.14.1	16,584,694,491 ns	6,063,881,109 ns
THREAD 1.15.1	16,581,884,597 ns	6,066,691,003 ns
THREAD 1.16.1	16,552,249,079 ns	6,096,326,521 ns
THREAD 1.17.1	17,696,312,290 ns	4,952,263,310 ns
THREAD 1.18.1	17,559,895,397 ns	5,088,680,203 ns
THREAD 1.19.1	17,765,120,265 ns	4,883,455,335 ns
THREAD 1.20.1	17,789,571,267 ns	4,859,004,333 ns
THREAD 1.21.1	17,776,313,367 ns	4,872,262,233 ns
THREAD 1.22.1	17,634,303,897 ns	5,014,271,703 ns
THREAD 1.23.1	17,607,616,849 ns	5,040,958,751 ns
THREAD 1.24.1	17,687,735,236 ns	4,960,840,364 ns
THREAD 1.25.1	17,523,305,783 ns	5,125,269,817 ns
THREAD 1.26.1	17,522,636,689 ns	5,125,938,911 ns
THREAD 1.27.1	17,447,183,888 ns	5,201,391,712 ns
THREAD 1.28.1	17,664,362,639 ns	4,984,212,961 ns
THREAD 1.29.1	17,609,853,554 ns	5,038,722,046 ns
THREAD 1.30.1	17,868,093,649 ns	4,780,481,951 ns

Figure 15:
Histogram related to the zoomed paraver tracer.



Total	507,244,269,838 ns	172,212,998,162 ns
Average	16,908,142,327.93 ns	5,740,433,272.07 ns
Maximum	17,868,093,649 ns	10,328,764,873 ns
Minimum	12,319,810,727 ns	4,780,481,951 ns
StDev	1,022,351,909.52 ns	1,022,351,909.52 ns
Avg/Max	0.95	0.56

Figure 16:
Statistic values related to the paraver tracer.

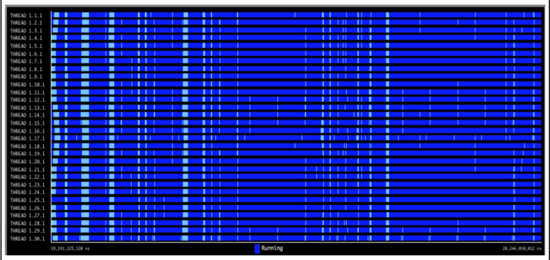


Figure 14:
Zoom on the paraver tracer highlighting the comparison between the running time and the idle time.

behind the roofline model is that an application is limited by two main factors: the peak floating point operations per seconds and the maximum memory bandwidth required to execute a given number of operations per seconds. Hence, to build the roofline model the following characteristics of the architecture must be taken into account:

- max Floating Point Operations per second;
- max Memory Bandwidth.

The first is the maximum number of floating point operations that a core can execute and often, for multicore chips, it is evaluated as the collective peak performance of all the cores on the chip. The second number refers to the Max Memory Bandwidth (MMB) of the architecture, i.e. the maximum speed at which you can transfer the data from/to the memory.
The other data we need are related to the computational kernel that we want to evaluate. First of all we need the total number of floating point operations performed by the kernel during a run, measured in GFlops. Another important factor to take into account is the Arithmetic Intensity (AI), which is the number of floating point operations performed per byte transferred to/from the memory. It is measured in (Floating Point Operations)/Byte and it can be considered as a measure of the density of all floating-point operations performed related to the total number of bytes of data transferred from DRAM.
The objective of our analysis was the classification of the most expensive routines using the roofline model. To collect the number of

Flops and the number of byte trasferred from the DRAM we used the hardware counters by means of the PAPI library. Actually, since the counter for the data volume transferred to/form the DRAM is not available on the architecture, we used the number of last level cache misses counter considering that for each cache miss a transfer of a cache line occurs. On the Athena cluster the cache line is 64 byte wide. We measured the flops and the last level cache misses for each routine instrumenting the code with the following PAPI calls:

- flops_counting_start;
- flops_counting_stop;
- cache_misses_counting_start;
- cache_misses_counting_stop.

