HPC Courses Training Organization and Experiences in Supercomputing Luxembourg EuroCC: National Competence Centre (NCC)

Ezhilmathi Krishnasamy
University of Luxembourg
ezhilmathi.krishnasamy@uni.lu

Pascal Bouvry
University of Luxembourg
pascal.bouvry@uni.lu

ABSTRACT

High-performance computing (HPC) is a crucial field in science and engineering. Although HPC [23], is often viewed as a pure field of computer science or a subset of it, it actually serves as a tool that enables us to achieve exceptional results in science and engineering [22]. Since early on, computers have been primarily utilized for extensive arithmetic computations. However, recent advancements in electronics have also made edge computing integral to high-performance computing. Additionally, we have witnessed remarkable growth in computer architecture, leading to the development of powerful HPC machines, with supercomputers now reaching exaflop powers. Nevertheless, there are still challenges in utilizing these powerful machines due to the lack of knowledge in integrating physics and mathematics into HPC. Furthermore, complications with the software stack and common parallel programming models that target exascale computing (heterogeneous computing) persist. In this context, we present our effective course design for HPC training, focusing on CUDA, OpenACC, and OpenMP courses, which aim to equip STEM graduates with HPC knowledge. We also discuss how our training stands out in comparison to other NCC training frameworks in the EuroCC context and promotes lifelong learning.

KEYWORDS

CUDA, OpenMP, OpenACC, EuroCC, HPC Training

1 INTRODUCTION

The concept of a computer has existed since the 19th century. In the early nineteenth century, Charles Badge had the idea of a differential machine [20], one of the early visions of present-day computers. Over the years, it has been developed, and now we are in the era of exascale computing power. However, as we develop advanced computer architecture, software and parallel programming models, we also see an increasing gap in the efficient use of these powerful machines. We recognize that HPC centres have provided training for STEM disciplines for decades, but the limited number of trainers has hampered the ability to scale HPC training, and much of the training has yet to reach a broad audience. The opportunity to introduce parallel programming and supercomputer architecture to a broader group of students studying mathematics, engineering, and physics is crucial because their applications are very demanding in terms of using the power of supercomputers, whether in computational fluid dynamics, material science, or cosmology. In this work, we present a different approach that uses online courses in order to scale HPC training. At the same time, we are fortunate to live with relatively advanced technology around us; for example, our smartphones have more than a single CPU core and our laptops. However, the question is whether those with an educational background in STEM are aware of this or whether they can use these cores efficiently if they want to do any computations. Even though academic programs exist for computational science or scientific computing, they are still not well-received, and we do not have enough people graduating from those programs. Recently, in the EU, an initiative focused on the master's programme in high-performance computing[1], which focuses on from computer science to scientific computing. This initiative aims to produce more graduates who can efficiently use the available HPC power we have now and in the future to enable better science and engineering.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright ©JOCSE, a supported publication of the Shodor Education Foundation Inc.

© 2024 Journal of Computational Science Education
https://doi.org/10.22369/issn.2153-4136/15/2/4

2 RELATED WORK

There are numerous platforms and organizations that offer HPC training courses through MOOCs [3, 4, 7, 8, 11], online courses from reputable universities, PRACE Training Events [19], and HPC centers such as BSC, Julich, and CINECA. Additionally, various MOOC platforms provide different learning paths for specific courses. Massachusetts Institute of Technology also offers online courses for a wide range of topics [9]. However, there are some constraints that could hinder learners from achieving their learning goals:

- MOOC courses usually require payment and may necessitate subscribing to multiple courses.
- If a participant misses an online session, they may not be able to catch up with the course content, even if they can access the slides afterwards due to limitations in the slide information.
- Some sessions may only include 2-3 hours of presentation recordings without practical sessions. For instance, in parallel programming courses, participants may not have the opportunity to apply what they have learned from the lecture videos.

3 IMPORTANCE OF HPC IN STEM

Computers are essential for performing calculations in STEM fields, which require extensive computation and visualization. However,

[1]https://eumaster4hpc.uni.lu/

there is a noticeable gap in high-performance computing (HPC) skills among graduates and individuals with backgrounds in physics, engineering, and mathematics. For instance, someone studying mechanical engineering may be proficient in using specific software for parallel computing but may lack knowledge about the parallelization capability of the software or the HPC machine being used. Each educational program typically emphasizes its own field, which is appropriate. However, it is also important to introduce programming and computer architecture to graduates and postgraduates alongside their core STEM courses. This approach not only fosters interdisciplinary understanding but also encourages parallel simulation and computation, leading to enhanced creativity among engineers, physicists, and mathematicians and potentially accelerating technological advancements.

