GSOC 2020 Proposal - Unified API for Algorithms

Contact Information

Name: Shrijit Singh
Email: shrijitsingh99@gmail.com
Phone: +91-9427425742
GitHub: https://github.com/shrijitsingh99/
LinkedIn: http://linkedin.com/in/shrijitsingh99/

Abstract

With the current trend of increasing parallelism, there is a need for a uniform way of making these parallel implementations available along with the default serial ones. Currently, in PCL, this has been done by providing separate classes/functions having independent APIs. There exists no consistent way, to switch between the various implementations i.e. SIMD, OpenMP, CUDA, etc. without refactoring existing codebase.
Even adding new parallel implementations for existing algorithms isn't that straight forward.

With the integration of execution policies and executors (part of C++23 hopefully) libraries like PCL will be able to provide a unified API to switch between different implementations. To ensure there is no sudden breakage in the existing API, a transition phase is needed to slowly phase out the old API in favour of a unified API. This can be done currently by using function

overloading and tag-dispatching to achieve this, which will be forward compatible with the proposed API for executors.

This project aims to transition the existing API to forward-compatible unified API.
The basic details are:

1. Using tag dispatching to enable function overloading, allowing users to select between different available implementations.
2. Lack of tags allows PCL to choose the best possible implementations, this allows maintaining backward compatibility with current API.
3. Missing implementation for a tag raises compile-time errors
4. Providing SIMD/OpenMP/CUDA implementations of existing algorithms/functions using the proposed unified API

Components

The proposal is comprised of 3 parts:

1. **Unified API:**
   The main focus of this proposal is developing a uniform/unified API for accessing various parallel implementations as making the API forward compatible with executors (when they become available), ensuring the new API is available before availability of executors will provide sufficient time to downstream targets to update their existing codebase before the original functionality is deprecated.
2. **Parallel Implementations**
   Since the Unified API will make it easier to use and switch between various implementations, there will be an increase in need of having various implementations of algorithms/functions available so that the user can select the implementation suitable for their use case. Focus will be to add more implementations of more commonly used functions and then various algorithms.
3. **CI**
   As PCL moves forwards, there will be many changes in the design as well as implementation. Even during the GSoC phase there will be various major changes which lead to errors creeping into the library. By adding more features and increasing the coverage of the CI, will allow us to catch such errors before they get merged. It will also help increase the quality and stability of the codebase.

Proposal Timeline

Ramp-Up Phase

Motivation

Reading up extensively on executors and other approaches will help me get a better insight into what all has been done till now, their advantages and disadvantages so that I have a deeper understanding and have better discussions with mentors. Continuing fixing issues will also help me get more familiar with various areas of the codebase.
Since the approach we take will lay the foundation of how executors will work and be integrated with PCL, lots of discussions and approaches need to be analysed carefully with their merits and demerits. This is one of the most important stages.

Having proper direction is of utmost importance, instead of just haphazardly adding parallel implementations of various functions, a more organized approach is needed. Commonly used functions and their potential speedup needs to be analysed, so as to not waste time on functions which do not provide sufficient benefit to the community.

A good robust CI system is a must need for any large project, especially as we start work on modernization, unified API and bindings which will cause a lot of things to break and catching such issues would be very beneficial if we catch them before they get merged in.

Unified API

- Research on executors, CPU-GPU unified approaches, task-based parallelism.
- Work on building a suitable software architecture, taking inspiration from other projects which have a similar unified API.
- Extensive discussion with mentors and PCL community regarding approach and implementation details.

Parallel Implementations

- Identify commonly used functions and areas where they could be a significant speed improvement by parallelization.

CI

- Work on the basic matrix strategy for CI, to allow future scaling, especially as we integrate the unified API and python bindings
- Integrating tutorials into CI (#2672)
- Fix compiler warnings so that they can be treated as errors (#3379)
- Research on feasibility and applicability of integration tools like clang-tidy, static-analysers, sanitizers, etc in the CI.

