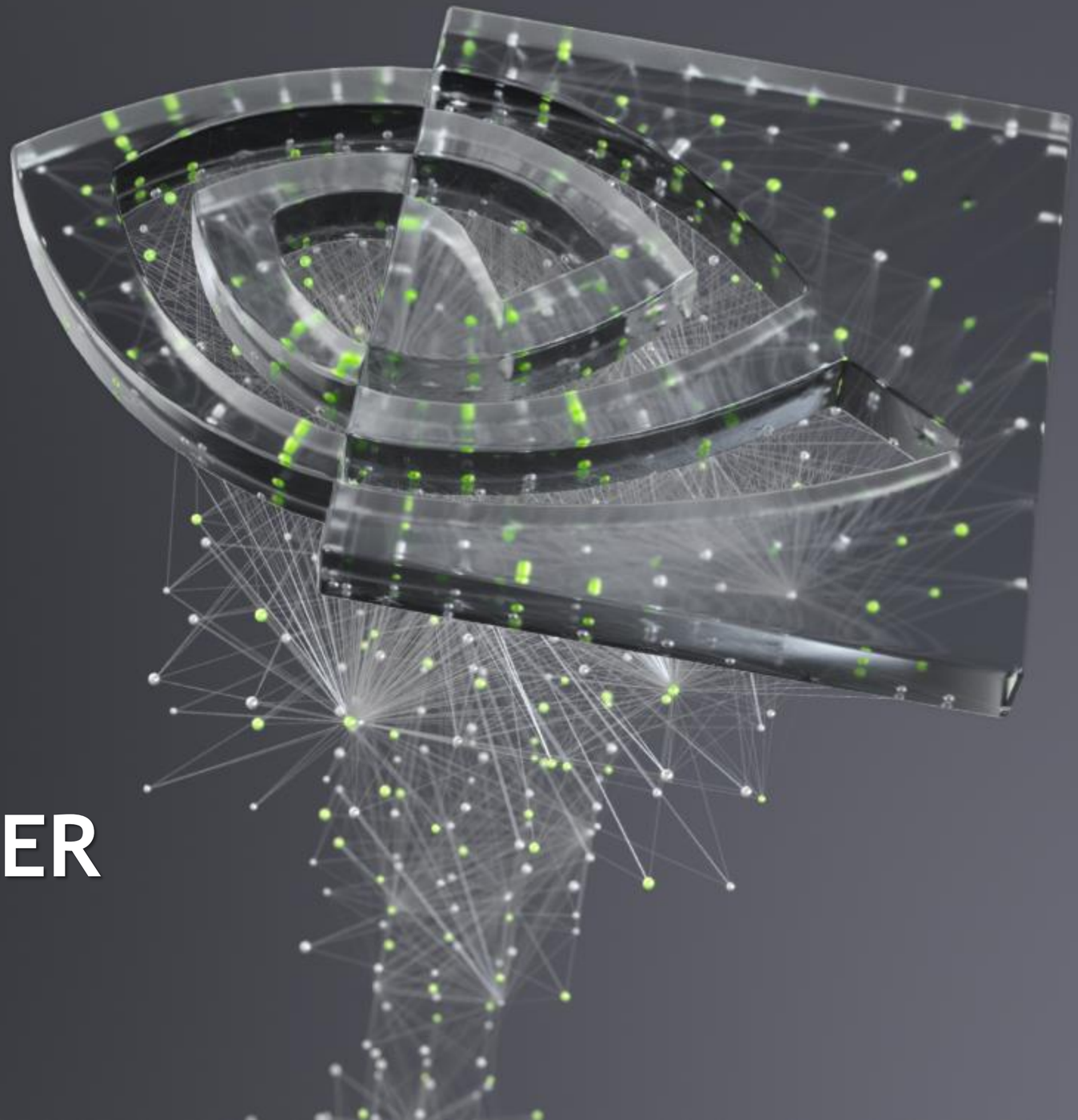


COLLECTIVE PROFILER

Avril 2021

MOTIVATION

Why a new profiler?