Efforts have been made to integrate computer and programming knowledge into STEM fields, resulting in the emergence of a new educational profile called "computational science." For example, a specialization like computational science: mechanics focuses on solids, fluids, and heat transfer, emphasizing methodologies and their computer implementation, including parallel architecture and parallel programming. We must realize how many universities worldwide offer these kinds of educational profiles. Therefore, it is essential not only for people following computational science to be exposed to computers and parallel programming but also to the entire STEM background. As we witness, we come across supercomputers and parallel programming applications in every field, whether artificial intelligence, material science, or mechanical engineering.

4 HPC KNOWLEDGE IN STEM

In the past decades, students studying computational science have been introduced to parallel architecture and parallel programming, which has enabled them to run simulations on supercomputers. Today, we are exposed to exascale supercomputers, laptops with powerful cores, and even our mobile phones. This growth is expected to continue, making a good understanding of parallel architecture and parallel programming beneficial for speeding up computations or visualizations in any STEM field.

However, we can not suddenly impose the idea that people from all STEM backgrounds must study parallel architecture or parallel programming. Therefore, interested individuals with a STEM background should be able to easily follow individual courses in computer architecture and parallel programming. Our training program is designed to help anyone quickly learn parallel programming to target various parallel architectures, such as GPU or multicore CPU.

5 HPC TRAINING INITIATIVE IN EUROPE

Europe has taken a significant initiative to promote HPC education and training. Previously, PRACE [10] made substantial efforts in promoting HPC education and training, along with other activities, such as building and maintaining new HPC machines in a centralized approach. After PRACE concluded, the EU initiated a new project called EuroHPC JU[2]. Its functionality is similar to PRACE, but it has a unique approach to research, installation of new HPC machines, education, and training. One of their primary training initiatives is EuroCC [6], in which many partner EU countries are participating and providing HPC training in their own country using either their own supercomputers or one of the JU supercomputers. This initiative is typically referred to as the National Competence Centre (NCC). Since Luxembourg is part of the EU, it also takes part in the EuroCC initiative, which promotes activities in high-performance computing (HPC) in Luxembourg, along with providing education and training in HPC. The EuroCC project is currently in its second phase, which began on September 1^{st}, 2023, and will continue until August 2026 [5].

[2]https://eurohpc-ju.europa.eu/

5.1 EuroCC

EuroCC was established around September 2020 [6], and since then, each EU member state has been promoting HPC training activities within their respective countries, particularly targeting individuals with a background in STEM education. As the initiative is relatively new, most of the National Competence Centers (NCCs) are working independently to develop courses that promote HPC education in their own countries. As part of this initiative in Luxembourg, we have conducted education and training in HPC, focusing on parallel programming courses in CUDA, OpenACC, and OpenMP.

5.2 HPC Training in NCC Luxembourg

Geographically, Luxembourg is a very small country compared to its neighbouring countries. NCC Luxembourg has three partners: Luxinnovation[3], Luxembourg University[4] and LuxProvide. Although it has many industries, including engineering and technology, especially the financial and law sectors, many professionals and graduates still lack HPC knowledge in the sense of using HPC machines (for example, intra-node and accelerators) efficiently and parallelizing their applications. Therefore, we provide HPC training in Luxembourg to promote scientific computing within Luxembourg and beyond. Our training is in English and uses the national supercomputer MeluXina (it is also part of the JU supercomputer).

