GSoC 2025 Project Proposal
CERN HSF
High Energy Physics Software Foundation

Implement and improve an efficient, layered tape with prefetching capabilities

Mentors

- Vassil Vassilev (vvasilev@cern.ch)
- David Lange (david.lange@cern.ch)

Personal Details

Aditi Milind Joshi
Computer Science and Engineering (Artificial Intelligence and Machine Learning) Undergraduate
Manipal Institute of Technology
Phone: +91 7977063540
Email: aditij0205@gmail.com
Github: github.com/aditimjoshi

PROJECT DETAILS

OVERVIEW

Automatic Differentiation (AD) is a computational technique that enables efficient and precise evaluation of derivatives for functions expressed in code. Unlike numerical differentiation, which suffers from approximation errors, or symbolic differentiation, which can be computationally expensive, AD systematically applies the chain rule to compute gradients with minimal overhead.

Clad is a Clang-based automatic differentiation tool that transforms C++ source code to compute derivatives efficiently. By leveraging Clang's compiler infrastructure, Clad performs source code transformations to generate derivative code for given functions, enabling users to compute gradients without manually rewriting their implementations. It supports both forward-mode and reverse-mode differentiation, making it useful for a range of applications.

A crucial component for AD in Clad is the tape, a stack-like data structure that stores intermediate values for reverse mode AD. This project aims to optimize and generalize the Clad tape to improve its efficiency, introduce multilayer storage, enhance thread safety, and enable CPU-GPU transfer.

TAPE

Reverse-mode AD computes gradients efficiently for scalar outputs with many inputs, and this process requires saving intermediate values during the forward pass that will later be used in the backward (gradient) pass. The tape is essentially a stack-like data structure designed to record and store these intermediate values and the sequence of operations performed on them. This enables the exact replay of operations in reverse during gradient computation. The tape follows a first-in, last-out (FILO) pattern which aligns naturally with the way reverse-mode AD works—later operations in the forward pass are reversed first.

In Clad's implementation, the tape is typically realized as a monolithic memory buffer where values and metadata are sequentially stored. It is designed to be lightweight and efficient for small to moderate computations, but it can become a bottleneck in memory-intensive or parallel workloads. Therefore, the tape's structure and efficiency directly impact the scalability, performance, and memory usage of reverse-mode differentiation in Clad. Enhancing the tape to support features like thread-safety, checkpointing, multi-level memory hierarchy (RAM/disk), and heterogeneous device support (e.g., CPU-GPU transfers) can significantly improve the robustness and applicability of Clad in real-world scientific and high-performance computing environments.

DELIVERABLES

1. Optimize the current tape by avoiding re-allocating on resize in favor of using connected slabs of array
2. Enhance existing benchmarks demonstrating the efficiency of the new tape
3. Add the tape thread safety
4. Implement multilayer tape being stored in memory and on disk
5. [Stretch goal] Support cpu-gpu transfer of the tape
6. [Stretch goal] Add infrastructure to enable checkpointing offload to the new tape
7. [Stretch goal] Performance benchmarks

IMPLEMENTATION PLAN

- **Task 1: Optimize the current tape by avoiding re-allocating on resize in favor of using connected slabs of array and implement small buffer optimization**

  The current implementation of the tape in Clad uses a contiguous dynamic array. Each time a new entry is pushed onto the tape and the underlying capacity is exceeded, the vector grows by allocating a new larger block of memory, copying all existing entries to the new block, and deallocating the old block.

  The current tape implementation dynamically resizes its storage using a growth factor of 2x when capacity is exceeded. This leads to expensive memory reallocation and copying overhead, significantly affecting performance in large-scale differentiation tasks.

Code for current implementation of function to resize tape:

```
CUDA_HOST_DEVICE void grow() {
  if (!_capacity)
    _capacity = _init_capacity;
  else
    _capacity *= 2;

  T* new_data = AllocateRawStorage(_capacity);
  MoveData(begin(), end(), new_data);
  destroy(begin(), end());
  ::operator delete(_data);
  _data = new_data;
}
```

  Instead of reallocating memory, we propose using a slab-based memory allocation strategy. This involves allocating connected memory chunks (slabs) and linking them dynamically as the tape grows, reducing unnecessary reallocations.

