Homework Assignment-1

Due by 11:59pm on 29th January 2023

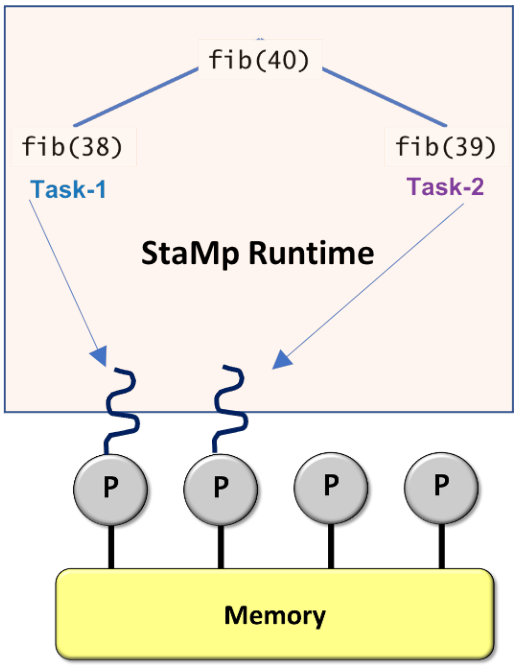
(Total weightage: 6%)

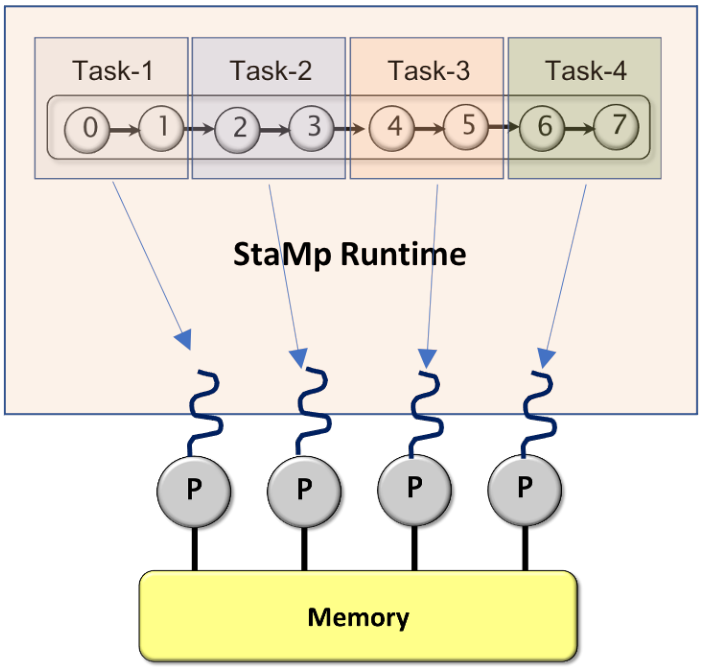
Instructor: Vivek Kumar

StaMp: Runtime for Static Mapping of Tasks to Threads

```
int fib(int n /*=40*/) {
  if (n < 2) {
    return n;
  } else {
    int x = fib(n-1);
    int y = fib(n-2);
    return (x + y);
  }
}
```

```
void foo() {
  for (int i=0; i<8; i++) {
    S(i); // can execute in parallel for all i
  }
}
```





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

Plagiarism: **This is an individual assignment.** All submitted programs are expected to be the result of your individual effort. You should never misrepresent someone else's work as your own. In case any plagiarism is detected, it will be dealt **strictly** as per the new plagiarism policy of IIITD.

Please also note that discussing the approach/solution of assignment with your batch mate, i.e. using your friend's idea is also plagiarism.

Instructions

1) Hardware requirements:
    a. You will require a Linux/Mac OS to complete this assignment.
    b. You can easily code/complete this assignment on your laptop.
2) Software requirements are:
    a. Pthread library.
    b. C++11 compiler.
    c. GNU make.
    d. You **must** do version controlling of all your code using github. You should only inside a **PRIVATE** repository. If you are found to be using a **PUBLIC** access repository then it will be considered as plagiarism.