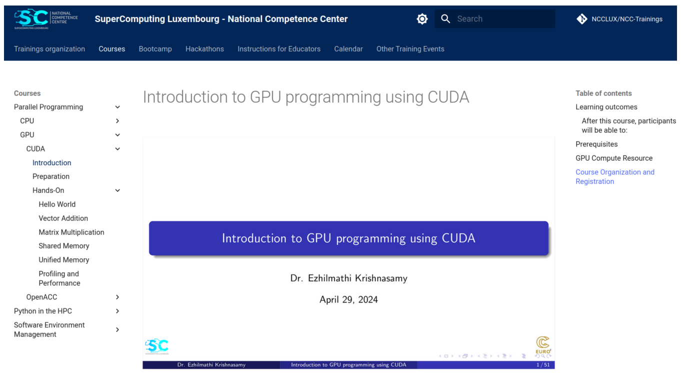

SuperComputing Luxembourg - National Competence Center
Search
NCCLUX/NCC-Trainings
Trainings organization
Courses
Bootcamp
Hackathons
Instructions for Educators
Calendar
Other Training Events
Courses
Parallel Programming
CPU
GPU
CUDA
Introduction
Preparation
Hands-On
Hello World
Vector Addition
Matrix Multiplication
Shared Memory
Unified Memory
Profiling and Performance
OpenACC
Python in the HPC
Software Environment Management
Introduction to GPU programming using CUDA
Introduction to GPU programming using CUDA
Dr. Ezhilmathi Krishnasamy
April 29, 2024
Dr. Ezhilmathi Krishnasamy
Introduction to GPU programming using CUDA
1 / 51
Table of contents
Learning outcomes
After this course, participants will be able to:
Prerequisites
GPU Compute Resource
Course Organization and Registration

Figure 1: The NCC Luxembourg training website page.

[3]https://www.luxinnovation.lu/
[4]https://www.uni.lu/en/

6 TRAINING STRUCTURE

Since our NCC is very young and before we start training, we wanted to make sure we provide excellent and optimistic training within HPC. For this, we have adopted two well-known and well-practised instructional design (ID) approaches: the ADDIE model [13], see Figure 2 (left) and Merrill's First Principle Instructions [17], see Figure 2 (right).

The ADDIE model consists of five key components:

- Analysis
- Design
- Development
- Implementation
- Evaluation

Analysis. Initially, we distributed questionnaires to public and private institutes in Luxembourg. The questionnaires aimed to gather information on the use of parallel simulation, GPU usage, knowledge of efficient multicore CPU utilization, and more. For example, we asked about the fields in which parallel simulation is being used, the use of GPUs or willingness to use GPUs, and knowledge of efficient multicore CPU utilization. This information will provide an overview and help reduce the performance gap of HPC skills in Luxembourg. The responses provided valuable insights, allowing us to understand the current landscape and identify areas for improvement. To identify knowledge and skills gaps, we collected essential information to design our HPC training. This included determining the target audience, skill levels, computing resources, and participant's interest in various HPC training topics. Based on our analysis, we decided to organize the training into the following courses:

- CUDA
- OpenACC
- OpenMP

Design. In the design phase of the ADDIE model, we focused on different topics and their learning outcomes. We primarily concentrated on the course structure and teaching aids, such as choosing the website layout for hosting course material to include both lecture material and hands-on sessions. We also selected appropriate topics (learning outcomes) for lectures and hands-on sessions. For hosting the teaching material, we decided on the GitHub repository and created the hands-on session; we chose MkDocs to create the website[5], please see Figure 1. During the design phase, we decided to structure our training as follows:

- Preparation Session (approximately 1 hour)
- Break 1 (lunch break, up to 60 min.)
- Lecture Part 1 (strictly 45 min.)
- Break 2 (up to 15 min.)
- Lecture Part 2 (strictly 45 min.)
- Break 3 (up to 15 min.)
- Hands-on Session 1 (strictly 45 min.)
- Break 4 (up to 15 min.)
- Hands-on Session 2 (strictly 45 min.)

We limit each lecture and hands-on session (except the preparation session) to a maximum of 45 minutes. This is primarily based on attention span and learning, cognitive load theory, spacing effect, student engagement, information retention, and learning outcomes [12, 14, 16, 18, 21]. In between lectures and hands-on sessions, we have a 15-minute break, which is also loosely based on Pomodoro Technique [15], where a short period of break increases the learning capacity. Our courses are meant to last only half a day, which is 4-5 hours. We believe that this would be very efficient and compact. In addition, HPC courses are mostly self-thought-out after participants take the classes. The more someone practices, the more experience they will get. Therefore, keeping the course for an entire day or more than a day would not be very efficient in terms of learning outcomes.

[5]https://ncclux.github.io/NCC-Trainings/

Development. In the Development section, we conduct our training online to ensure that everyone can participate without any constraints, such as needing to be on-site (which would require travel time) or take leave from work. This is especially important as it allows people with transportation issues to participate as well. Additionally, we wanted our training activities to reach beyond Luxembourg, so we prefer to keep our training online. During the development stage, we prepared the lecture and hands-on session using GitHub with MkDocs, while ensuring that our training material fulfills the learning outcomes. We also utilized Material for MkDocs, which provides a nice layout for text, figures, and coding.

