Enhance Taskflow's Pipeline Infrastructure

Abstract

I would like to apply for the **Taskflow: Enhance Pipeline Infrastructure** project. This project is to add a layer of data abstraction on top of taskflow's original pipeline, so that users can either choose to prepare the data buffer manually and use an efficient pipeline without data abstraction, or choose to use a pipeline with data abstraction and let the program allocate the buffer automatically.

Introduction to taskflow

> Parallel programming has advanced many of today's scientific computing projects to a new level. However, writing a program that utilizes parallel and heterogeneous computing resources is not easy because you often need to deal with a lot of technical details, such as load balancing, programming complexity, and concurrency control. Taskflow streamlines this process and helps you quickly create high-performance computing (HPC) applications with programming productivity.

> Taskflow has introduced a new task-parallel pipeline programming framework in v3.3. The current pipeline design is primitive and does not provide any data abstraction. For many data-centric pipeline applications, this can be inconvenient as users need to repetitively create data arrays. Therefore, the goal of this project is to derive a pipeline class with data abstraction to streamline the implementation of data-centric pipeline applications.

> We rate this project a difficult level of difficulty, since it involves both implementation and algorithm challenges. However, participants in this project will not just learn how to implement a real module atop Taskflow but also gain practical research knowledge about parallel scheduling algorithm.

Why make such a pipeline

- Pipeline is a fundamental parallel pattern to model parallel executions through a linear chain of stages. Each stage processes a data token after the previous stage, applies an abstract function to that data token, and then resolves the dependency for the next stage. Multiple data tokens can be processed simultaneously across different stages whenever dependencies are met.
  As modern computing applications continue to adopt pipeline parallelism in various forms, there is always a need for new pipeline programming frameworks to streamline the implementation complexity of pipeline algorithms.

- The current taskflow pipeline is efficient, but we need a more user-friendly pipeline. The current taskflow pipeline is not equipped with a data abstraction layer and requires the user to manually allocate a data buffer before it can work.
  A good example of this is the following program.

  ```
  const size_t num_lines = 2;
  // input data
  std::vector<std::string> input = {
    "abade",
    "ddddf",
  ```

```
  "eefge",
  "xyzzd",
  "ijjjj",
  "jiiii",
  "kkijk"
};
// custom data storage
using data_type = std::variant<
  std::string, std::unordered_map<char, size_t>, std::pair<char, size_t>
>;
std::array<data_type, num_lines> buffer;
// the pipeline consists of three pipes(serial-parallel-serial)
// and up to two concurrent scheduling tokens
tf::Pipeline pl(num_lines,
  // first pipe processes the input data
  tf::Pipe{tf::PipeType::SERIAL, [&](tf::Pipeflow& pf) {
    if(pf.token() == input.size()) {
      pf.stop();
    }
    else {
      buffer[pf.line()] = input[pf.token()];
      printf("stage 1: input token = %s\n", input[pf.token()].c_str());
    }
  }},
  // second pipe counts the frequency of each character
  tf::Pipe{tf::PipeType::PARALLEL, [&](tf::Pipeflow& pf) {
    std::unordered_map<char, size_t> map;
    for(auto c : std::get<std::string>(buffer[pf.line()])) {
      map[c]++;
    }
    buffer[pf.line()] = map;
    printf("stage 2: map = %s\n", format_map(map).c_str());
  }},
  // third pipe reduces the most frequent character
  tf::Pipe{tf::PipeType::SERIAL, [&buffer](tf::Pipeflow& pf) {
    auto& map = std::get<std::unordered_map<char, size_t>>
(buffer[pf.line()]);
    auto sol = std::max_element(map.begin(), map.end(), [](auto& a, auto& b)
{
      return a.second < b.second;
    });
    printf("stage 3: %c:%zu\n", sol->first, sol->second);
  }}
);
```

In the above code, the user needs to manually prepare a `std::array<data_type, num_lines> buffer` before using taskflow's pipeline, and needs to use the buffer to read and write data in the function corresponding to each `pipe`.
This is a very cumbersome thing for many data-centric programs, so we want to add a layer of data abstraction to allow the program to allocate buffer efficiently.

