Enable automatic differentiation of OpenMP programs with Clad

Mentors

- Vassil Vassilev
- David Lange

Contact

Name: Jiayang Li
Time Zone: UTC+8 (Shanghai)
Email: lijiayang404@gmail.com
Github: Errant404

Synopsis

Clad is an automatic differentiation (AD) clang plugin for C++. Given a C++ source code of a mathematical function, it can automatically generate C++ code for computing derivatives of the function. Clad is useful in powering statistical analysis and uncertainty assessment applications. OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared-memory multiprocessing programming in C, C++, and other computing platforms.

This project aims to develop infrastructure in Clad to support the differentiation of programs that contain OpenMP primitives.

Implementation Details

Determining the Scope of OpenMP Primitives Support

OpenMP includes many directives, which we categorize into high-priority and low-priority groups based on general expectations and typical usage.
To further refine and validate this categorization, we plan to review OpenMP-related simulation codes, such as ParFlow, to extract real-world usage patterns of OpenMP directives.

This analysis will help ensure that our implementation priorities are aligned with practical needs.

- High-priority primitives (initial selection):
  - `#pragma omp parallel for`,
  - `#pragma omp reduction`,
  - `#pragma omp critical`,
  - `#pragma omp atomic`
- Low-priority primitives:
  - `#pragma omp sections`,
  - `#pragma omp barrier`,
  - `#pragma omp task`

Parsing OpenMP Directives

Enhance Clad's `ASTVisitor` to capture nodes such as `OMPParallelForDirective`, enabling recognition and parsing of common OpenMP directives.

Variable Scope Analysis and Handling

Differentiate shared variables (`shared`), private variables (`private`), and reduction variables (`reduction`), generating corresponding differential variable mappings.

- Create thread-local copies for private variables.
- Synchronize gradient updates for shared variables (e.g., using atomic operations).
- Maintain the reduction logic in gradient computation for reduction operations.

OpenMP Directive Handling Strategy

For the automatic differentiation of OpenMP parallel code, we consider two computation strategies: Forward Mode and Reverse Mode, drawing inspiration from Enzyme's handling of fork/sync transformations at the LLVM level.

- Forward Mode: Generate differential "forward" code corresponding to the original function within each OpenMP parallel region. If intermediate values need to be stored (e.g., for subsequent reverse mode computations), thread-local storage should be used within each thread.
- Reverse Mode: Reverse the control flow. For example, fork operations in forward mode correspond to sync operations in reverse mode. Additionally, the local gradients from each thread must be correctly reduced to shared variables. This approach is similar to Enzyme's fork/sync transformation at the LLVM level.

During implementation, I will further discuss with my mentors and refer to existing tools (such as Enzyme) to ensure a reasonable and efficient approach to OpenMP directive automatic differentiation.

Related Work

Through extensive research, the currently available automatic differentiation tools that support OpenMP include:

- Tapenade (performs source-to-source transformation)
- Enzyme (works at the LLVM IR level)

In the actual development process, the above open source projects and the following papers may provide valuable reference or help.

- Source-to-Source Automatic Differentiation of OpenMP Parallel Loops
- Scalable Automatic Differentiation of Multiple Parallel Paradigms through Compiler Augmentation

Timeline

Community Bonding Period	
May 8 - June 1	• Gain an in-depth understanding of the code structure and logical details of the Clad project. • Clarify the specific handling strategies for various OpenMP primitives, preferably documented through tests or written documentation. Reference can be made to other projects that support OpenMP automatic differentiation, such as Enzyme. • Try to develop a basic infrastructure to parse and respond to pragma-based differentiation requests, serving both as a technical exploration and a practical starting point for developing broader pragma-based infrastructure in Clad. • Discussing these aspects and implementation details with the mentor to ensure the correctness of the implementation logic.
Coding Period	
Week 1 (June 2 - June 8)	Complete AST capture for OpenMP directives. **Deliverable:** Enhanced OpenMP pragma handling support
Week 2,3 (June 9 - June 22)	Support for basic OpenMP parallel directives. **Deliverable:** Automatic differentiation support for OpenMP `parallel` and `for` directives.
Week 4 (June 23 - June 29)	Complete variable scope analysis for OpenMP directives. **Deliverable:** Automatic differentiation support for

	OpenMP directives with `shared` and `private` variables.
Week 5 (June 30 - July 6)	Support for `reduction` directives in forward mode. **Deliverable:** Automatic differentiation support for OpenMP `reduction` directives.
Week 6 (July 7 - July 13)	Buffer time.
Week 7 (July 14 - July 20)	Midterm evaluation.
Week 8 (July 21 - July 27)	Add support for atomic operations, critical sections, etc in forward mode. **Deliverable**: Comprehensive support for high-priority primitives in forward mode.
Week 9,10 (July 28 - August 10)	Reverse mode. **Deliverable**: Reverse mode support for all previously implemented features.
Week 11 (August 11 - August 17)	Finalize testing. **Deliverable**: Comprehensive and well-developed test cases with high code coverage.
Week 12 (August 18 - August 24)	Document writing **Deliverable:** README content, tutorials, and documentation related to the OpenMP usage of Clad.
Week 13 (August 25 - August 31)	Buffer time.

Why Me?

I am a third-year student majoring in Computer Science at Shanghai University. As an open-source enthusiast, I have contributed to many open-source projects.
I participated in OSPP 2024, a program similar to Google Summer of Code (GSoC), which has made me familiar with GSoC's timeline.
I have extensive experience in the field of High-Performance Computing (HPC), making me proficient in the OpenMP programming model. Additionally, I have a solid understanding of the principles of automatic differentiation.
Currently, I also have a PR merged into Clad, which demonstrates my strong ability to get started with the project and my thorough understanding of it.

Candidate’s Other Commitments

The postgraduate recommendation system in China allows top undergraduate students to be admitted to master's programs without taking the national entrance exam, with the selection process typically occurring from September to October. Since I am considering pursuing a master's degree, this might overlap with the GSoC timeline, as I may take some universities' on-site written tests and interviews from June to September. However, I believe it won’t take up too much of my time. In fact, I am applying for GSoC precisely because I expect to have some free time during this period.

Why GSoC with Clad project in CERN HSF ?

I am interested in this project not only because I see it as a great opportunity to improve my programming skills and showcase my abilities but also because I deeply admire CERN. Even before my undergraduate studies, I had a strong passion for both physics and computer science. Although I eventually chose computer science as my major, my enthusiasm for physics has never faded. My active involvement in the field of HPC is a testament to this, as it represents the perfect way to combine my two greatest academic interests.

For someone like me, who is passionate about physics, CERN is nothing short of romantic. The thought that I might, in one way or another, contribute to the software used in CERN projects excites me immensely.