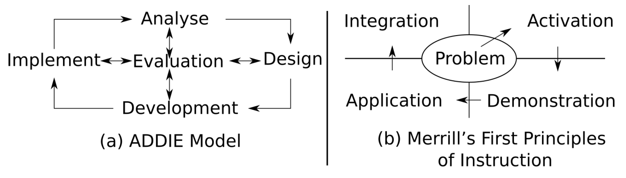


Figure 2: A schematic overview of the ADDIE Model (left) and Merrill's First Principles of Instruction (right).

Implementation. This phase involves delivering the course to the participants. We typically announce the course 5-6 weeks before it starts to allow participants to register. We use various communication channels such as Twitter, LinkedIn, and our course webpage to promote the training. After each course, we gather feedback from the participants to make improvements.

Evaluation. We analyze the feedback collected during and after the training to determine if any suggestions need to be adopted for future iterations. We maintain this approach on a regular basis to ensure efficient learning outcomes for learners. Even if we meet the learning outcomes after the training, we still look for further improvement in the course development. Typically, after the course, we send out a questionnaire to participants to improve our course content. For example: 1) Was the trial-run session informative, or do you need more instructions on the training page? 2) Did the lecture portion was informative and aligned with the course learning outcomes? 3) Were the hands-on sessions compatible with the lecture content and aligned with the learning outcomes?

We received the following feedback: participants would like more information on the trial-run session, and in addition, they would like to participate in advanced courses for inter-node computation.

This feedback will be included in the following training sessions, and new training on inter-node environment is being prepared.

Participants liked our approach, such as 45-minute lectures (45 X 2) with breaks and 45-minute hands-on sessions (45 X 2). This is because in some courses, the lecture part goes beyond 45 minutes, and the hands-on session also goes on continuously without a break. Additionally, they are really happy with the hands-on sessions and how they are organized, which are aligned with the lecture part and learning outcomes. Overall, participants are satisfied with each of our courses, which deal with logging in to supercomputers, learning parallel programming, and performing analysis.

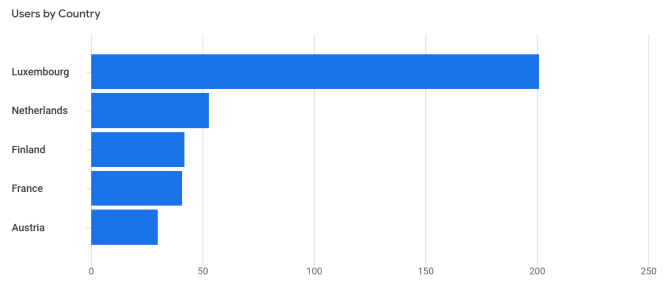


Figure 3: The number of participants who actively used the learning website during and after the course.

We also monitor our training page through Google Analytics for further analysis of hands-on sessions; for example, some participants spend less or more time on one topic. This eventually leads to more thought about that topic for further improvement; for instance, Figure 3 shows the user's statistics (by country) we obtained from Google Analytics.

As we have discussed earlier, the ADDIE model has been adopted to develop our courses; however, we paid even more attention to the hands-on session and followed Merrill's First Principles of Instruction. The following shows those principles and how they have been adopted to organise our hands-on session:

- Problem-centered: We start with a simple `hello world` program and then progress to more BLAS examples.
- Activation: Each part of the hands-on session addresses key questions derived from the learning outcomes stated in the course description, ensuring active participation throughout the session.
- Demonstration: Initially, we present an example of a specific problem, such as vector addition. We display the C/C++ code of the sequential version and provide the framework for converting it to the parallel version. Participants are required to focus on crucial areas to subsequently convert the code to the parallel version.
- Application: We align practice activities with learning outcomes and gradually reduce guidance to foster learner's independence. For instance, we introduce entirely new questions based on previous problems that are still closely related. Please refer to list 7.4 for an example, where we pose questions that are linked to previous ones, but participants need to work independently.
- Integration: We encourage learners to apply their learning to their own scientific problems at work or in research. Towards the end of the training, we prompt participants to apply the learning outcomes to their own problems. For example, the CUDA programming model can be employed in fields such as CFD, solid mechanics, material science, and biomedical applications.

7 DETAILED TRAINING OUTLINE

In this section, we provide practical details about our training, focusing on learning outcomes to give readers a better idea of how the training was conducted.