- Project was initiated late spring 2020
- Tool specifically to *investigate* the behavior of MPI collective operations (this is not a product)
  - MPI standard: https://mpi-forum.org
  - Currently focusing on alltoallv; support for alltoall under development
- Be able to gather data about MPI collective operations to analyze performance bottlenecks
  - Without access to the application code
  - On any execution platform, even with limited access
  - With a large count of ranks and nodes
  - With a toolchain to analyze the data and help us understand how the collectives behave
- Git repository: https://github.com/gvallee/collective_profiler

ARCHITECTURE OVERVIEW

- Two separate parts integrated together in the repository
  - The profiler itself: C/PMPI code in the *src* directory; a set of shared libraries used when executing MPI applications
  - The post-mortem analysis tool: Golang code in the *tools* directory; a set of commands to analyze and investigate profiles
- *README.md* file provides information about how to install and use both
- A few commands overview (more details in the coming slides)
  - *make* compiles the profiler and the postmortem analysis tool (MPI and Golang needs to be already installed)
  - mpirun -x LD_PRELOAD=liballtoallv_counts.so ./app.exe creates a section of the profile
  - profile -dir ~/data/alltoallv_profile goes through the profile and compiles data and statistics

DESIGN AND IMPLEMENTATION CONSTRAINTS

- The design and implementation is based on the following constraints
  - Must support ~5,000 MPI ranks
  - Must support ~1,000,000 alltoallv calls
- These scales require to choose and implement algorithms and data structures very carefully
- All contributions are expected to respect these constraints
- A validation tool is available to help: *tools/cmd/validate/validate*
  - Separate command and infrastructure because it goes beyond unit testing: unit testing + end-to-end testing
  - If the validation passes and the code does not create maintenance issues, it gets into the repository
  - If issues are discovered after the code has been included, the validation process is extended to detect these issues and avoid future regressions

CREATION OF A ALLTOALLV PROFILE

- Profiles are composed of 4 different types of data: counts, backtraces, rank locations, timings (both late arrival and time spent executing the collective)
- 5 different shared libraries: liballtoallv_counts.so, liballtoallv_backtrace.so, liballtoallv_location.so, liballtoallv_late_arrival.so, liballtoallv_exec_timings.so
  - Having separate shared libraries minimizes interferences between different aspects of profiling
  - Give the opportunity to optimize the gathering of specific data (out-of-scope of this presentation), e.g., low-memory systems
- You will need data from profiling the applications with the 5 shared libraries
- Multiple data formats available that are designed to minimize issues due to our constraints (e.g., number of files, size of files). Default format is suitable in most cases.
- Please read the README.md file for more details

PROFILES

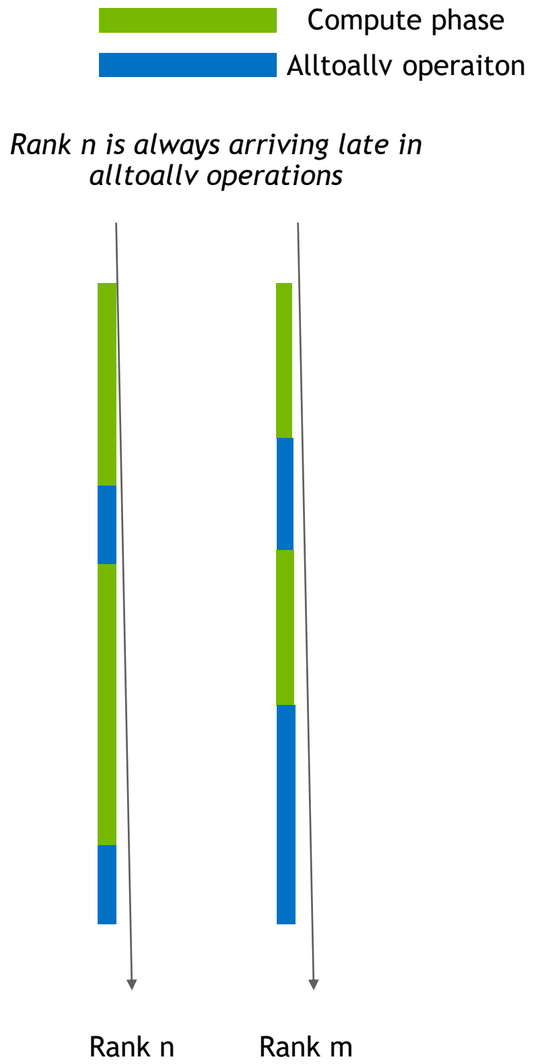
Concept of counts, location and backtraces

