Porting and optimizing Meso-NH to AMD MI250X GPUs

Naïma ALAOUI ISMAILI†
Département calcul intensif
CINES
Montpellier France
alaoui@cines.fr

Juan ESCOBAR
Laboratoire d'Aérologie
CNRS/Université de Toulouse
Toulouse France
juan.escobar@aero.obs-mip.fr

Philippe WAUTELET
Laboratoire d'Aérologie
CNRS/Université de Toulouse
Toulouse France
philippe.wautelet@aero.obs-mip.fr

Pascal VEZOLLE
HPE
Montpellier
pascal.vezolle@hpe.com

Gabriel HAUTREUX
Département calcul intensif
CINES
Montpellier France
hautreux@cines.fr

ABSTRACT

This paper presents the results of our efforts to port Meso-NH, an atmospheric non-hydrostatic research model, to AMD MI250X GPUs using OpenACC on the ADASTRA Machine, a technology similar to the Frontier system [1]. Meso-NH is a versatile model that covers a wide range of resolutions from synoptic to turbulent scales, and is designed for studies of physics and chemistry.

Numerical simulation of the atmosphere is crucial for understanding and predicting weather and climate extremes. Current numerical weather prediction codes are limited to specific resolutions on global and regional scales. The Meso-NH code, however, tackles scales and complexities beyond what is typically used in operational forecasting.

We collaborated with GENCI, CINES, HPE, and AMD on the "progress contract," for ADASTRA machine aiming to achieve simulations at hectometric resolution for recent storms in the Atlantic and Mediterranean regions, characterized by extreme wind gusts. This higher resolution allows us to explicitly represent a cascade of scales, from the storm core (>100 km) to the deep and shallow convective circulations behind the gusts (<1 km). The successful porting of Meso-NH to AMD MI250X GPUs made this numerical achievement possible.

The paper focuses on the challenges faced during the porting process, optimization strategies, and the lessons learned throughout the project. Additionally, we share the current performance results from relevant benchmark problems. A comparative study highlights the performance and energy consumption aspects of a fraction of the code that dominates the computation time. Notably, we achieved a speed-up of 3.5 times compared to computation on AMD- Genoa processors.

The results of this porting effort open up new possibilities for atmospheric simulations at hectometric resolutions, enhancing the accuracy and sophistication of weather phenomena representation. The collaboration between different institutions has paved the way for advanced meteorological research, contributing to the ongoing progress in the field.

KEYWORDS

Meso-NH, HPC, AMD, GPU, OpenACC, optimisation

ACM Reference format:

Naima ALAOUI ISMAILI, Juan ESCOBAR, Philippe WAUTELET, Gabriel HAUTREUX and Pascal VEZOLLE 2023. Porting and optimizing Meso-NH to AMD MI250X GPUs. In *Proceedings of ACM Woodstock conference (WOODSTOCK'23). ACM, Denver, USA, 6 pages.*

1 Introduction

Meso-NH [2] is a French mesoscale meteorological research model, initially developed by the Centre National de Recherches Météorologiques (CNRM – CNRS/ Météo-France) and the Laboratoire d'Aérologie (LA – UPS/CNRS).

It operates as a grid-point-limited area model based on a non-hydrostatic system of equations. To account for the Earth's sphericity, the equations are formulated on the conformal plane. The model ensures anelastic continuity by solving an elliptic equation with high accuracy to determine pressure perturbations. In its early stages, the Richardson iterative method was used, and later, a more efficient method following Skamarock et al. [3] was developed, employing a conjugate residual algorithm accelerated by a flat Laplacian preconditioner, and has been parallelized both vertically and horizontally.

Meso-NH is maintained by computer and research scientists from LA and CNRM. The code is written in Fortran 90. Running scripts are in shell and use make files. Much of the

Meso-NH model has been parallel since 1999 [4], for distributed memory clusters. Significant portions of Meso-NH have been parallelized, employing 2-D domain decomposition, where the physical domain is divided into horizontal subdomains in the x and y directions. Communication between multiple processes is achieved through the Message Passing Interface (MPI).