For each routine, the flops of the process having the highest elapsed time have been considered, while the cache misses have been extracted from the first occurrence of the routine executed by the process with the highest elapsed time. The following routines have been considered:

- fast_waves_runge_kutta_
- complete_tendencies_uvwtpp_
- implicit_vert_diffusion_uvwt_
- complete_tendencies_qwqcqi_tke_
- org_runge_kutta_

Starting from the flops and the cache misses values, we can define the Arithmetic Intensity and the Gflops for each routine and hence place it on the roofline chart. The coordinates of the points representing the routines into the roofline model can be evaluated as follows:

$$x = \frac{Flops}{Byte} = \frac{Flops}{CacheMisses * 64} \quad (6)$$

$$y = \frac{GFlops}{Sec} \quad (7)$$

Table 3 shows the collected data, while the graph locates the analysed routines into the roofline model (figure 17).
The three routines at the left of the ridge point cannot be improved over a given threshold due to the bandwidth limit, while a solution could be to increase the AI to get them as close as possible to the ridge point. For the routines at the right of the ridge point, since they are located under the "no vectorisation" line, their vectorisation should be improved to better exploit the SSE or AVX1 instruction sets and to decrease the Floating Point operations execution time.

Table 3
Data collected to trace the roofline model.

Routine	Cache misses	Flops	Elapsed time
fast_waves_runge_kutta	19530	546864393	0.215878
complete_tendencies_uvwtpp	22477	1893552	0.061917
implicit_vert_diffusion_uvwt	8753	1802797	0.068709
complete_tendencies_qvqcqi_tke	17930	1788636	0.075288
org_runge_kutta	308196	743732162	0.839186

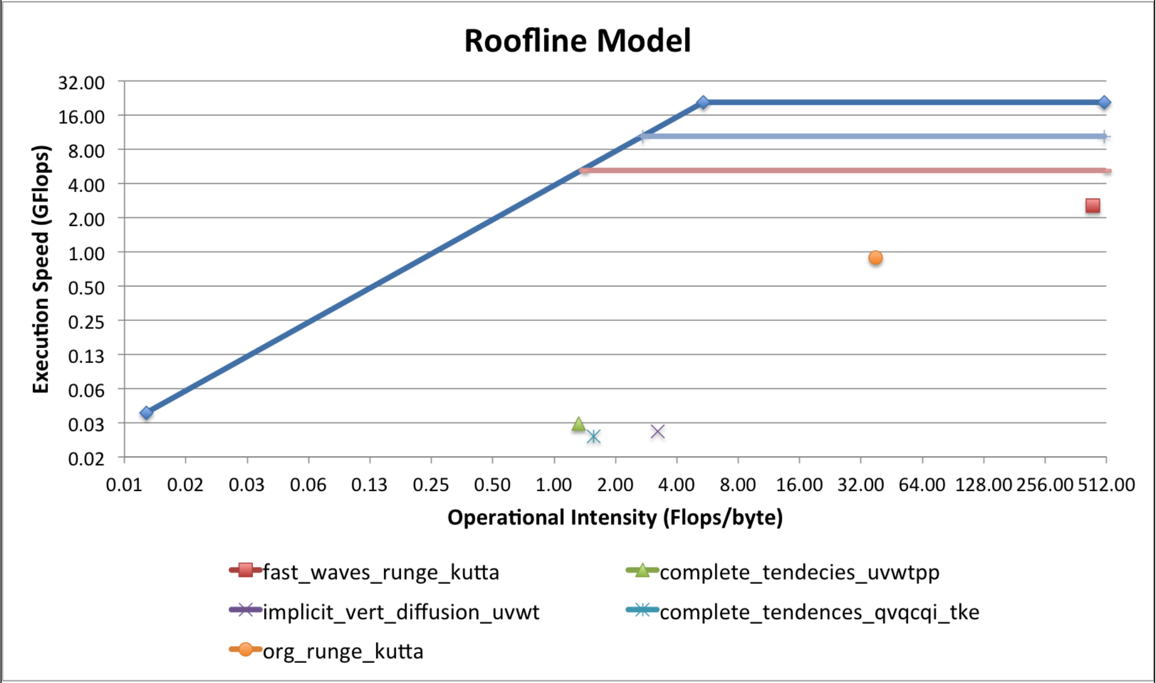


Figure 17:
Roofline model for the main CCLM kernels.

Bibliography

[1] Barcelona Supercomputing Centre. http://www.bsc.es/computer-sciences/performance-tools/paraver. Technical report.

[2] U. Schattler, G. Doms, and C. Schraff. A description of the nonhydrostatic regional cosmo-model. Technical report, Consortium for Small-Scale Modelling, 2013.

[3] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

© **Centro Euro-Mediterraneo sui Cambiamenti Climatici 2015**

Visit **www.cmcc.it** for information on our activities and publications.

The Euro-Mediteranean Centre on Climate Change is a Ltd Company with its registered office and administration in Lecce and local units in Bologna, Venice, Capua, Sassari, Viterbo, Benevento and Milan. The society doesn't pursue profitable ends and aims to realize and manage the Centre, its promotion, and research coordination and different scientific and applied activities in the field of climate change study.

cmcc
Centro Euro-Mediterraneo sui Cambiamenti Climatici