- Counts: please refer to the MPI standard
- Location: on which node are ranks of a communicator located?
  - Useful when trying to know if the network capacity to a node is entirely utilized
  - Can be used to investigate better placement of ranks
- Backtrace: application context in which the alltoallv operations are called
  - Extract useful information even without access to the source code
  - Help understand the context of under-performing alltoallv operations

PROFILES

Concept of late arrival and execution times

- Late arrivals
  - Based on what each rank is computing, some ranks may start alltoallv operations late compared to others
  - Early ranks need to wait for the late ranks
  - Create imbalance that are often greatly degrading the performance of alltoallv operations
- Execution times
  - Time actively spent in the alltoallv operation
  - Use a specific methodology to differentiate delays and execution time (out-of-scope of this presentation)



POST-MORTEM ANALYSIS

Overview

- Things to keep in mind
  - The tool's APIs, algorithms and data structures heavily rely on Golang maps: efficient both in terms of execution time (good access complexity) and memory usage (memory pointer within the Golang runtime)
  - Analyzing large datasets may require a system with a decent amount of resources (CPU and memory)
- Two commands are of interest for your project: *profile* and *webui*
  - Respectively in `./tools/cmd/profile` and `./tools/cmd/webui`
  - Default arguments should be adequate
  - More details in the next slides

POST-MORTEM ANALYSIS

Concept of patterns

- MPI collectives involve all ranks in the communicator that is used
- But for alltoallv, the amount of data sent/received by each rank is defined by the counts
- A typical issue is that developers use alltoallv operations to implement data exchange between a small subset of the ranks
- The concept of pattern captures how many ranks are actively communicating
  - 1-to-1: only a few ranks are exchanging data
  - 1-to-n: a few ranks sent data to many other ranks
  - n-to-1: many ranks send data to a few ranks
  - n-to-m: most of the ranks exchange data
- Patterns therefore focus on the number of ranks that are actively involved

POST-MORTEM ANALYSIS

Concept of heat map

- Based on the counts and the datatype defined during the alltoallv operation, we know the exact amount of data exchanged between ranks
- The concept of heat map captures how much data is exchanged, either between rank (rank-centric heat map) or hosts (host-centric heat map)
- Heat maps therefore focus on the amount of data that is exchanged