In recent years, efforts have been made to port Meso-NH to NVIDIA GPUs, utilizing OpenACC directives for GPU-based parallelization. NVIDIA's robust support for OpenACC in its software stack made this transition smoother. Additionally, HPE CRAY FORTRAN compiler also provided OpenACC GPU offloading support, enabling the compilation of Meso-NH on the new AMD MI250X GPU-based ADASTRA system.

This work outlines the initial experiences in porting Meso-NH, originally running on AMD GPUs on the ADASTRA system with the HPE/CRAY environment [5]. The paper discusses the challenges encountered during the porting and optimization process, transitioning from HPC systems based on NVIDIA GPUs to AMD GPU-based systems on ADASTRA. The focus then shifts to presenting case studies using Meso-NH with GPUs, showcasing the achieved high performance on AMD MI250X GPU-based systems, along with the energy efficiency results obtained. The paper concludes by summarizing the key findings.

2 The porting effort

Since the code had already been ported to NVIDIA GPUs, our strategy was to first compile the application on the GPUs, validate the existing OpenACC directives in the code, and then proceed to the performance optimization phase. Second, port the non-accelerated part of the code to the GPU.

Compiling the Meso-NH OpenACC-based code using the CRAY compiler for AMD GPUs was relatively straightforward. There were a few places where the compiler indicated that the existing OpenACC directives were missing required fields, even though we are reasonably confident that the OpenACC specification lists them as optional and the NVIDIA compiler did not have an issue. Nevertheless, the error messages were clear and the required changes minimal.

After successfully compiling and conducting regression testing on the AMD GPU-based systems, an extensive optimization effort was initiated to compare the relative performance on AMD MI250X GPU-based nodes on the ADASTRA system with that on NVIDIA V100 GPU-based nodes on another system (MARCONI-100/CINECA and Jean-Zay/IDRIS).

During the porting phase, two main challenges emerged. The first challenge was regarding performance issues, and the second pertained to ensuring bit-reproducibility of numerical results between the GPU and CPU versions.

Optimizing a Fortran scientific application like Meso-NH in a manner that is both portable and productive proved to be quite challenging. A significant aspect that posed difficulties was Meso-NH's extensive use of array syntax. This specific feature caused difficulties when using certain compilers, such as the Fortran CRAY compiler with OpenACC offloading.

For benchmarking purposes, two representative cases were selected: one with a grid size of 256x256 and the other with a grid size of 512x512. These grid sizes, though modest, are physically meaningful for simulations. In this section, we will primarily focus on the difficulties encountered during the porting process, while the results related to pure compute performance will be discussed in the following section.

2.1 Challenge 1: False recurrence

During the porting process, we encountered a notable bug in the Fortran CRAY compiler (versions including *cce/15.01*) that led to the detection of false recurrence between variables in OpenACC, significantly impacting parallelization. To work around this issue, we found a solution by adding "present(...)" directives to the *"!$acc kernels*" directives. However, this approach caused a conflict with the Nvidia "nvfortran" compiler, which detected an error due to double declarations. The variables declared in the *"!$acc kernels present()"* directive were already declared in a *"!$acc data present(...)"* directive, encompassing all kernels. To address this situation, we introduced a *"present_cr"* macro and incorporated compiler flags to enable the bypass only for the CRAY compiler *(flags -DMNH_COMPILER_CCE)*. Bellow an extract from the code highlighting illustrates the false recurrence problem. Additionally, in other sections of the code, we encountered the same recurrence problem, where the compiler seemed to ignore the independent clause in the *"!$acc loop"* directive. To resolve this, we found that the addition of the *"!DIR$CONCURRENT"* directive proved effective.

* * *

```
ZDMQ(:,:,IKB-1) = &
SIGN( (MIN( ABS(ZDMQ(:,:,IKB-1)),
2.0*(PSRC(:,:,IKB-1) - &
MIN(PSRC(:,:,IKE-1),PSRC(:,:,IKB-
1),PSRC(:,:,IKB))), &
2.0*(MAX(PSRC(:,:,IKE-1),PSRC(:,:,IKB-
1),PSRC(:,:,IKB)) - &
PSRC(:,:,IKB-1)) )), ZDMQ(:,:,IKB-1) )
!$acc end kernels
!$acc kernels present_cr(ZDMQ,PSRC) ZDMQ(:,:,IKE+1) = &
SIGN( (MIN( ABS(ZDMQ(:,:,IKE+1)),
2.0*(PSRC(:,:,IKE+1) - & MIN(PSRC(:,:,IKE),PSRC(:,:,IKE+1),PSRC(:,:,I
KB+1))), &
2.0*(MAX(PSRC(:,:,IKE),PSRC(:,:,IKE+1),PS
RC(:,:,IKB+1)) - &
PSRC(:,:,IKE+1)) )), ZDMQ(:,:,IKE+1) )
!$acc end kernels
```

