D4.5

First report on the identified bottlenecks for an efficient usage of Nordic ESMs on EuroHPC

Report on the bottlenecks that would hinder the efficient usage of Nordic ESMs on EuroHPC and possible remediation actions (I/Os, adding GPU support, etc.) with clear information on costs in terms of manpower.

Task leader: Alok Gupta
Project manager: Anne Fouilloux
Work Package leader: Jean Iaquinta
Internal reviewer: Oskar Landgren

Introduction

Grand challenges in Earth System Modelling mean that always more resources are needed (increased resolution, longer time periods considered, higher complexity of the bio-physico-chemical processes represented). In the past, advances in computer chips were sufficient to achieve significant speedups in real applications such as Earth System Modelling. It took about 20 years to make the models run about x1000 faster just because the hardware (computers) and software (algorithms) improved (see ExtremeEarth proposal), this period was often referred to as the “golden age” of numerical modelling. However, such a speedup factor is not achievable any more, even if we were ready to wait for several decades. The famous Moore's law came to an end already about 5-10 years ago (Khan, H.N., Hounshell, D.A. & Fuchs, E.R.H. Science and research policy at the end of Moore’s law. *Nat Electron* 1, 14−21 (2018). https://doi.org/10.1038/s41928-017-0005-9), and accelerating ESM simulations now requires a radical redesign of the current codes.
The newer/bigger machines are extremely expensive to conceive, build, operate and maintain (then dismantle/recycle when possible), therefore although they are needed

for Earth System Modelling, we must get the best out of them (in terms of “science per second” or “science per Megawatt”) so that work on climate change scenarios and impact studies does not lead to increased energy consumption, global emissions and overwall warming than what it may contribute to reduce...
Expected “novelties” with exascale computers and in particular EuroHPC will include a diverse range of architectures at massive scale, all of which are relevant to climate modelling. Major issues stem from the parallel programming related to the massive parallelism and heterogeneity (different architectures, hardware from a multitude of vendors, mix of CPUs/GPUs/accelerators, etc.), and those that we have identified as the most difficult to tackle are listed hereafter:

- CPU/GPU types, different interconnect, vectorization, etc.;
- ESM portability on different machines that should provide comparative scientific results. ESMs are applied worldwide on most existing architectures but bitwise reproducibility will always be a challenge. However, for validation between machines a long term climatology should be compared. Also, an experiment from in-between can not be transferred to another machine. For example, CESM is having some benchmarks which could be executed to compare results between different machines if they are satisfactory;
- Use of containers: this technology could partially resolve several of our portability problems. However, we would first need to demonstrate that the same performance can be obtained and that the outputs from the containerized model are still in line with those from the original ESM running on “bare metal”;
- ESM efficiency and performance on different machines/architectures: there is no general framework for analyzing the performance of both NorESM and EC-EARTH. However, there was some work done within IS-ENES3 for CMIP6 and the results of the comparisons of the computational performance of various ESMs have been presented here and those are also in continuous developments.

Within NICEST2 and Task 4.2, our goal is to understand the needs for running Earth System Models used in the Nordics (namely NorESM and EC-EARTH) on the future architectures within EuroHPC and on the European Open Science Cloud (EOSC).

This document reports on the bottlenecks that would hinder the efficient usage of the Nordic ESMs on EuroHPC and suggests possible remediation actions (I/Os, adding GPU support, etc.) with some information on costs in terms of manpower based on our current analysis and knowledge.

Background

Since it was not possible to address all the points mentioned above in the frame of the NICEST2 project, and as a preliminary study, it was decided to focus on porting to GPUs (for which some staff had expertise, although in different application areas).

We also chose to send a team at the CSC GPU Hackathon (October 26, November 02-04 2020) to work with NorESM, in particular BLOM (the standalone version of the ocean model used in NorESM) in order to learn more about what can be achieved in practice, where the main difficulties are, and to get something started.

BLOM (Bergen Layered Ocean Model) is the ocean component from NorESM. A full description of the BLOM model is available in the NorESM documentation.