7.1 Preparation

Our courses are tailored for individuals with a STEM educational background, so we don't anticipate extensive experience with the HPC machine. Therefore, we commence our training with a preparation session. During this session, participants can log into the MeluXina[6] HPC machine to familiarize themselves with it. This is crucial as they will need to use the machine later for the practical sessions. The purpose of this session is to minimize any time loss during the practical sessions and to assist participants in loading the necessary modules for compiling and executing the test problem. This way, they will already have an understanding of how to log in to the machine, load the modules, compile, and execute the tasks. We schedule this session for about an hour, usually before lunch, typically between 11:00-12:00.

7.2 Lecture part 1

In this first part of the lecture we will outline the course structure and discuss the significance of HPC (High Performance Computing) along with its various applications. We will cover examples of its use in computational fluid dynamics and material science. Since our participants come from diverse educational backgrounds, we aim to demonstrate the relevance of HPC in their respective fields rather than focusing on a single domain.

We will begin by explaining the basic architecture of computers. Our goal is to establish a solid understanding of parallel architecture before diving into parallel programming syntax, such as CUDA or OpenMP. By understanding the distinctions between multicore CPUs vs. single-core CPUs and CPUs vs. GPUs, participants will be better prepared to grasp parallel programming concepts. This practical approach is similar to understanding a car's design before learning to drive. It's beneficial to comprehend the car's design, such as the number of available gears, speed, camera settings, etc., for better manoeuvring the vehicle.

One example that illustrates the distinct difference between GPUs and CPUs is their core count. GPUs have more cores than CPUs, making them faster. For instance, the Intel® Core™ i7-10700K Processor has a base frequency of `3.80 GHz`, while the Nvidia Ampere GPU has a frequency of `0.765 GHz`. Although the CPU's frequency is higher, the GPU can achieve high throughput due to its numerous cores and ability to handle parallel threads more efficiently than the CPU. Additionally, we provide various figures

[6]https://www.luxprovide.lu/meluxina/

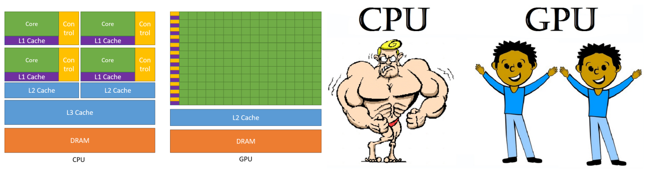


Figure 4: A simple illustration of the difference between the CPU and GPU (left). The CPU has a higher frequency than the GPU; however, the GPU operates in groups using SIMT (right).

to support these concepts; Figure 4 is an example that visually demonstrates the disparity between CPU and GPU.

In the subsequent sections, we will delve into essential code syntax and APIs of parallel programming models, explaining their use cases and suggesting effective utilization strategies.

7.3 Lecture Part 2

The second part of the lecture analyses more detailed aspects of the topic. For example, when discussing CUDA programming, we start by explaining how to print a simple `hello world` program from the GPU. We then cover fundamental concepts in parallel programming applicable to both CPUs and GPUs. Additionally, we explain memory management, and how parallel threads can be created and used efficiently. Finally, we conclude lecture part 2 by presenting examples of BLAS operations and emphasizing the use of profiling tools.

7.4 Hands-on Session 1

For the hands-on session, we have developed a website that participants can access, for example, in OpenACC [2]. We cover a minimum of 5 key topics in the hands-on session; the first session will cover topics 1 to 2. Each section will examine a specific topic concept in detail with an explanation and will also include questions. Instead of simply asking participants to solve problems, we pose questions related to techniques that could be used to solve a problem. For instance, in the CUDA programming course, in the vector addition section[7], we ask the following questions for the hands-on session topic 1:

- What happens if you remove the `_syncthreads()` from the `__global void vector_add(float *a, float *b, float *out, int n)` function?
- Can you remove the if condition $if(i \leq n)$ from the `global void vector_add(float *a, float *b, float *out, int n)` function? If so, how can you do that?
- Here, we do not use the `cudaDeviceSynchronize()` in the main application. Can you figure out why we do not need to use it?

7.5 Hands-on Session 2

In the second part of the hands-on session, we will cover the remaining training topics. The hands-on exercises are well-designed, with questions for participants to work on. Each topic will cover specific concepts, from fundamental to advanced, and we have provided code skeletons for each concept to ensure that participants can complete the exercises within the allotted time. Even if someone cannot finish during the training session, they can still use the material for self-paced learning, as we plan to keep the training material available for an extended period.