* * *

In all the other parts of the code where the same recurrence problem was found, the compiler ignored the independent clause in the *"!$acc loop"* directive, the addition of the *!DIR$CONCURRENT* directive solved the problem, as shown below.

* * *

```
!$acc kernels
DO JK=IKTB,IKTE
!DIR$CONCURRENT
!$acc loop independent collapse(2) private( ZCOND, ZQ1, ZTEMP )
            DO JJ=KJB,KJE
            DO JI=KIB,KIE
ZTEMP = PT(JI,JJ,JK)
            ENDDO
            ENDDO
ENDDO
```

* * *

2.2 Challenge 2: extra copy

Another significant challenge arose when pointers were used in the code, leading to potential aliasing between the pointers and the array syntax. The loops were not recognized as parallelizable, resulting in performance issues. However, when we used "allocatable" instead of pointers, the compiler acknowledged that there was no aliasing, leading to improved performance.

In cases where array syntax was used with pointers, we found that a temporary array was created. For example, "*A=A+B*" became "*T=A+B*" followed by "*A+T*". Consequently, two separate kernel statements were executed. The extra copy observed in the code was attributed to this temporary array. Although we managed to address the copy, we still faced the challenge of executing two kernels. To mitigate this issue, we reached out to HPE support and devised a workaround involving the addition of specific directives. We introduced the necessary directives in certain situations, such as adding "present(...)" directives to *the "!$acc kernels"* directive, which helped resolve the performance problem due to the extra copy.

As part of our efforts to document these performance problems comprehensively, we created a reproducer that consolidates various scenarios, encompassing array syntax and loops with or without the "present" clause. This reproducer serves as a valuable resource in understanding and addressing these challenges efficiently. The code below shows a reproducer extract of different types of operations.

* * *

```
#ifdef Doconcurrent
do concurrent(i=1:NX,j=1:NY,k=1:NZ)
TZ_U(i,j,k) = TZ_U(i,j,k) + TX_U(i,j,k) + 2.0 * TY_U(i,j,k)
#elif defined(Arraysyntax)
TZ_U(:,:,:) = TZ_U(:,:,:) + TX_U(:,:,:) + 2.0 * TY_U(:,:,:)
#elif defined(Do loop)
!$acc loop independent collapse(3) do i=1,NX; do j=1,NY; do k=1,NZ
TZ_U(i,j,k) = TZ_U(i,j,k) + TX_U(i,j,k) + 2.0 * TY_U(i,j,k)
#elif defined(Do loop with collapse)
!$acc loop
!dir$ collapse(i,j,k) (Do loo with collapse) do i=1,NX; do j=1,NY; do k=1,NZ
TZ_U(i,j,k) = TZ_U(i,j,k) + TX_U(i,j,k) + 2.0 * TY_U(i,j,k)
```

* * *

In table 1, we present the performance results, the best performance is obtained with array syntax by adding the present clause in the acc kernel directive in both cases either with or without pointer.

GPU ADASTRA	Array declaration			
	Pointer Fortran		No Pointer	
	!$acc kernels	*!$acc kernels present*	*!$acc kernels*	*!$acc kernels present*
Do concurrent	0.304	0.303	0.297	0.297
Array syntax	9.18	0.179	0.296	0.200
Do loop *!dir$ collapse*	0.203	0.204	0.209	0.209
Do loop *independent collapse*	-	8.779	8.771	8.773

Table 1: Performance results with different types of operations

2.3 Challenge 3: Flush-to-zero

Ensuring bit-reproducibility of CPU/GPU results is crucial in scientific applications like Meso-NH. To tackle this crucial aspect, two internal tools have been developed within Meso-NH.