Report on the GPU hackathon, including preparation

Preparation & introduction to the hackathon

Training on OpenACC

Before the beginning of the hackathon, we took part in a PRACE-PTC online training course (October 22-23, 2020) on "GPU Programming with OpenACC" organized by CSC - IT Center for Science Ltd. (Finland). This training was most useful to those BLOM developers and staff with little prior knowledge about accelerators. However, many examples were over simplistic (“Hello world” type) and some were not even provided in the Fortran language. On the other side the training was not sufficiently advanced for the few GPU specialists.

- Slides
- Tutorial/hands-on

As a result, although the course was useful for those completely new to GPUs it did not bring all the benefits expected to facilitate the actual work which was to be done.

Format of the hackathon

- CSC GPU Hackathon Day 1
  October 26, 2020

- CSC GPU Hackathon Day 2
  November 02, 2020
- CSC GPU Hackathon Day 3
  November 03, 2020
- CSC GPU Hackathon Day 4
  November 04, 2020

The dates above correspond to online meetings with our mentors. Most of the work was supposed to be done outside these meetings.

Tools to facilitate porting (from NVIDIA)

- Containers: used by NVIDIA mentors but most NICEST2 did not have at that time sufficient experience with containers to benefit from them. So far the containers used by our NVIDIA mentors have not been much used afterwards.
- Performance analysis tools: we had a short introduction to NVIDIA Nsight Visual Studio Edition Analysis Tools. A webinar is also available online here.
- Expertise from NVIDIA and other mentors (not all the mentors were employed by NVIDIA).

Questions addressed

Porting (e.g. running the model but not necessarily obtaining good performance) of the global climate model NorESM to the EuroHPC machine LUMI (on the CPUs partition alone) should not be challenging as the model is currently in use on various CPU based machines across Europe and Asia. On the Norwegian Sigma2 operated machine Betzy, NorESM1 has been executed on up to 18K CPU cores, providing good throughput but this was a specific high resolution model configuration. This configuration is not necessarily used routinely by researchers: NorESM is usually run at much lower resolutions (typically 1 or 2 degrees resolution) but in ensemble mode (with model perturbation) that of course scales very well (no communication because the same configuration is run 100 times with slightly perturbed inputs). Depending on resolution and I/O requirements (as each and every experiment is different), I/O and processor counts could be bottlenecks impacting model throughput.

The EuroHPC LUMI system, which will be essentially GPU-based, would require a substantial human effort for porting and efficiently running the NorESM model on the GPU-partition. As NorESM consists of more than 1.4 million lines of code and we

only recently made a first attempt to make parts of the code utilize GPUs by participating in the 2020 CSC/NVIDIA GPU hackathon, it is difficult to estimate the total manpower and time frame at this stage. But, some of the challenges from the hackathon are reported below. We participated in the hackathon with the Bergen Layered Ocean Model (BLOM) code, which is a very small part of the whole model and typically accounts for up to approximately 30% of the total CPU cost (depending on the processor layout and load balancing).

Our goal was not to focus on the possible performance improvement of the overall code but to learn and get experience about how to tackle the porting, become more familiar with the directives and get our hands dirty.

Results achieved

- About 20-30% of the main BLOM code was converted to GPUs but as mentioned earlier, the performance was not improved; actually performance is degraded as only a part of the code is migrated and data movement (CPU to/from GPU) is then dominating.
- Code with GPU instructions: https://github.com/NorESMhub/BLOM/tree/feature_GPU_channel2_hackathon. In terms of code, the most time consuming routines which account for around 30% of BLOM CPUs time were ported to GPUs. Some subroutines show a performance of factor 5 if data exchange time between CPUs and GPUs is not taken into account. Of course, when taking into account data exchange time between CPUs and GPUs, we can observe a significant degradation in performance. However, the main objective of the hackathon was not to focus on performance but to learn how to migrate an ESM to GPUs. Substantial work is required to gain performance and most of the gain would most likely come from code re-engineering.
- Tools for analyzing the performance: we have used NVIDIA Nsight for understanding the performance of the code on GPUs. Figure-1 shows the profile before starting the porting so where all the computations occur on the CPU.