8 TOPICS AND LEARNING OUTCOMES

The "Topics and Learning Outcomes" section outlines the three courses and their respective learning outcomes, as well as our approach to training based on these outcomes.

8.1 Training in CUDA

The CUDA training is designed for participants at beginner to intermediate levels [1], and the learning outcomes are as follows:

- Understanding the GPU architecture (and also the difference between GPU and CPU)
  - Streaming architecture
  - Thread blocks
- Implement the CUDA programming model
  - Programming structure
  - Device calls (thread blocks organization)
  - Host calls (kernel calls)
- Efficient handling of memory management
  - Host to Device
  - Unified memory
- Apply the CUDA programming knowledge to accelerate examples from science and engineering
  - Iterative solvers from science and engineering
  - Matrix multiplication, matrix-vector multiplication, BLAS routines, etc.

Lectures. In the "Lectures" section, the focus is on comparing the basic architecture of the CPU and GPU. The concept of streaming multiprocessors (SMs), thread blocks, and single instruction multiple threads (SIMT) programming methodology on GPUs is discussed. The organization of SMs in the GPU and its memory organization between global, L2, shared, and L1 are detailed. The organization of thread blocks in the CUDA programming model is emphasized, along with the creation and conversion of 1D, 2D, and 3D thread blocks as needed. The latest GPU architecture advancements and compute capability are also presented. A simple CUDA programming model is demonstrated, including the construction of device functions and device calls from the host (kernel) using necessary API and CUDA qualifiers. Finally, essential BLAS operations like vector addition and multiplication using the CUDA programming model are showcased. The section concludes with a brief introduction to profiling the code and identifying bottlenecks, such as GPU occupancy and data transfer time between CPU and GPU.

Hands-on Session. During the hands-on session, participants can go through the example and work on some questions based on that example. In this way, they would have a better understanding, which they could connect with lectures and hands-on sessions.

We structured our hands-on session as follows:

[7]https://ncclux.github.io/NCC-Trainings/cuda/exercise-2/

- Hello World: It shows how to use simple CUDA APIs to work on `Hello World`, for example, kernel calls and how to include the thread blocks in the kernel block.
- Vector Addition: It is interesting, in terms of creating more thread blocks, for example, 1D simultaneously, how computation can be offloaded to GPU, such as data transfer to GPU and transferring back solution to CPU.
- Matrix Multiplication: Till now, participants have seen how to make just one loop into a 1D block that matches SMIT, whereas in matrix multiplication, out of three loops, inner loops should be sequentialised. This is an excellent example of parallelizing multiple loops.
- Shared Memory: Most of the time, we might end up doing computation, which uses data that is being used often. Hence, knowing how to keep data in the shared memory decreases the latency; we support this example with matrix multiplication.
- Unified Memory: It simplifies the CUDA programming structure. Most programmers make data-handling mistakes between CPU and GPU. Therefore, `unified memory` option would be a good option if someone does not want to pay more attention to data movement between CPU and GPU.
- Profiling and Performance: It is essential to understand how the given code is being executed efficiently in the GPU; we show some examples of Nsight Systems and Nsight Compute.

8.2 Training in OpenACC

Below are the learning objectives for the OpenACC programming model.

- Understanding the GPU architecture (including the differences between GPU and CPU)
  - Streaming architecture
  - Thread blocks
- Implementing the OpenACC programming model
  - Compute constructs
  - Loop constructs
  - Data clauses
- Efficient memory management
  - Host to Device
  - Unified memory
- Applying OpenACC programming knowledge to accelerate science and engineering examples
  - Iterative solvers from science and engineering
  - Matrix multiplication, matrix-vector multiplication, BLAS routines, etc.

In our OpenACC course, we primarily focus on GPU execution. This is because we aim to provide an alternative option for GPU programming, instead of the CUDA programming model. During the lectures, we investigate the fundamental differences between GPU and CPU architectures, including details such as streaming multiprocessors (SMs), GPU processing cluster (GPC), and memory hierarchy in the GPUs. We also discuss thread blocks in OpenACC, which, despite its similarity to directive programming, provides a low-level API for defining thread blocks that can be set manually.

The first part of the lecture covers important APIs from OpenACC, such as `parallel` and `kernel`, which are used for parallelizing loops. Data movement in the programming model is a crucial aspect of GPU programming, and we cover high-level APIs available in OpenACC for data movement between CPU and GPU, such as `create`, `copyin`, `copyout`, and `copy`. Additionally, we demonstrate working examples of BLAS routines, which involve nested loops and data movement of dynamic arrays between CPU and GPU. Finally, we explore the use of `unified memory` programming options and profiling tools to efficiently parallelize sequential code on a GPU.