The Bit-Reproducibility Library: This library plays a key role in maintaining bit-reproducibility by recoding intrinsic functions such as log, exp, power, and others. The functions are initially recoded in C++ +OpenACC and later translated to OpenMP. To interface with elemental functions, a Fortran interface is used. Leveraging this library enables users to obtain bit-reproducible results, which is essential for the reliability and consistency of simulations.

MPPDB_CHECK: Functioning as a debugging tool, MPPDB_CHECK launches two executables simultaneously - an executable for the CPU version and another for the GPU version using the same data. It then compares all the intermediate arrays utilized during the calculations. This tool proves highly beneficial when dealing with non-bit-reproducible results, as it helps identify the specific arrays responsible for the discrepancies, thereby facilitating the resolution of the underlying issues.

However, as the number of iterations increased, we discovered that the results were no longer bit- reproducible after some iterations. This inconsistency was attributed to the "flush-to-zero" problem.

The "flush-to-zero" setting is a hardware feature that resets any computed floating-point result to zero if the result would otherwise be subnormal. Generally, compilers set this mode by default or if options like "-fast" are utilized. In "flush-to-zero" mode, the hardware assumes that operands are either normalized floating-point values or NaN or Infinity values—all of which are natively handled by the hardware.

Within the IEEE 754 specification, the term "tiny non-zero" refers to "subnormal" numbers, which serve as representations to fill the gap between zero and the smallest

normalized number. However, the behavior of these numbers differs slightly, and using them can lead to a decrease in performance. The threshold value corresponds to the smallest normalized number for the specific data width being used (e.g., $\sim 2.225.10^{-308}$ for float64).

By default, CCE sets "abrupt underflow" floating- point handling, meaning that whenever arithmetic operations would produce a "tiny non-zero" result, it is replaced with a zero of the same signs. For instance, the expected result of $-1.7236.10^{-308}$ is considered tiny and replaced by -0. Conversely, another example value, RR3 = $-8.1459090792699401.10^{-300}$, produces a non-tiny/normal result and is not flushed to zero.

In fact, values below the min (2-1022= $\sim 2.225.10^{-308}$), and in particular values called subnormal or "au tiny non-zero" in the IEE 754 spec, are flushed to zero, in our case on the CPU and not on the GPU.

To solve this problem, it was necessary to apply a filter to very small values, setting them to zero, the code below shows an extract of code with the filter applied.

* * *

```
ELEMENTAL FUNCTION BR_FZ(PVAL)
!$acc routine seq
REAL, INTENT(IN) :: PVAL REAL      :: BR_FZ
IF ( ABS(PVAL) < 2.225*(10.0**(-308)) ) THEN
BR_FZ=sign(0.0,PVAL) ELSE
BR_FZ=PVAL ENDIF
END FUNCTION BR_FZ
```

* * *

3 Performance results

All benchmarks presented in the following were run on the AMD MI250X GPU-based ADASTRA system (that hosts 64 cores Trento CPUs coupled to 8 AMD GPUs). The performance measurements for the 256x256 grid test case are presented in Table 2.

Hector 256x256 (MG)		NNODES(GPU)
		1(8)
ADASTRA-GPU	Number of MPI	16
	Time(s)	106.0
ADASTRA-CPU	Number of MPI	64(3cpu per task)
	Time(s)	386.2
Speed up		**3.643**

Table 2: Performance results and speed up on ADASTRA GPU/CPU for the test case 256x256

These measurements were obtained with an acceleration factor of 3.643 using the multigrid solver (MG) version for 100-time steps (to ensure fair performance comparison in all tested cases, all runs based on Meso-NH were using *Prg-Env cray/8,3,3, cce/15,0,1* compiler and their own implementation of MPICH. The results show the speedup achieved on the GPU/CPU system.

Table 3 summarizes the various performance measurements on the 512x512 grid test case. The multigrid solver (MG) version performed well, with an acceleration factor of 5.5, 3.5 and 3.1 for cases 1, 2 and 4 nodes respectively.