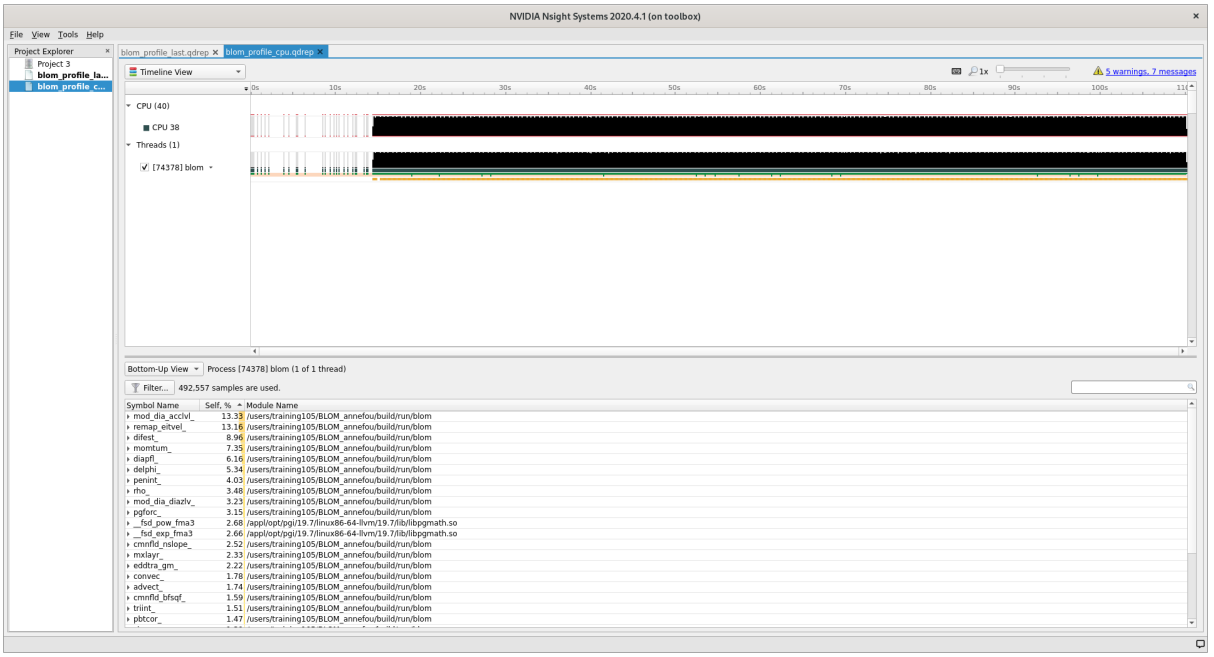

Figure-1: Profile obtained with Nsight before any porting (CPU only).

Finally Figure-2 below shows what has been achieved during the hackathon e.g. porting of the 3 routines on GPU.