Additionally, to further optimize performance for small-scale or short-lived tapes, we introduce a small buffer optimization (SBO) as part of the design. With SBO, a small statically allocated buffer is embedded directly inside the tape object—usually on the stack. Only when this buffer overflows does the system transition to heap-allocated slabs. During the transition, the contents of the small buffer are copied into the first slab, and subsequent entries are written to the slabs as usual. This optimization significantly reduces the memory allocation overhead for small differentiation tasks, allowing them to complete faster and with lower memory footprint.

Example code for proposed structure of tape:

```
struct Slab {
   T* data;
   Slab* next;
};
Slab* head;
Slab* current;
size_t slab_size;

static constexpr size_t SBO_CAPACITY = SBOSize;
T sboBuffer[SBO_CAPACITY];
```

- **Task 2: Add Tape Thread Safety**

  The current implementation of the tape in Clad is designed for sequential execution, assuming that all differentiation and tape operations occur in a single-threaded context. As a result, it does not include concurrency primitives such as locks or atomic operations, and the tape is not thread-safe. In a sequential execution, operations like pushing values onto the tape and popping them during the reverse pass occur in a deterministic order without interference, so mutual exclusion is not required.

  However, in multithreaded scenarios, the tape would act as a globally shared resource accessed by multiple threads simultaneously. Without synchronization, concurrent reads and writes could lead to race conditions, corrupting the tape state or causing incorrect gradients. To make the tape thread-safe, we propose integrating a locking mechanism, such as a **std::mutex**. By acquiring a lock before any modification or reading of the tape structure, we can ensure mutual exclusion, maintaining the consistency and correctness of the tape. This change is essential for enabling parallel reverse-mode AD, particularly when checkpointing or executing independent gradient segments across threads.

- **Task 3: Implement Multi-Layer Tape Stored in Memory and Disk**

Currently, the Clad tape implementation stores all intermediate values entirely in system memory and reuses them during the reverse pass to compute gradients. As the number of operations grows (for large models or long sequences), so does the size of the tape. While memory (RAM) is fast, it's also limited. To scale AD beyond memory limits, we can offload older or less frequently accessed portions of the tape to disk, like how operating systems page memory.

This leads to the idea of a multilayer tape where recent entries stay in memory and older entries are paged to the disk. The idea is to treat the tape like a hybrid memory buffer similar to LRU caching where slabs are evicted to disk when memory exceeds a threshold.

Example code for storing multilayer tape in memory and disk:

```
size_t memoryLimit;
std::unordered_map<int, T> memoryCache;
std::list<int> lruList;
std::string diskStoragePath;

// Evict the least recently used item to disk
void evictToDisk() {
  int lruKey = lruList.back();
  lruList.pop_back();

  std::ofstream outFile(diskStoragePath + "/" + std::to_string(lruKey) + ".tape", std::ios::binary);
  T& value = memoryCache[lruKey];
  outFile.write(reinterpret_cast<char*>(&value), sizeof(T));
  outFile.close();

  memoryCache.erase(lruKey);
}

// Load an item from disk if it's not in memory
T loadFromDisk(int key) {
  T value;
  std::ifstream inFile(diskStoragePath + "/" + std::to_string(key) + ".tape", std::ios::binary);
  inFile.read(reinterpret_cast<char*>(&value), sizeof(T));
  inFile.close();
  return value;
}
```

- **Task 4: Support CPU-GPU Transfer of the Tape**

  Currently, the Clad tape operates entirely in host (CPU) memory, with no provisions for allocating or accessing data on device (GPU) memory. Once the slab-based tape from Task 1 is implemented, where the tape is made up of connected slabs of memory instead of relying on frequent reallocations, it becomes significantly easier to manage contiguous

memory chunks that are friendly for memory transfer to the GPU. These slabs can be treated as blocks of data, allocated using **cudaMalloc()** and copied between the CPU and GPU using standard memory transfer operations like **cudaMemcpy()**.