Prototype Phase

Motivation

Mock-ups will help test and weed out any inconsistencies in the proposed API, making it more refined. Modules with fewer interdependencies can be transitioned to the new API first instead of the more complicated ones since it might lead to further complexities which could lead to delays, so identifying such modules is important before even beginning implementation else could lead to delays.

Very few modules have support for SIMD/OpenMP/CUDA even though most of the algorithms in PCL can be heavily parallelized since most operations are repetitive in nature providing a considerable speed boost. Making sure a standard interface will allow to avoid API breaks in the future.

Having a CI which can build and test with various build options will allow errors in the new API to be caught before merging. Various tools like clang-tidy, static-analysers, sanitizers will help write better code and catch edge cases.

Unified API

- Build simple mockups, get feedback and integrate suggested changes to API
- Identifying and classifying modules based on the amount of effort required i.e. having more interdependencies increases the effort needed.
- Create a timeline using the GitHub project board to keep track of ongoing, to be done and completed tasks.

Parallel Implementations

- Discuss and prioritize which implementations to go forward with first like SIMD, OpenMP, CUDA based on the time required and potential speedup.
- Make a standard PCL interface for various libraries such as AVX & SSE intrinsic, so that PCL API for downstream targets won't be affected due to API changes in these libraries.

CI

- Integrate feasible tool from the previous phase into CI
- Enable auto-generation of docker images using CI
- Research and discuss the integration of various build options e.g. SSE, OpenMP, etc., build types and libraries into CI without overloading the number of jobs.
- Discuss the design of unified tests for various implementations.

Implementation Phase

Motivation

Reducing time spent by CI will help add more build options and give more flexibility in the future to add more features to CI without overloading it. Currently, GPU modules are not part of CI having them is necessary especially as we integrate the unified API and more GPU implementations of various modules become available.

Unified API

- Laying down the common base framework of the decided API design and partition files accordingly.
- Start working on integrating API into low-effort modules.
- Discuss and come up with solutions for modules having complex dependencies such as GPU.
- Document work and add tests for the integrated modules.
- Write tutorials, design ideology and roadmap of new API

Parallel Implementations

- Implement parallel implementations of the modules using the frameworks discussed in the previous phase.
- Document and add tests for new implementations

CI

- Integrate various build option discussed in the previous phase into CI
- Add GPU modules to CI (#3586, #3575)

- Optimize and reduce time spent by CI by caching and using stages.

Extended Phase

Motivation

Even though most of the new API will automatically be tested by the new additions to the CI. There is still a need to thoroughly analyse the new API and ensure there are nor overlooked errors or breakage in API.

Unified API

- Integrate new API with remaining modules.
- Document work and add tests for the integrated modules.
- Extensive testing of all modules with new API besides already existing tests.

Parallel Implementations

- Continue adding more parallel implementations
- Document and add tests for new implementations

Stretch Goals

Motivation

The GPU module needs quite a bit of work and revamp. With the current trend of parallelization using GPU and its availability, this module plays an important role in the future. Especially as more deep learning approaches become popular people will look increasingly towards PCL for processing the data, so PCL should not be a bottleneck in the pipeline.

- Look into improving GPU/CUDA module support as well as extending functionality

Current Progress

1. Build a demo unified CPU & GPU interface for Euclidean Clustering.
   Sample Code: https://github.com/shrijitsingh99/pcl_clustering
   Modified Library: https://github.com/shrijitsingh99/pcl/tree/unified-api
2. Added SSE implementations of a few functions in the common module.
   Modified Library: https://github.com/shrijitsingh99/pcl/tree/sac_model_simd
3. Added matrix strategy (#3783) and convert CI to a multi-stage pipeline (#3795)
4. Resolved various issues (#3665, #3676, #3677, #3683, #3685, #3692, #3693, #3745, #3783, #3789, #3802, #3795, #3731)

Helpful Links

1. https://www.khronos.org/assets/uploads/developers/library/2016-supercomputing/SC16-compareSYCL-Michael-Wong_Nov16.pdf
2. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0443r1.html
3. http://open-std.org/JTC1/SC22/WG21/docs/papers/2018/p1055r0.pdf
4. https://github.com/chriskohlhoff/executors
5. http://chriskohlhoff.github.io/executors/concurrency.html
6. *Other unified task parallelization libraries*