Hands-on Session. During the hands-on session, we covered the following topics:

- Compute Constructs and Parallelize Loops: We provided examples of using `parallel` and `kernels`, such as the `hello world` example with a loop.
- Data Locality: We discussed OpenACC's low-level and high-level APIs for data movement between GPU and CPU, and we considered the use of BLAS routines to support this concept.
- Optimize Loops: By default, the OpenACC compiler selects an appropriate number of thread blocks to parallelize the loops. However, sometimes it is beneficial to set multiple thread blocks for better performance. This section covers these approaches.
- Unified Memory: Similar to the concept of CUDA, in OpenACC, enabling `unified memory` is as simple as adding a compiler flag. This simplifies the necessary data transfer API in OpenACC.
- Profiling and Performance: This provides more information during compilation, such as kernel execution thread blocks and the frequency of other API calls throughout the application execution. Additionally, the NVHPC compiler offers numerous options for profiling OpenACC code.

8.3 Training in OpenMP

The OpenMP training introduces parallel architecture and gradually introduces parallel programming; the following are the learning outcomes from OpenMP training.

- Understanding the shared memory architecture
  - Unified memory access (UMA) and Non-unified memory access (NUMA)
  - Hybrid distributed shared memory architecture
- Implementing the OpenMP programming model
  - Parallel region
  - Environment routines
  - Data sharing
- Efficient handling of OpenMP constructs
  - Work sharing
  - Synchronization constructs
  - Single instruction multiple data (SIMD) directive
- Applying OpenMP programming knowledge to parallelize examples from science and engineering:
  - Iterative solvers from science and engineering
  - Matrix multiplication, matrix-vector multiplication, BLAS routines, etc.

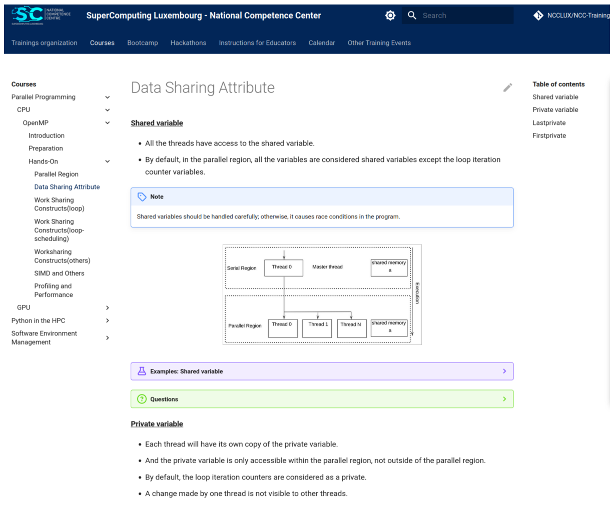
Lectures. OpenMP begins with an overview of computer architecture, covering shared and distributed memory architecture. It briefly explains single instruction single data (SISD), single instruction multiple data (SIMD), multiple instruction single data (MISD), and multiple instruction multiple data (MIMD) architectures. This knowledge is essential for understanding how programming can utilize the OpenMP programming model.

Next, the lecture explores the important role of the OpenMP environment, which is necessary for the successful implementation of the OpenMP programming model. The discussion then covers topics such as data sharing and work sharing, including the loop construct, sections construct, single construct, and master construct, which are crucial for SIMD and data lock or to avoid data race conditions.

Finally, the lecture concludes by demonstrating the use of various profiling tools for OpenMP performance analysis.

Hands-on Session. The hands-on session for OpenMP is organized as follows:

- Parallel Region: The `parallel` construct is crucial to understand when learning the OpenMP programming model. In this section, we demonstrate how to use the `parallel` construct and how to invoke it in the application. We also introduce other OpenMP API routines needed for the `parallel` construct.
- Data Sharing Attribute: To avoid data race or deadlock in OpenMP, we explain the methodology and provide examples using `private`, `shared`, `firstprivate`, and `lastprivate`.
- Work Sharing Constructs (loop): The previous `parallel` construct created a parallel region, but often we need parallelism in the loop. For this, we introduce `#pragma omp for`, which enables SIMD programming capability.
- Work Sharing Constructs (loop-scheduling): This section helps programmers efficiently share loops based on the problem and available resources, using `static`, `dynamic`, `guided`, and `auto`.
- Worksharing Constructs (others): Here, we introduce how to parallelize nested loops, which is commonly encountered, especially during BLAS operations.
- Profiling and Performance: We introduce profiling tools such as Intel APS, ARM Forge, Intel Advisor, Intel Inspector, and Intel VTune for a better understanding of profiling the code, memory leaks, latency, finding potential hot spots, etc. Figure 5 shows an example of the hands-on session topics in OpenMP.