- **Task 5: Add Checkpointing Offload to New Tape**

  Checkpointing in the context of reverse mode automatic differentiation (AD) refers to the strategic storage of intermediate program states during the forward pass, so that during the reverse pass, one can recompute parts of the program efficiently without storing all intermediate values. It's a trade-off between memory and computation—reducing memory usage by allowing parts of the tape to be discarded and recomputed later if needed.

  To implement this, in the forward pass store only certain intermediate values at the checkpoints and during the reverse pass recompute the intermediate values from the nearest checkpoint.

PERFORMANCE ANALYSIS

As part of the community bonding period, we will use an example application where the tape plays a critical role in performance. A suitable candidate is the lulesh-1.0 branch in the Clad fork, specifically the lulesh.cu file located in the benchmark/LULESH directory. Toward the end of this file, derivative computations are invoked using Clad's automatic differentiation. We will use this application as a testbed to benchmark the current tape implementation and compare it with the proposed implementation.

TIMELINE

Community Bonding Period	
May 8 - June 1	Engage with the community. Set up the development environment.
Coding period begins	
Week 1	Modify the structure of the tape to use slabs of arrays and a small buffer instead of a contiguous dynamic memory block and make changes in the **grow()** and **AllocateRawStorage()** functions in the **Tape.h** file. **Deliverable:** Optimize the current tape by avoiding re-allocating on resize in favor of using connected slabs of array.
Week 2	Enhance existing benchmarks to demonstrate the efficiency of the new tape.

Week 3	Add locking mechanisms to the modification and reading operations on tape to ensure tape thread safety. **Deliverable:** Add the tape thread safety.
Week 4	Add tests for tape thread safety.
Week 5	Implement LRU algorithm for offloading segments of the tape to disk. **Deliverable:** Implement multilayer tape being stored in memory and on disk.
Week 6	Buffer Week
Midterm Evaluations	
Week 8	Implement functions to allocate memory on GPU and facilitate transfer of tape from CPU to GPU and vice-versa. **Deliverable:** Support cpu-gpu transfer of the tape
Week 9	Implement checkpointing mechanism to recompute values during reverse pass into the tape structure and modify the functions in **Tape.h** file to store only the checkpointed values instead of all intermediate values. **Deliverable:** Add infrastructure to enable checkpointing offload to the new tape.
Week 10	Performance benchmarks
Week 11	Buffer Week
Week 12	Extended testing, developing documentation, presenting the work.

BACKGROUND AND MOTIVATION

Regarding my educational background, I am a third year B' Tech undergraduate student studying Computer Science and Engineering (Artificial Intelligence and Machine Learning) at Manipal Institute of Technology. Over the past two years, I've been an active member of my university's official AI and robotics team, Project Manas, where I've worked on autonomous systems and AI projects that exposed me to concepts such as automatic differentiation, parallel programming, and optimization.

I became particularly interested in automatic differentiation during my exploration of how neural networks work internally. This led me to build a neural network from scratch using CUDA C++, where I handled both forward and backward passes on the GPU. While working on this, I discovered the CLAD repository, and its focus on bringing efficient automatic differentiation to C++ immediately resonated with my interests.

To get involved, I've already raised issue #1327 as part of the evaluation process and was working on a pull request for issue #1274, which involves fixing an incorrect differentiation result involving std::pair.

Beyond AD, I have prior experience with Clang/LLVM through my work on a compiler project based on the Kaleidoscope tutorial. I built a language frontend and developed an understanding of how IR generation and code transformation work.

I found the project "Implement and improve an efficient, layered tape with prefetching capabilities" particularly exciting, as it aligns closely with my interests in optimization, memory-efficient design, and parallel computing. I believe contributing to this project will help me gain a deeper understanding of compiler-assisted program transformation which I can carry further into other projects and bring real performance benefits to scientific computing workloads.

AVAILABILITY

As for my availability, I have no conflicting academic or professional commitments during the summer. I am fully available and can dedicate the required 20–25 hours per week (or more, if needed) to the project throughout the duration of GSoC. I am comfortable communicating over Email, or Zoom, and I'm happy to adapt to the mentors' preferred communication platform. I will remain responsive and will make it a point to regularly share updates and discuss my doubts with the mentors.