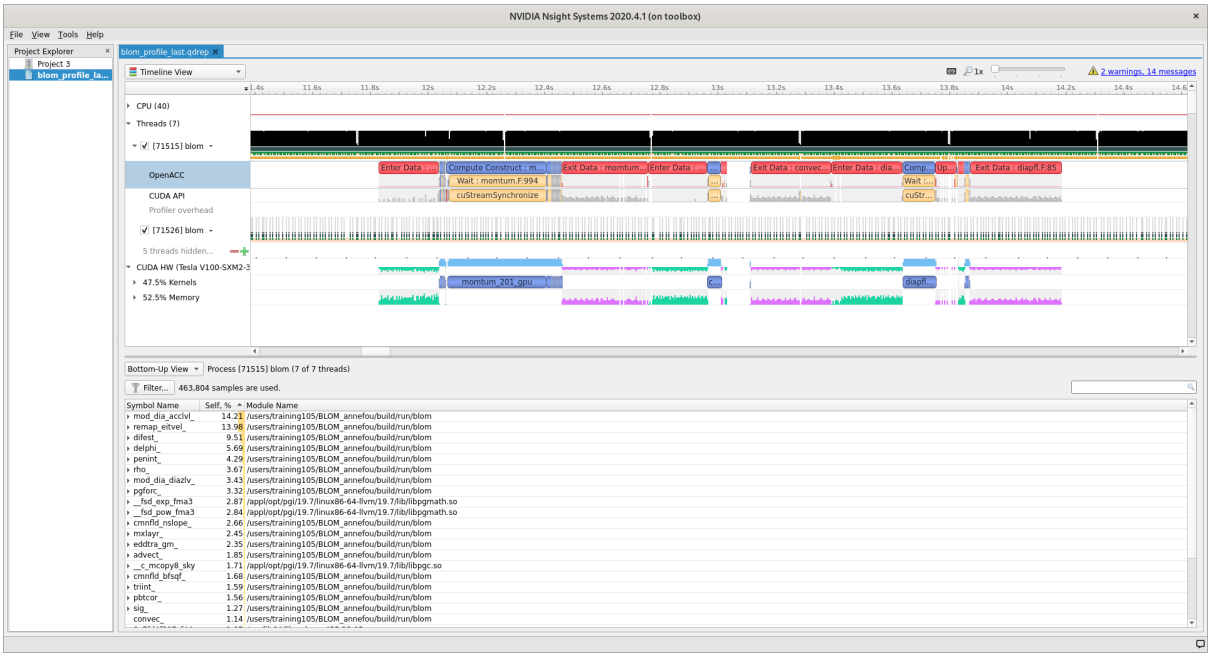

Figure-2: Profile after porting the 3 subroutines on GPUs.

In our case, as we did not focus on performance gain, the performance analysis graphs were used to highlight the porting of the routines on the GPU (see the final presentation given during the hackathon for more information; https://doi.org/10.5281/zenodo.4719793).

Lesson learned, bottlenecks and questions/Answers

Participating in the hackathon was a good idea for getting work actually initiated on the porting, obtaining support from GPU experts and, for the NICEST2 participants, acquiring valuable knowledge in this new area. However, it needs to be effectively prepared if the full benefit of such an event is to be realized. The remarks below are meant to help future teams to work more efficiently:

- BLOM contained obsolete code and needed to be significantly refactored before starting porting (COMMON blocks, SAVE, Public variables, etc.);
- GPU experts were not able to help much because they did not know climate models in general, and even less NorESM or BLOM, and they did not receive a proper introduction on it;
- General lack of knowledge of modern Fortran; also only few mentors mastered Fortran.
- Lack of commitment of some participants: several participants claimed they did not know they were supposed to work outside the organized meetings. We could have used some of the participants for performing independent checks (i.e., not those involved in porting) but we were not really prepared and did not have any code ready to be checked.
- The lack of access to machines with GPUs to continue the work after the end of the hackathon prevented us from quickly following up.
- Need to use the same software development environment: a GPU dedicated container with the NVHPC suite (built around what was known as the PGI compilers before the company became part of NVIDIA) has been developed in NICEST2 and is now available but that has not been used yet.

Several questions came up:

- What has to be checked to validate the porting, since bit-for-bit reproducibility is unlikely?
- How to involve scientists for consistency checks, to make sure that any change made did not affect the overall behaviour of the model?
- How to compare the outputs/performance of the CPU and GPU versions?
    - We should compare CPUs vs. GPUs performance and energy consumption as that comes at a cost. Also, our code should be seriously optimised to perform well on CPUs before we move to GPUs. From what other teams doing similar porting experienced it became clear that most performance gains came from improving the code and algorithms themselves in the first place.

These questions are way beyond the scope of the hackathon, but would need to be addressed to move forward.

Q&A

In this section, we have gathered a list of questions and answered based on our experience and also from inputs gathered from other partners.

- How long would it take to port our model system to GPUs?

  Starting from scratch, we spent a handful of man-days in total porting a handful of routines, so it should be feasible to parallelize all BLOM loops in 6-8 weeks. Optimizing the code, also taking MPI decomposition into account would probably be several man-months of work.

- How much efficiency we can expect for BLOM code on GPUs?