Personal Details & Experience:

Name: Shrijit Singh
Email: shrijitsingh99@gmail.com
GitHub: https://github.com/shrijitsingh99/
GitLab: https://gitlab.com/shrijitsingh99
Website: https://shrijitsingh.com
LinkedIn: http://linkedin.com/in/shrijitsingh99/

Technical Summary

Languages & Tools: C, C++, Python, Swift, Embedded C, SQL, Javascript, HTML, CSS, Git, Jetbrains IDE, Xcode
Technologies: ROS, MATLAB, Tensorflow, Git, Gazebo, OpenCV, PCL, Numpy, Flask

Education

Manipal Institute of Technology - B.Tech in Computer Science Engineering (2017-2021)
Relevant Coursework: Graph Theory, Probability & Statistics, Linear Algebra (Current)

Work Experience

- **Project MANAS**
  Tech Head (Feb 2019 - March 2020)
  AI Member (Feb 2018 - Feb 2019)
  Led and managed a team of over 40 undergraduates as well as supervised all the projects being worked on. Collaborated and worked with various team members on several projects including an autonomous car, UGV and UAV.

  1. ***Mahindra Rise Prize - Self-Driving Car For Indian Conditions***
     • Co-designed and developed the entire software system architecture with consideration for performance and modularity • Currently developing a semantic segmentation, scene identification and obstacle tracking system using sparse LiDAR
     pointcloud
     • Built communication interface for the car's CAN gateway and serial communication framework
  2. ***IGVC 2018 & 2019 - Unmanned Ground Vehicle***

• Incorporated elastic band based motion planner into the system and improved the performance of A* based path planner using specialised scenario based heuristics
• Integrated sensors like INS, encoders and cameras into the localization system based on unscented kalman filters • Built and integrated a differential drive PID controller with Hercules MCU
• Created a simulation environment with accurate physics for extensive testing for software system

- **VTOL Aviation In Collaboration With IIT Kanpur**
  Intern May 2019 - July 2019
  • Integrated motor controls as well as remote controls of a quad-copter running FreeRTOS on a STM32
  • Built an onboard real-time data logging framework as well as a wireless data communication system for data logging
  and monitoring

Projects

- ***bROS***
  bROS is a high performance, modern and modular navigation stack built for ROS2. It is aimed towards general purpose robots which require easy to use and highly adaptable navigation systems.
- ***xDWA***
  A local planner which builds upon the Dynamic Window Approach adding ability to build paths which are globally more optimal by sampling at multiple future timesteps to approximate non-linear paths.

Project Links

1. Modular Navigation Framework for ROS2
   https://gitlab.com/groups/project-manas/ai/bROS
2. Intelligent Ground Vehicle Competition 2019 Software Stack
   https://gitlab.com/project-manas/ai/igvc_2019
3. Intelligent Ground Vehicle Competition 2018 Software Stack
   https://gitlab.com/project-manas/ai/igvc_2018
4. Extended Dynamic Window Approach Planner
   https://github.com/shrijitsingh99/xDWA
5. Vehicle Steering Driving Imitation
   https://github.com/Project-MANAS/driving_imitation
6. ROS Interface for VectorNav INS
   https://github.com/shrijitsingh99/vectornav
7. Fast Intersection & Road Detection
   https://github.com/shrijitsingh99/fastseg
8. Fast Range Image Based Lidar Clustering
   https://github.com/shrijitsingh99/run_based_segmentation
9. Self Driving Car Software Stack (Partly Hidden due to Ongoing Development)
   https://gitlab.com/project-manas/ai/eve

Other Links

1. Current Organization Codebase
   https://github.com/project-manas
   https://gitlab.com/project-manas/ai
2. Codebase
   https://github.com/shrijitsingh99/
   https://gitlab.com/users/shrijitsingh99/