Course Project: Deadline-1

ArgoLib: High Performance C/C++ Parallel Runtime for fork/join Parallelism

Due by 11:59pm on 9th October 2022
(Total **20%** weightage)

Instructor: Vivek Kumar

No extensions will be provided. Any submission after the deadline will not be evaluated. If you see an ambiguity or inconsistency in a question, please seek clarification from the teaching staff.

Plagiarism: **This is a pair programming-based project that you must do with the group member that you have already chosen. No change to the group is allowed. You are not allowed to discuss the approach/solution outside your group.** You should never misrepresent some other group's work as your own. In case any plagiarism case is detected, it will be dealt as per the new plagiarism policy of IIITD **and will be applied to each member in the group.**

General Instructions

a) Hardware requirements:
    a) You will require a Linux/Mac OS to complete this project. For your final experimentations, we will provide you access to a 32-core / 20-core processor running Ubuntu OS.
b) Software requirements are:
    a) C++11 compiler and GNU make.
    b) You **must** do version controlling of all your code using github. You should only use a **PRIVATE** repository. If you are found to be using a **PUBLIC** access repository, then it will be considered plagiarism. NOTE that TAs will check your github repository during the demo.

Requirements

1) This is a group project where each group comprises a maximum of **two** students. You cannot change your group member once the project has started.
2) You may choose to work as a single member group, but there will not be any relaxations in marking scheme/deadlines, and the same rubric will be followed for each group.
3) We are going to use deadline chaining in this course project, where the runtime built for the first deadline will be the input for the next deadline. Hence, don't miss any deadline, as in that case you would miss the next deadline automatically.

Project Details

This deadline has two parts as described below. There is a bonus component as well as described towards the end of this document.

Part-1: Implementing a Simple ArgoLib Runtime
Weightage: 8%

Linguistic Interfaces

You will have to build a library-based frontend for the ArgoBots parallel runtime called as “ArgoLib”. Requirements for this library are as follows:

1) ArgoLib supports both C & C++ APIs in a single runtime for writing task parallel program.

Note the below declarations used in the ArgoLib APIs:

```
typedef ABT_thread Task_handle;
typedef void (*fork_t)(void* args);
```

The C APIs are as follows:

```
/**
 * Initializes the ArgoLib runtime, and it should be the first thing to call in the user main.
 * Arguments “argc” and “argv” are the ones passed in the call to user main method.
 */
void argolib_init(int argc, char **argv);

/**
 * Finalize the ArgoLib runtime, and performs the cleanup.
 */
void argolib_finalize();

/**
 * User can use this API to launch the top-level computation kernel. This API
 * gathers statistics for the parallel execution and reports after the completion
 * of the computation kernel. Some of the statistics are: execution time, total
 * tasks created, etc. This top-level kernel would actually be launching the recursive tasks.
 */
void argolib_kernel(fork_t fptr, void* args);

/**
 * Creates an Argobot ULT that would execute a user method with the specified argument.
 * It returns a pointer to the task handle that would be used for joining this ULT.
 */
Task_handle* argolib_fork(fork_t fptr, void* args);

/**
 * Used for joining one more ULTs using the corresponding task handles. In case of more than one
 * task handles, user can pass an array of Task_handle*. The parameter “size” is the array size.
 */
void argolib_join(Task_handle** list, int size);
```

The C++11 APIs are as follows. “T” is template parameter below.

```
/**
 * C++ equivalent of the argolib_init API.
 */
void argolib::init(int argc, char **argv);

/**
 * C++ equivalent of the argolib_finalize API.
 */
void argolib::finalize();

/**
 * C++ equivalent of the argolib_kernel API. User lambda passed instead of function pointer and args.
 */
void argolib::kernel(T &&lambda);

/**
 * C++ equivalent of the argolib_fork API. User lambda passed instead of function pointer and args.
 */
Task_handle* argolib::fork(T &&lambda);

/**
 * C++ equivalent of the argolib_join API.
 * All the task handles are passed as comma separated list. The three dots “…” represents variable
 * number of arguments.
 */
void argolib::join(T… handles);
```

Example of Parallel Fibonacci using ArgoLib

```
#include “argolib.hpp”

int fib(int n) {
        if(n<2) return n;
        int x, y;
        Task_handle* task1 = argolib::fork([&]() {    x = fib(n-1);    });
        Task_handle* task2 = argolib::fork([&]() {    y = fib(n-2);    });
        argolib::join(task1, task2);
        return x+y;
}

int main(int argc, chat **argv) {
        argolib::init(argc, argv);
        int result;
        argolib::kernel([&]() {
                result = fib(40);
        });
        std::cout <’’Fib(40) = “<<result<<std::endl;
        argolib::finalize();
        return 0;
}
```

Note that ArgoLib supports serial elision for C++ programs

ArgoLib Runtime Usage

For using ArgoLib the steps are as follows:

1) Download the Argobots from its **master** branch on github:

```
git clone https://github.com/pmodels/argobots.git
```

2) Build and install it in a separate directory. You should never have Argobots repository/installation inside your github project repository. We will use our own copy of Argobots for evaluating your project. Instructions for building and installing Argobots is provided in its README.md file on github.