  If we assume we can get the model state into GPU memory and only need to consider the cost of spawning threads for work and synchronization, large grids should give high speedups, perhaps 5-10x that of a single node on Saga (a Sigma2 operated machine). We base this assumption on the fact that most of the i- and j-loops (where indices i and j vary laterally) are independent. But in a real setup, we would likely decompose the model grid with MPI at the top (so with multiple GPUs), and feed each GPU a smaller grid, maybe of roughly the same size as the test case we used at the hackathon. This would give less work per thread, and hence **reduced** speedup. But, small grids can also be good with regards to memory access, as you suddenly can fit the whole model state into a fast local memory. This is what we see on traditional systems when we get “superlinear” speedups. One example from the hackathon case, is the routine **diffus.F**. With a simplified configuration compared to what would typically be used in NorESM, only accessing arrays in GPU memory and only measuring the time the routine uses without copying, the speedup achieved using simple “parallel loop” type OpenACC directives with collapse() was around x5 compared to a single CPU core. This routine has independent i- and j-loops, but access 3D arrays with the typical finite difference stencil pattern, which on CPUs typically lead to high rate of Translation Lookaside Buffer (TLB) misses. On GPUs there will be similar problems, as the arrays in GPU memory are pulled into a relatively small local memory that threads can access.

- Should we use OpenACC, CUDA, or OpenMP?

  https://www.nextplatform.com/2019/01/16/burying-the-openmp-versus-openacc-hatchet

OpenACC on NVIDIA (via PGI based compilers) seems to be more mature and performant than other vendors on AMD, but support for OpenACC is coming in several compilers like GCC and LLVM.

OpenMP GPU offloading support has improved in recent years, and several presentations show that one can get nearly the performance of OpenACC on the NVIDIA platform. OpenMP has more support as an industry standard than OpenACC, so for future portability it may be better. The Intel compiler only supports OpenMP.

CUDA can give excellent performance, but is unfortunately only available for NVIDIA GPUs, and is lower level (i.e. requires more work and insight into the basic code).

For now using OpenACC seems the most realistic approach and migrating to OpenMP in the future would not be difficult once all the necessary refactoring and reengineering will have been finalized.

- What will be the major challenges?

  Substantial training should be offered on the code to facilitate migration: main code structure, code “walkthrough”, programming language (Fortran may not be well known by GPU coding experts and we may still use a few obsolete features such as common or global variables in modules, etc.), provide more detailed information and documented examples (benchmark cases) to ease comparison against the current performance of the model, identify routine and programming work that can be done by newcomers and explain in detail what these routines do and what is expected from them.

  More importantly, NorESM is closely derived from the Community Earth System Model (CESM) developed at NCAR, the National Center for Atmospheric Research (Boulder, USA). Therefore, for the most parts of the code there are few or no Nordic experts, and an efficient porting of NorESM to GPU-accelerated systems entirely relies on CESM staff, which is a big issue since, as far as we know, NCAR were not considering porting CESM to GPU but had chosen a different route (energy consumption not being a priority as in Europe). However with the recent changes initiated by the new US President this policy may be re-considered...

  For porting to GPU to be really worth it, it is paramount that either:

  > i) the **model cost** in terms of processing hours per simulated year is lower (so that a given simulation “costs” less when performed with GPUs), or

ii) the **model throughput** in terms simulated year per computational day is higher (which means that simulation results are obtainer faster with GPUs, whatever the overall performances)

in comparison to CPUs, otherwise the return on investment would be poor.

Sigma2 recently revealed their plans for adding GPU nodes on Betzy and Saga (probably not on Fram which may be too old) which means that it will be possible to do work on these machines for porting our ESMs. Also, the current trend with HPCs is to have larger and larger fractions of the compute power on GPUs (and accelerators), and less on conventional CPUs, which means that pressure to port codes to heterogeneous environments is increasing, not only to realize the full potential of this infrastructure but to be allowed to be able to run at all. Put simply: all codes not meeting certain GPU/CPU balance criteria (not fully defined yet) will not be permitted access to these new machines, which will be a serious blow.