- There is currently no efficient data abstraction pipeline, and this design is valuable to explore.
  The dominant pipeline parallel model framework on the market is intel oneTBB, but the existing taskflow pipeline without data abstraction is more efficient than it. For example, taskflow's pipeline outperforms oneTBB 24% and 10% faster in a VLSI placement and a timing analysis workloads that adopt pipeline parallelism to speed up runtimes, which is evaluated in this paper.
  Furthermore, oneTBB only provides pipelines with data abstraction, whereas taskflow plans to provide both pipelines, with and without data abstraction, giving the user more choice and more scope for optimising the performance of the program.
  From an academic point of view, this is also an interesting topic. The allocation of data abstraction should not be arbitrary, otherwise when two threads read similar data in the same address at the same time, there will be false sharing, making the parallel program serialised and less efficient. Therefore, we need to find a smart way of allocating data abstraction that avoids false sharing as much as possible.

Feasibility of the task

- The programming environment for taskflow is already fully set up, and the examples, tests and benchmarks in the repository and my own modifications are all working properly.

- I have read the existing taskflow source code and related paper Pipeflow: An Efficient Task-Parallel Pipeline Programming Framework using Modern C++ carefully and have a deep understanding of the system.

- I have had several video meetings with my supervisors to discuss ideas and tasks. The technical details below are the result of many comunications with my supervisors.

Technical Details

- First, we give an example of the use of the data abstraction after it has been added, which will be our design goal. This code block corresponds to the code block above. A comparison of the two shows the convenience of data abstraction.

```
// input data
std::vector<std::string> input = {
  "abade",
  "ddddf",
  "eefge",
  "xyzzd",
  "ijjjj",
  "jiiii",
  "kkijk"
};
// the pipeline consists of three pipes (serial-parallel-serial)
// and up to four concurrent scheduling tokens
tf::Pipeline pl(num_lines,
  // first pipe processes the input data
  tf::Pipe<void, std::string>{tf::PipeType::SERIAL, [&](tf::Pipeflow& pf) {
    if(pf.token() == input.size()) {
      pf.stop();
    }
    else {
      printf("stage 1: input token = %s\n", input[pf.token()].c_str());
      return input[pf.token()];
    }
  }},

  // second pipe counts the frequency of each character
  tf::Pipe<std::string, std::unordered_map<char, size_t> >
{tf::PipeType::PARALLEL, [&](std::string str) {
    std::unordered_map<char, size_t> map;
    for(auto c : str) {
      map[c]++;
    }
    printf("stage 2: map = %s\n", format_map(map).c_str());
    return map;
  }},

  // third pipe reduces the most frequent character
  tf::Pipe<std::unordered_map<char, size_t>, void>{tf::PipeType::SERIAL, []
(std::unordered_map<char, size_t> map) {
    auto sol = std::max_element(map.begin(), map.end(), [](auto& a, auto& b)
{
      return a.second < b.second;
    });
    printf("stage 3: %c:%zu\n", sol->first, sol->second);
  }}
);
```

- The most basic design is to allocate a buffer of the corresponding size based on `num_lines` and the input data type when the pipeline class is created. This is a simple design, and the space allocated is contiguous, so cache utilisation is high as the pipeline reads data continuously. However, false sharing at the parallel pipe level can be very serious, making parallelism almost impossible and performance poor.

- Therefore, to avoid false sharing, we came up with an alternative design, i.e., giving every line a buffer of a multiple of the cache line size. As every line corresponds to a process, we can avoid false sharing at the parallel pipe level, as the data from every line can fill up the cache of the corresponding process. However, the cache will be updated every time the serial pipe reads data, and the cache hit rate will be very low.

- At this point, we will have implemented two designs of data abstraction, one with high cache utilisation and severe false sharing, and one that avoids false sharing but has very low cache utilisation.
  We can start writing some unit tests to ensure that the pipeline with data abstraction is correct and easy to use, and to prepare for the subsequent implementation of a high performance data abstraction.