POST-MORTEM ANALYSIS

Webui

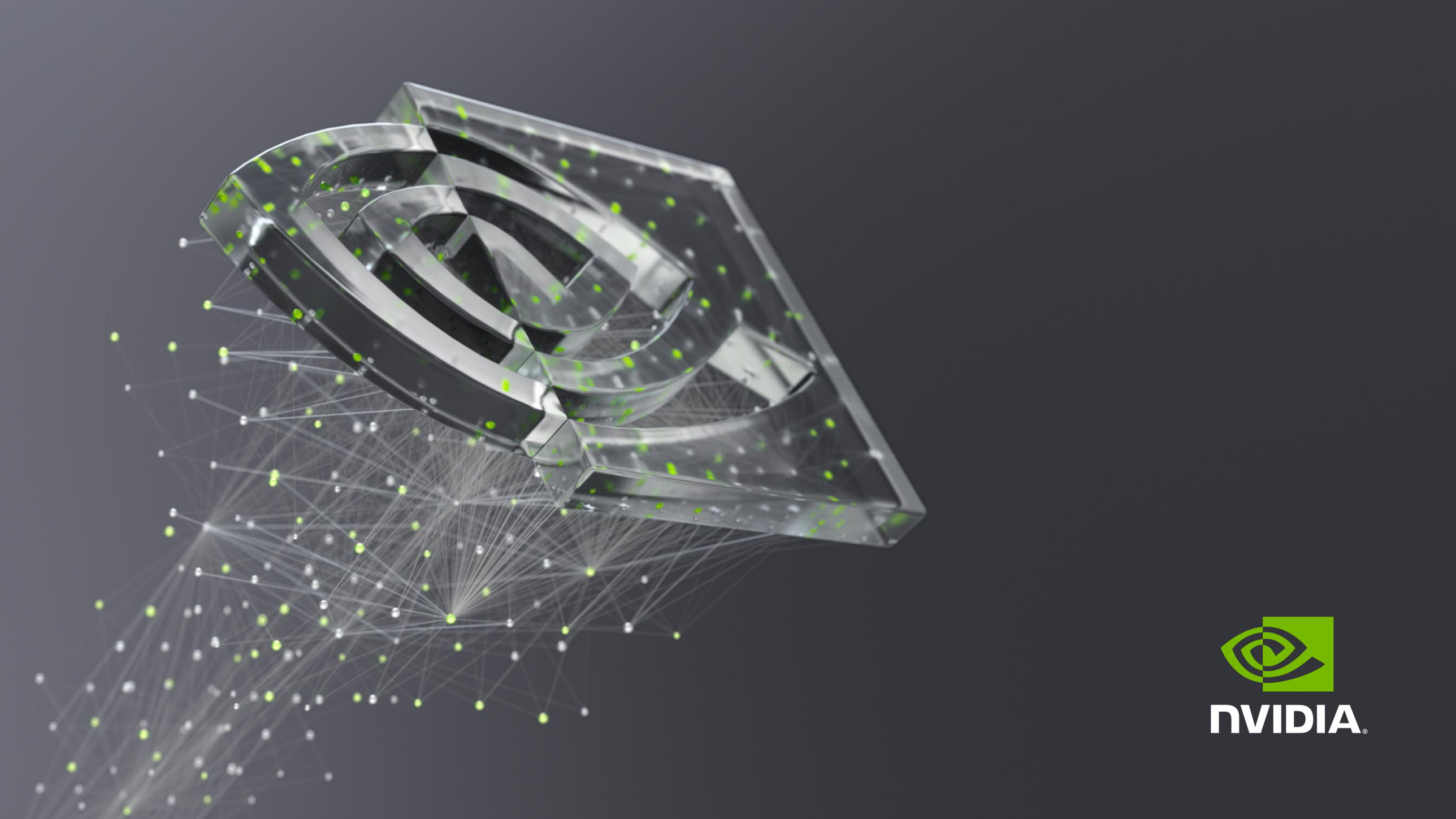
- Tool to visualize and investigate profiles: *tools/cmd/webui/webui*
- *Requires gnuplot*
- Key features
  - Automatically performs missing post-mortem analysis when required, including plots
  - List of all the calls and possibility to select a call to see details
  - Display patterns that has been detected
- Its layout still has limitations, it is still evolving (e.g., no support for multi-communicator profiles)
- Demo (if we have time)

THE CODING CHALLENGE TASKS

- Understand MPI_alltoallv calls - write a simple program that shows differences between two balanced and unbalanced patterns
- Get to know the profiler, be able to run it
- Pattern Display
- Create a map of patterns
- Use more colors for the patterns with different possible formulas
- Find a way to use it in a real application, in our case WRF (3 domain input, already available)
- Bonus tasks
  - Find ways to reduce the running time of the profiler
  - Find ways to reduce disk space of the profiler

WORKING MODE

- We suggest that each team use a cloned environment for this project and add Geoff V. to review the code when ready. The cloned environment can be private.
- The best codes will be merged into the master after being reviewed


NVIDIA