Finally, the biggest bottleneck for re-factoring and porting ESM to GPUs is the lack of dedicated funding from the scientific community: there is a significant amount of work required, especially with BLOM, and there is no existing project to cover this as of today.

How to do better next time?

Go for a hackathon, but:

Get your code ready for porting

- **Prerequisites**
  - Remove any obsolescence from your code (COMMON blocks, SAVE, Public variables, etc.);
  - Refactor your code: be aware that most improvements from porting to GPUs come from code refactoring and algorithm changes (such as use of AI, different algorithms, better inputs and outputs, memory management, etc.)
- **Documentation**
  - Provide datasets with inputs/outputs and test cases (to check that the overall code still behaves the same)
  - Check that the code compiles is a sine qua non condition but this is not sufficient: it is important to have a minimum set of checks to make sure the model results are “acceptable”.

    - Provide small/idealized configurations which test only a restricted number of features but run fast
    - Provide more comprehensive configurations which call all the subroutines (normally used in NorESM)
- **Relevant criteria for comparing CPU and GPU outputs** (since bit-reproducibility is unlikely to be achieved and mere sums might hide issues)
    - Define a number of metrics calculated on “sensitive variables” (related to dynamics, energy conservation, momentum, etc.) to detect bugs
    - Time series to check longer term trends.
- **Performance analysis** (reference cases, reference datasets)
    - Get familiar with performance analysis tools and get ready to use them “routinely” during the hackathon (and after).

Get your team ready for porting

- **Fortran training**: so that all those to be involved speak the same language and really understand the basic principles of modern Fortran, parallel/vectorial programming
- **Best software practices training**: version control, social coding (i.e., how to get the most when working simultaneously/independently on the same code without that affecting each-other’s efforts), modular code development, etc. Training on best software practices is offered by NeIC as part of CodeRefinery.
- **Introduction to GPUs**: OpenACC/OpenMP5 training for GPUs porting
- **Introduction to Earth System Modelling for GPUs specialist**:
    - Prepare for instance a one hour presentation on the code that you plan to use in the hackathon. Also, you should have a tutorial on how to execute the model during hackathon and what are the model data and what would be output.
- **Introduction to performance analysis and other relevant tools**
    - Offer training to assess/quantify progress, validate what is being done, etc.
- **Set realistic objectives**
    - Make sure to identify a few routines before using `gprof` and present the code to the GPU specialist.
    - Organize discussions/meeting prior to the hackathon to answer a few questions:
        - Can a single black-point (e.g., not “optimizable” part of the code) ruin everything?
        - Can the model be further developed (scientific upgrade) while porting work is still in progress, to make the transition easier, or will a “new” fully ported version suddenly come up one day, with a step change (in terms of results and performance)? For instance, create a GPU branch and bring changes from original

model time to time because porting to GPUs takes time and the GPU code will only be ready to use in a couple of months or years.
    - Is a “better” model code going to facilitate further developments? As mentioned earlier, most of the improvements will come from code refactoring. Think about what could be done to facilitate future developments. This should be done before hackathon. It is also important to coordinate the code refactoring to make sure all the improvements are also incorporated in the main development code (where scientific developments are on-going).
- **Prepare examples of inputs/outputs:**
    - Make sure you prepare examples with inputs/outputs (along with all and the necessary scripts to prepare, set-up and run the experiments) with increasing complexity, to be able to quantify any performance changes, check that the model still behaves the same, etc.

Get “longer-term” access to GPUs for porting

- Secure long-term access to GPUs hardware (the hackathon used Puthi but this machine is not available to NorESM users any more) to be able to continue the work after the hackathon. If you wait too long, you will most likely forget what you have learned.
- After the hackathon, discuss long-term training (best software practices, OpenACC, OpenMP5, performance analysis, etc.), documentation, performance assessment tools, performance reporting, reference benchmarks, etc.
- Review and eventually update your working practices: setup proper Github/Gitlab repositories, improve usage of version control systems (and exploit them correctly, not take them as a burden), code standards, code reviewing, test suites, etc.
- What about Domain Specific Languages (DSLs) and related software already used by other people for a similar task? It would require more funding or dedicated projects to investigate various other options.
- **Secure funding!** If you do not get proper funding for performing the porting to GPUs, your staff will be most likely diverted to other tasks and will never be able to concentrate on the porting and get this out of the way once and for all.