- As each of the first two approaches has its own strengths and weaknesses, we wanted to combine the first two approaches to explore a more efficient way of designing the data abstraction to improve the performance of the pipeline.
  This part needs to be designed carefully, so more reading is needed, such as the oneTBB source code, and some related papers such as On reducing false sharing while improving locality on shared memory

multiprocessors, Deterministic Scale-Free Pipeline Parallelism with Hyperqueues, etc. The exploration of ideas in this section is planned to take place before the summer holiday.

- Use the already implemented high performance taskflow pipeline with data abstraction to implement the ferret and dedup benchmarks of the PARSEC suite, which were designed for pipeline parallelism. Considering there is a lot of open source code on the internet for implementing PARSEC using oneTBB and the usage of the taskflow pipeline we designed is similar to oneTBB, this part is easy.

Schedule of Deliverables

We will divide the task into five points in the TECHNICAL DETAILS and complete the task in the corresponding phase.

Community Bonding Period May 20 - June 12

- This period will be used to read about some resources of data abstraction, such as oneTBB source code and related papers, in order to improve my understanding of pipeline model optimisation and false sharing, and in turn to come up with creative data abstraction design solutions.
- I will continue to keep in touch with the community and mentor to discuss efficient design solutions.
- GSoC needs us to keep blogging about the progress of the open source project. As I have already built my own website, I will send links to the community in the meantime.

Phase 1 June 13 - June 25

- This is the time to start the formal work of writing the code, implementing the most basic data abstraction.
- When the pipeline class is created, a buffer is allocated based on `num_lines` and the input data type. It is worth noting that in this step we need to get all the data types that appear in the pipeline and determine the size of each one to allocate the buffer from.
  For the moment, we are considering using `std::variant` to implement the need to store different data types at the same address.
- In addition to this, the `_on_pipe` function of the pipeline needs to be modified. The original `_on_pipe` function is as follows.

  ```
  void Pipeline<Ps...>::_on_pipe(Pipeflow& pf, Runtime&) {
    visit_tuple([&](auto&& pipe){
      pipe._callable(pf);
    }, _pipes, pf._pipe);
  }
  ```

  The `callable` function takes only one argument `pf`. It is because originally there was no data abstraction, the user prepares the buffer himself and writes explicit data manipulation in the function that is passed to `pipe`, for example

```
tf::Pipe{tf::PipeType::PARALLEL, [&](tf::Pipeflow& pf) {
  std::unordered_map<char, size_t> map;
  for(auto c : std::get<std::string>(buffer[pf.line()])) {
    map[c]++;
  }
  buffer[pf.line()] = map;
  printf("stage 2: map = %s\n", format_map(map).c_str());
}},
```

However, with the addition of the data abstraction, the signature of the function passed into the pipe changes, so `_on_pipe` also needs to be adjusted.

- milestones: The result of this stage is that the code sample above works correctly.

Phase 2 July 25 - Aug 7

- Implement unit tests for pipelines with data abstraction, as described in `unittests/pipeline.cpp`, which is available in the repository.
- Another design option is to give every line a buffer that is a multiple of the cache line size. This solution is different from the previous one in that it allocates a different buffer size, and we need to consider how to get the cache line size for the running device. The rest of the design is basically the same as the previous one.
- Use oneTBB to implement two benchmarks in PARSEC, ferret and dedup, and get the performance data
- milestones: unitests work with both data abstraction solutions

Phase 3 Aug 7 - Aug 27

- Use the current pipeline with data abstraction to implement two benchmarks in PARSEC, ferret and dedup, and get the performance data
- Implementing an efficient data abstraction based on the first two design options
- Run PARSEC benchmarks with the redesigned pipeline and evaluate them against oneTBB
- milestones: the pipeline can run the PARSEC benchmarks successfully and efficiently

Phase 4 Aug 27 - Sept 12

- Time to complete what was not finished or was not done good enough.
- Explore additional pipeline using scenarios and implement them with the taskflow pipeline.
- milestones: complete all the tasks listed in the technical details.

Development Experience

This is my first time to participate in an open source community project and I have no previous experience in open source projects. I hope that through this GSoC project, I can get into the open source community and

experience the joy of writing code with different people in the open source community. It will help me a lot in my future career development. This is why I wanted to participate in GSoC during the summer.
I do, however, have relevant experience in the field of parallel computing.