Hector 512x512		NNODES(GPU)		
		1(8)	2(16)	4(32)
ADASTRA-GPU (MG)	Number of MPI	16	16	64
	Time(s)	252.3	184.4	106.0
ADASTRA-GPU (FFT)	Number of MPI	16	32	64
	Time(s)	300.3	186.0	121.0
ADASTRA-CPU (MG)	Number of MPI	64	256	256
	Time(s)	1377.4	642.6	330.2
ADASTRA-CPU (FFT)	Number of MPI			
	Time(s)	861.2	398.7	187.1
Speed up (MG)		**5.5**	**3.5**	**3.1**
Speed up (FFT)		**2.9**	**2.1**	**1.5**

Table 3: Performance results and speed up on ADASTRA GPU/CPU for the test case 512x512: comparison FFT and MultiGrid solver

Performance on the 512x512 grid test case (Meso-NH-MG) shows that execution on 1 GPU node is faster than on 4 AMD GENOA CPU nodes (192 cores @ 2.4 GHz per node), with an execution time of 252.3 on 1 GPU node versus 330.3 seconds on 4 CPU nodes. We find a very good scalability of the FFT solver version on the ADASTRA-CPU partition, with an execution time of 861.2 seconds for 1 node and 187 seconds for 4 nodes. The absence of any constraint on the number of MPI ranks for the FFT solver, we were able to utilize fully the resources available to the GENOA node resources and cache bandwidth.

4 Energy consumption

4.1 Impact of the CPU turbo mode usage

In the energy studies, the team investigated the impact of activating the turbo mode on the CPUs of the Trento nodes in the ADASTRA GPU partition. The tests were conducted by running benchmarks with and without the turbo mode activated (via SLURM prolog). Figure 1 illustrates the impact of CPU turbo mode usage.

The findings revealed that energy consumption on a AMD 9654 96-cores GENOA bi-socket node is three times greater than on a AMD MI250X GPU node. Moreover, the performance measurements for the 256x256 grid test case, as presented in Table 2, demonstrated a significant impact of turbo mode on CPU time to solution for the Meso-NH application. In this specific test case, using turbo mode on the GPU resulted in lower energy consumption.

Figure 1 shows that energy consumption on AMD GENOA node is 3 times greater than on a AMD MI250X GPU node, and the performance measurements presented in Figure 1. for the test case of the 256x256 grid on a single node. The impact of turbo on CPU for time to solution is significant for Meso-NH application. This test case consumes less energy using turbo mode on GPU.

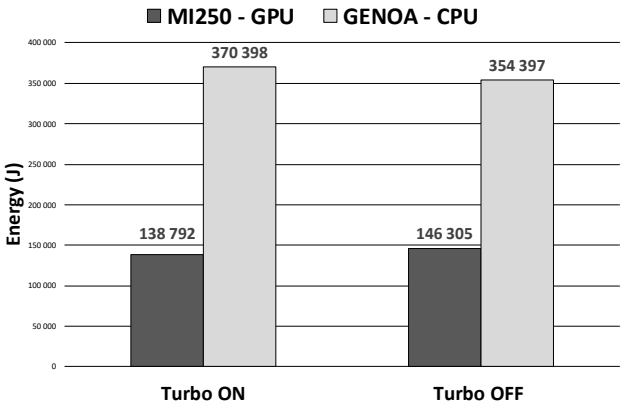

Figure 1: Impact of the CPU turbo mode usage on energy consumption.

4.2 Frequency capping studies

Frequency capping studies were performed using the Energy Reporting Tool in Slurm (ERIS). This tool allowed the team to control the CPU frequency, ensuring that the exact same node selection is used for all the runs performed in order to minimize the noise in the output results for a reliable impact investigation on energy consumption for the Meso-NH application. The study is performed by setting the maximum frequency from 0.8GHz to 1.7GHz.

Detailed results of these frequency capping studies were provided in Figure 2(a) and 2(b).

This study aimed at identifying the most suitable frequency to ensure efficient utilization of available resources and to optimize energy consumption with limited performance impact. The ERIS script played a vital role in managing and analyzing the CPU frequency, contributing to a comprehensive understanding of energy consumption patterns and strategies for achieving energy efficiency.

The Meso-NH application was run on Grid 512 and on 4 nodes, we can define the best frequency values for this application. This allows us to get the optimum power consumption for the workload.

Figure 2 shows a 1.2GHz GPU frequency sweet spot where relative energy is lowest for our global workload, although at the cost of a 6% drop in performance, this frequency is still an interesting energy improvement.