Possible conflicts

- To get fast code you may need to break “good software engineering practices”
  Optimised code can be hard to understand and hard to extend/maintain
- Model code typically riddled with conditional compilation and other directives
  Hard to test, different coverage depending on compiler flags

Difficult to “read” and recognize the underlying physics equations

Conclusion

ESMs used in the Nordic countries are clearly not ready for EuroHPC. The codes are relatively large and complex, and they were continually developed over decades with contributions from generations of scientists (and very few “proper” scientific software engineers). They often include code sections with very “inconsistent” programming styles, often non-standard structures (eg. Fortran 77) which sometimes turn out as bugs with modern compilers.

We should combine efforts with other users of these ESMs elsewhere also moving to GPUs (EC-Earth consortium, CESM community, NCAR, etc.). Without communities it would be extremely difficult as we have not even got the necessary knowledge about the individual components of the various models.

Several other organizations have done extensive work to port their ESMs or NWPs to new architectures. For instance, we have heard of “success stories” from the UK MetOffice and MeteoSwiss but the amount of human labor that was needed is never explicitly stated. When asked, organizations often state they do not know because the work has been done over a long period of time (here again partly due to the lack of dedicated funding). In many cases, part of the code was re-written from scratch and the rest was ported using OpenACC and other tools, some of which “automatically” generated code suited for GPU.

The speedup achievable in other areas/disciplines was quite significant with about x40 times speedup obtained, regardless of CPUs vs. GPUs comparison, although it is not clear how much of this improvement was due to general code refactoring and how much is related to use of GPU. So, for these organizations and codes, porting to GPUs was worth it. From a general point of view, there is a need to consider code refactoring as a “current” task as important as code development itself. For instance COSMO (non-hydrostatic limited area atmospheric model used for numerical weather prediction) was ported to GPUs, which resulted in a significant energy saving and speed up. More information about their experience can be found here.

During the hackathon and within Task 4.2, we have tried to identify what has to be done to prepare NorESM/EC Earth for EuroHPC. It became clear that we should not only look in the direction of GPUs: we also have to investigate code re-engineering, vectorisation and DSLs (Domain Specific Languages).

Combining efforts for NorESM and EC-Earth would not make much sense as the two models are quite different. But, involved staff can have regular meetings which would be very helpful for knowledge exchange as none of these models is more advanced than the other one, which means that similar critical decisions will have to be made and working together to identify the most appropriate solutions could be beneficial.

At this stage, after the hackathon, there are a few lessons learned we can highlight again. One of the most important aspects is to avoid old Fortran standards and stop using common blocks and header files. Adopting best software practices is obviously a must for porting ESMs to new architecture; probably more important than the final technology used (OpenACC, OpenMP, DSL, etc.). This can be difficult for the Nordic ESM community because both NorESM and EC-EARTH are community models e.g. mostly developed and maintained by external partners. For instance, NorESM shares 95% of the code with CESM. Then, a very huge scientific group has to come together for these specifications to be heard at an internal level.

In the hackathon, and for the time being, we have chosen to use OpenACC. Moving to OpenMP would be straightforward once re-engineering work has been completed. However if one needs to start with another component than BLOM, or another ESM, then choosing OpenMP could be justified, although the main problem would then be that there is no training on OpenMP yet because this is too new.

Finally, we would like to emphasize once again that migrating to EuroHPC is not only about porting to GPUs: a lot of other aspects have to be considered to make the best of the sheer computational power, including: using other compilers optimized for the CPUs the machine will be built on, exploiting a dynamical core with better scalability, using other grids to achieve high-resolution where needed, introducing technologies (like machine learning) which can be more suited for massive parallelism than some of the existing parameterizations, using containers and workflows, etc. Porting, code refactoring and more generally the adoption of best software practices are key enabler and will allow scientists to significantly improve Earth System Models and their understanding of climate change.