- I have taken courses in parallel computing related to the task at university.
  Here is the related code that I have written, which is assignments for the Introduction to High Performance Computing class at Tsinghua University, covering parallel programming techniques such as MPI, OpenMP and CUDA.
  The assignments includes using MPI to complete odd even sort, using OpenMP to accelerate breadth-first search using OpenMP, and matrix multiplication using CUDA to utilize GPU.

- Currently, I am also participating in the undergrauate research on parallel computing, which is about optimising large-scale recommender systems.

- I am participating in ISC 2022, a student cluster competetion, this semester, and I am responsible for the coding challenge part, where the main task is to convert xcompact3d, a fan airflow simulation program, from blocking communication to non-blocking communication.

Therefore, I have experience in the parallel computing field and I am able to do a good job on the taskflow parallel pipeline project.

Other Experiences

I am a sophomore undergraduate in the Department of Computer Science and Technology at Tsinghua University.
I entered the university with 7th rank in the university entrance exam of my province (about the top 0.001% of all those taking the exam), and currently maintain a GPA of 3.8/4.0. Therefore, I suppose I have sufficient ability to complete the program I have chosen.
I have a great passion for learning new things on my own. In addition to studying the school curriculum, I enjoy taking various open courses, including computer, maths, physics, philosophy classes and so on.
Some of the computer and related maths classes I have taken outside of school are (in chronological order)

- MIT Structure and Interpretation of computer programs
  This course was the first computer course I started taking during the summer before entering the university. In this course, using the lisp language, I learned a lot about elegant abstractions of functional programming, such as Computational Processes, Higher-order Procedures, Streams and Infinite Data Structures, Object- oriented Programming, Meta-linguistic Abstraction and many more. It led me to the field of computer programming.

- Berkeley cs61a, Structure and Interpretation of Computer Programs
  After entering university, I took berkeley's SICP, cs61a. In this course, I reviewed what I had learnt over the summer and implemented it concretely, and also learnt a lot about the techniques of using python.

- Stanford cs106x, Programming Abstractions in C++
  In China, most schools get students started with c++. To increase my level of using c++, I took this course and gain a deeper understanding of c++ syntax, recursion, etc. and was able to use c++ to solve specific problems. In my programming class in the first semester of my freshman year at Tsinghua, I used c++ to design a tower defence game imitating Kingdom Rush as a project for this class.

- Stanford cs106l, Standard C++ Programming
  Since stanford's cs106x uses stanford's own library to help students get started with c++, and does not expose students to c++ as it is used in real life, I took this cs106l course. In this course I was introduced to modern c++ language features including Streams, Templates, Functions and Lambdas, Move Semantics, Smart Pointers and more, which were later covered in the OOP course at Tsinghua University.

- Berkeley cs61b, Data Structures
  This was my introductory data structures course, learning basic data structures and algorithms. Later, I also learned more advanced data structures and algorithms in my data structures class at Tsinghua University, such as splay tree, red black tree, kd-tree, segment-tree, shell sort, KMP, etc.

- Berkeley cs170, Efficient Algorithms and Intractable Problems
  This course is about concept and basic techniques in the design and analysis of algorithms; models of computation; lower bounds; algorithms for optimum search trees, balanced trees and UNION-FIND algorithms; numerical and algebraic algorithms; combinatorial algorithms. Turing machines, how to count steps, deterministic and nondeterministic Turing machines, NP-completeness.

- Berkeley cs188, Introduction to AI
  This is my introductory course to artificial intelligence. This course introduces the basic ideas and techniques underlying the design of intelligent computer systems with a specific emphasis on the statistical and decision-theoretic modeling paradigm.

- Berkeley cs61c, Great Ideas in Computer Architecture
  This was my introduction to computer architecture. I learned how high-level language code is turned into assembly code step by step, and then into binary strings of numbers that reach the CPU for execution, and also learned about basic concepts such as CPU, cache, and virtual memory, et al. The most exciting part of the course was the third project, which is to implement a 2-stage pipeline CPU that could execute RISCV assembly code.