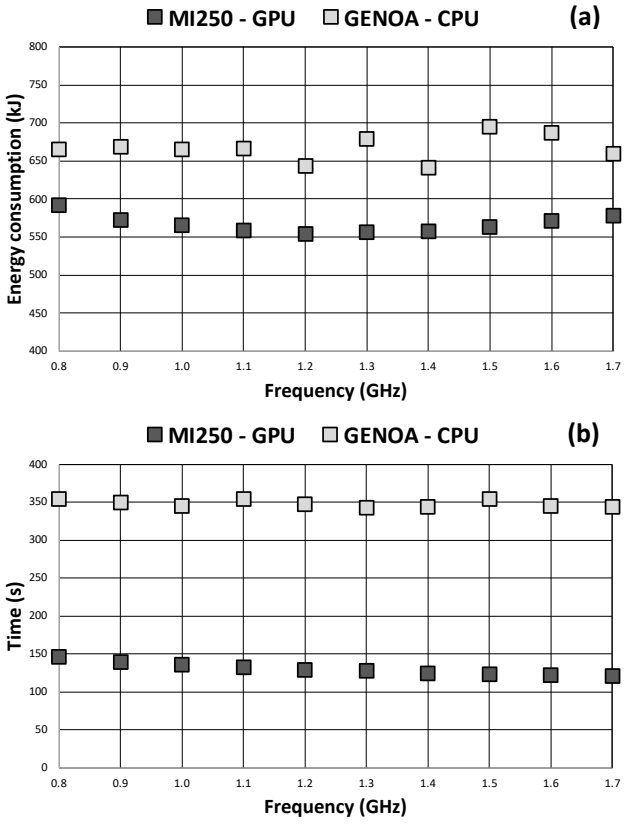

Figure 2: Results of energy consumption (a) and time performance (b) versus frequency variation

5 Summary and conclusions

In this paper, we presented the successful porting of the Meso-NH application to the ADASTRA machine as part of the Progress Contract. The porting process involved compiling the application in the CRAY environment and analyzing its performance. We encountered various challenges during the porting, especially related to bit- reproducibility, which required careful attention and code adaptations to achieve both accurate numerical results and optimal performance. During the fifteen-month project, we extended the accelerated part and ported the FFT solver, enabling us to conduct scientific simulations using 128 GPU-AMD nodes at a grid resolution of 1.5 billion points with 100m resolution (Giga-LES). These simulations were instrumental in studying recent storms in the Atlantic and Mediterranean, including extreme wind gusts. The initial results obtained on the GPU-AMD platform demonstrated a remarkable factor of 3.3 speedup while significantly reducing energy consumption compared to the best current CPU computing architectures, specifically ADASTRA's Genoa CPU partition, and were presented during the inauguration of the ADASTRA supercomputer on May 04, 2023 at CINES.

ACKNOWLEDGMENTS

This work was performed using HPC resources from GENCI–CINES, a big thank you goes to them.

We are deeply grateful to all those who played a role in the success of this project. We would like to thank Juan ESCOBAR and Philippe WAUTELET for their guidance and help making this porting on ADASTRA possible.

Thank you to AMD and HPE for their invaluable input throughout the porting process. Their insights and expertise were instrumental in shaping the direction of this project.

REFERENCES

[1] Frontier web page. https://www.olcf.ornl.gov/frontier/

[2] J.P. Lafore, J. Stein, N. Asencio, P. Bougeault, V. Ducrocq, J. Duron, C. Fischer, P. Héreil, P. Mascart, V. Masson, J.P. Pinty, J.L. Redelsperger, E.Richard, The Meso-NH Atmospheric Simulation System. Part I: Adiabatic formulation and control simulations. Scientific objectives and experimental design, Ann. Geophys, 16(1998), 90-109.

[3] W.C. Kamarock and J.B. Klemp, A time-split nonhydrostatic atmospheric model for weather research and forecasting applications, J. Comput. Phys., 227 (2008), 3465–3485.

[4] Meso-NH web page. http://mesonh.aero.obs-mip.fr/

[5] J. Escobar, P. Wautelet, N. Alaoui Ismaili, P. Vezolle and P.E. Bernard, portage de Méso-Nh sur Super-Calculateur avec OpenACC sur GPU NVIDIA et AMD.
web page http://www.meteo.fr/cic/meetings/2022/AMA/programme.pdf