3) Your ArgoLib will be a shared library (.so). While building your ArgoLib you would have to only provide the path to include folder inside Argobot installation directory. Use an environment variable to capture the path to the installation directory:

```
export ARGOBOTS_ INSTALL_DIR=/absolute/path/to/argobots/installation/directory
```

4) For compiling applications using your ArgoLib (e.g., Fib shown above), you will have to use the g++ as follows:

```
g++ -O3 -I$(ARGOLIB_INSTALL_DIR/include -I$(ARGOBOTS_INSTALL_DIR/include -L$(ARGOBOTS_INSTALL_DIR/lib
-L$(ARGOLIB_INSTALL_DIR)/lib -o fib fib.cpp -largolib -labt
```

5) For running applications using your ArgoLib (e.g., Fib shown above), you will have to first set the LD_LIBRARY_PATH as follows:

```
export LD_LIBRARY_PATH=/absolute/path/to/argobots/installation/directory/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/absolute/path/to/argolib/installation/directory/lib:$LD_LIBRARY_PATH
```

6) Launch fib as follows:

```
ARGOLIB_WORKERS=4 ./fib
```

Where, ARGOLIB_WORKERS specifies total number of execution streams to be used in Argobots.

ArgoLib Implementation

Following are the requirements for implementing ArgoLib:

1) As Argobots is a C-based implementation, your ArgoLib runtime will also be a C-based implementation. However, it would support both C/C++ user applications as mentioned above.

2) You must do proper documentation in your code.

3) ArgoLib should use a work-stealing runtime. You will have to look at examples inside the argobots directory as mentioned in the lecture slides. All the examples inside Argobots are inside the directories "examples" and "test".

4) You should not use the inbuilt work-stealing scheduler, but rather build your own using the basic building blocks provided by the Argobots. Marks will be deducted if you use the inbuild work-stealing scheduler.

5) Your scheduler should use push, pop, and steal as in happens in the default case in work-stealing.

6) Provide Makefile(s) for building argolib and sample test cases. You can implement fib.cpp on your own as shown above for testing your ArgoLib. We will provide more benchmarks later.

Part-2: Implementing a High Performance Extension for ArgoLib
Weightage: 12%

For this part, you have to optimize your implementation of ArgoLib by supporting a high performance feature as explained below.

1) You will have to implement **ONE** of the optimizations taught in the lecture slides. Research papers for all those optimizations are mentioned below.

a) **Scheduling I/O latency hiding futures in ArgoLib**
https://www.cse.wustl.edu/~angelee/home_page/papers/futureIO.pdf
Above paper use some dummy operations for demonstrating IO latency (e.g., sleep). You can also build similar type benchmark for demonstrating the effectiveness of your optimization. We won't provide benchmark for evaluating this feature.

b) **Lazy binary-splitting: a runtime adaptive work-stealing scheduler**
https://terpconnect.umd.edu/~barua/ppopp164.pdf
Although this paper explains the idea w.r.t. a parallel for loop, but as taught in lecture slides, this idea can be applied to any recursive algorithm.

c) Scheduling parallel programs by work-stealing with private deques
https://hal.inria.fr/hal-00863028/document
This paper has presented two types of algorithm for task initiation. You can build only one type as explained in the lecture slides.

d) An adaptive cut-off for task parallelism
https://ieeexplore.ieee.org/document/5213927
You should implementation the ATC algorithm as explained in the paper.

2) You must implement your solution inside your above implementation of ArgoLib, and by using the facilities provided by Argobots.

3) You should use compile time flags to enable/disable building of your default implementation of ArgoLib, and for building your modified ArgoLib implementation with the above optimization.

Bonus Option-1: Combining Two Optimizations Above
Weightage: 3%

You can get this bonus by implementing an extra optimization on top of your optimization in Part-2 above. You have to choose this extra optimization from the above four options only. If you implement this bonus, then provide compile time flag to enable/disable the compilation of bonus implementation.

Bonus Option-2: Recursive to Iterative Library
Weightage: 2%

To get this bonus you will have to implement some template/library that a programmer can use to automatically convert his recursive algorithm into an iterative algorithm. You can implement this solution outside your ArgoLib library. There are tutorials available on web that you can read on this topic. You will have to justify your solution using some well-known recursion (e.g., quicksort, mergesort, binary search, Fibonacci, etc.). You should also submit the limitations of your approach.

Choose only one bonus. You can submit bonus part towards the end of the semester.