- MIT 6.s081, Operating Systems
  After learning cs61c, I wanted to learn more about operating systems, so I chose to start learning 6.s081, but since I have been delayed by research and some other things, I have only finished the virtual memory section so far.

- MIT 6.S191, Introduction to Deep Learning
  In this course, I was exposed to deep learning with applications to computer vision, natural language processing, biology, etc. I gained foundational knowledge of deep learning algorithms and get practical experience in building neural networks in TensorFlow.

- Berkeley cs267, Applications of Parallel Computers and Stanford cme213, Introduction to parallel computing using MPI, openMP, and CUDA
  At the beginning of my sophomore year, I vaguely found myself fascinated by the concept of "computation". In my opinion, computation is a predictive method that can be generalised, where we can model problems in the real world mathematically and use computers to compute large scale data to get amazing results. I was so excited by this conception that I decided to join Tsinghua's High Performance Computing Lab to give it a try, and so I began my studies in parallel computing.
  These two courses were designed to teach students how to program parallel computers to efficiently solve challenging problems in science and engineering, where very fast computers are required either

to perform complex simulations or to analyze enormous datasets. I am still doing undergraduate research in the field of parallel computing.

- Harvard am205, Advanced Scientific Computing: Numerical Methods
  Having encountered some obstacles in the parallel matrix decomposition section of cs267, I decided to learn something about numerical computation. This course taught me the basics of numerical computation.

- Berkeley eecs151, Introduction to Digital Design and Integrated Circuits
  Since high performance computing also involves how to use hardware efficiently, I wanted to gain an understanding of the hardware field. This course allowed me to learn more about computer hardware in depth, and I also learned to write FPGA code in verilog. This semester at Tsinghua I also took a Digital Design course and teamed up with a senior student to implement a multi-launch cpu with a 5-stage pipeline.

Why this project?

The most immediate reason I chose this project was because I wanted to gain a better understanding of pipeline parallelism through this project. As I mentioned in the previous part of the proposal, I think pipeline parallelism is a valuable research topic that can be applied to many areas, including taskflow's motivation source , which is timing analysis, and the recent hot topic like neural network training.
If we talk about the deeper reasons, I think it is the same reason why I chose to explore the field of parallel computing, that I like the concept of "computation". I like the idea of "computation" and taskflow pipeline is a framework that allows people to use computer to compute efficiently, so I chose this project.

Appendix

check points

- 5pts Have you communicated with the organization's mentors?
  Yes, I have several video meeting with my mentor.

- 5pts Have you communicated with the community?
  Yes, I have communicated with the Taskflow community.

- 5pts Did you reference projects you coded WITH links to repos or provided code?
  This is my github page.

- 5pts Did you provide several methods to contact you? (email, skype, mobile/phone, twitter, chat, and/or tumblr if available)
  My email is xiongz462@gmail.com or xiongzc20@mails.tsinghua.edu.cn
  My phone is +86-18755314010, which can also be used to find me on WhatsApp and Telegram

- 3pts Did you include a preliminary project plan (before, during, after GSoC)?
  Yes, I wrote it in my proposal.

- 3pts Did you state which project you are applying for and why you think you will end up completing the project?
  Yes, I would like to apply for the Enhance Pipeline Infrastructure project and I stated my plan and ability to complete the project in the proposal.

- 3pts Do you have time for GSoC? This is a paid job! State that you have time in your motivation letter, and list other commitments!
  Yes, I do have time. If I was selected to participate in GSoC, this will be my only task during my summer. (Maybe I will watch some open courses at the same time, but that is flexible and I will do that on the condition that I can perfectly complete the GSoC task.)
- 1pts Did you add a link to ALL your application files to a cloud hoster like GitHub or Dropbox? (easy points! 😉)
  Yes.
- 0pts Be honest! Only universal Karma points. 🙂
  I am.
- 5pts Did you create a pull request on the existing code?
  Currently no, because I have to design carefully before I begin to implement. But to demonstrate the feasibility of the task, I have discussed several time with my mentors and I have the ability to complete that.
- 5pts Did you continue communication until accepted students are announced?
  I will.