Data Sharing Attribute

Shared variable

- All the threads have access to the shared variable.
- By default, in the parallel region, all the variables are considered shared variables except the loop iteration counter variables.

Note

Shared variables should be handled carefully; otherwise, it causes race conditions in the program.

Private variable

- Each thread will have its own copy of the private variable.
- And the private variable is only accessible within the parallel region, not outside of the parallel region.
- By default, the loop iteration counters are considered as a private.
- A change made by one thread is not visible to other threads.

Figure 5: An example from our hands-on session on OpenMP.

9 CONCLUSIONS AND FUTURE DIRECTIONS

The training within NCC Luxembourg took place in 2023. Each course had around 20-30 participants. We have observed that the approach we adopted, especially the course delivery structures, is working very well for HPC courses. We are expanding our offerings to include more domains and additional parallel programming courses such as OpenMP Offloading, SYCL, OpenCL, etc. Additionally, we plan to offer courses in HPC software for science and engineering applications, such as ANSYS, OpenFOAM, GROMACS, and ABINIT. Furthermore, we intend to organize boot camps and hackathons in collaboration with local companies and hardware vendors like Nvidia and AMD.

ACKNOWLEDGMENTS

The authors express their gratitude to EuroHPC JU and EuroCC for funding the training initiative in Luxembourg. Additionally, appreciation is extended to our NCC partners: Luxinnovation, Luxembourg University, and LuxProvide, for their support during the training event.

REFERENCES

[1] 2023-2024. Introduction to GPU Programming Using CUDA. https://ncclux.github.io/NCC-Trainings/cuda/
[2] 2023-2024. Introduction to OpenACC for Heterogeneous Computing. https://ncclux.github.io/NCC-Trainings/openacc/
[3] 2024. Coursera. https://www.coursera.org/
[4] 2024. edX. https://www.edx.org/
[5] 2024. EuroHPC JU - EuroCC1. https://www.eurocc-access.eu/
[6] 2024. EuroHPC JU - EuroCC2. https://eurohpc-ju.europa.eu/
[7] 2024. FutureLearn. https://www.futurelearn.com/
[8] 2024. Khanacademy. https://www.khanacademy.org/
[9] 2024. MIT OpenCourseWare. https://ocw.mit.edu/
[10] 2024. Partnership For Advanced Computing In Europe. https://prace-ri.eu/
[11] 2024. Udacity. https://www.udacity.com/
[12] Donald Bligh. 1985. What's the use of lectures? *Journal of Geography in Higher Education* 9, 1 (1985), 105–106.
[13] Robert Maribe Branch. 2009. *Instructional design: The ADDIE approach.* Vol. 722. Springer.
[14] Nicholas J Cepeda, Harold Pashler, Edward Vul, John T Wixted, and Doug Rohrer. 2006. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological bulletin* 132, 3 (2006), 354.
[15] Francesco Cirillo. 2018. *The Pomodoro technique: The acclaimed time-management system that has transformed how we work.* Currency.
[16] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. 2014. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the national academy of sciences* 111, 23 (2014), 8410–8415.
[17] M David Merrill. 2007. First principles of instruction: A synthesis. *Trends and issues in instructional design and technology* 2 (2007), 62–71.
[18] Fred Paas, Alexander Renkl, and John Sweller. 2003. Cognitive load theory and instructional design: Recent developments. *Educational psychologist* 38, 1 (2003), 1–4.
[19] PRACE 2024. *PRACE Training Events.* PRACE. https://events.prace-ri.eu/category/1/.

[20] Doron Swade and Charles Babbage. 2001. *Difference engine: Charles Babbage and the quest to build the First Computer.* Viking Penguin.
[21] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.
[22] U.S. Department of Energy 2024. *Exascale Computing Project.* U.S. Department of Energy. https://www.exascaleproject.org/.
[23] Laurence T Yang and Minyi Guo. 2005. *High-performance computing: paradigm and infrastructure.* John Wiley & Sons.