Learning Objectives

1) Hands-on over C and C++11
2) Learning Pthread APIs (thread creation and termination)
3) Learning Makefiles
4) Learning dynamic linking

Problem Description

This assignment assumes that you are well familiar with Pthread based programming and Makefiles. From Lecture 02 (and from your past coding experience), you would recall how much programming effort someone has to spend if they want to parallelize an algorithm using Pthreads. For example, a parallel version of simple recursive Fibonacci could have 5x the lines of code than its corresponding sequential implementation.

In this assignment, you will have to help the Pthread programmers by abstracting away some of the essential programming efforts into a shared library that the programmer can link in their program during execution. This library aims to improve **productivity** in **Sta**tic **M**a**p**ping (a.k.a. "**StaMp**") of tasks to threads. We have provided three sample programs along with this assignment that uses StaMp APIs for harnessing parallelism.

Linguistic Interfaces to StaMp (Total 5)

1) Following are the linguistic interfaces for exposing the Pthread based parallelism in StaMp:

```
// accepts two C++11 lambda functions and runs the them in parallel
void execute_tuple(std::function<void()> &&lambda1, std::function<void()>  &&lambda2);

// parallel_for accepts a C++11 lambda function and runs the loop body (lambda) in
// parallel by using 'numThreads' number of Pthreads created inside StaMp
void parallel_for(int low, int high, int stride, std::function<void(int)> &&lambda, int numThreads);

// Shorthand for using parallel_for when lowbound is zero and stride is one.
void parallel_for(int high,  std::function<void(int)> &&lambda, int numThreads);

// This version of parallel_for is for parallelizing two-dimensional for-loops, i.e., an outter for-i loop and
// an inner for-j loop. Loop properties, i.e. low, high, and stride are mentioned below for both outter
// and inner for-loops. The suffixes "1" and "2" represents outter and inner loop properties respectively.
void parallel_for(int low1, int high1, int stride1,  int low2, int high2, int stride2,
                                                    std::function<void(int, int)>  &&lambda, int numThreads);
// Shorthand for using parallel_for if for both outter and inner for-loops, lowbound is zero and stride is one.
// In that case only highBounds for both these loop should be sufficient to provide. The suffixes "1" and "2"
// represents outter and inner loop properties respectively.
void parallel_for(int high1, int high2,  std::function<void(int, int)>   &&lambda, int numThreads);
```

StaMp Requirements

1) StaMp **must not use** any concept of task/thread pool. StaMp should simply create Pthreads whenever stamp::parallel_for APIs are invoked in the user program. You should implement StaMp in C++. **You are not allowed to use C++11 threading APIs (std::thread) or parallel programming APIs (e.g. std:async, etc.)**

2) StaMp runtime execution **must** have the exact number of threads specified by the programmer.

3) You don't need to do any error handling for the input arguments to StaMp's APIs. Assume only valid input arguments are provided.

4) Every call to StaMp linguistic interfaces (e.g., stamp::parallel_for) will create a new set of Pthreads and they will terminate as soon as the scope of that linguistic interfaces has ended.

5) Proper code documentation must be done in your StaMp implementation.

6) StaMp APIs and their implementation should be within the "stamp" namespace (see examples provided).

7) Your code should be modular and must avoid code repetitions.

8) Your assignment directory should be composed of following structure/files **ONLY** (don't change the structure/file names and its functionality):

   (A) Two sub-directories only inside your submission directory: a) "**library**", and b) "**examples**".

   (B) Sub-directory "**library**" should contain the following files:
   a. **stamp.h**: This header file should only contain the declarations of APIs supported by StaMp (see the Linguistic Interface section above).
   b. **stamp.cpp**: This should contain all of your stamp implementation.
   c. **Makefile:** This should be able to compile your implementation of stamp inside "library" directory and generate shared library (**libstamp.so**) in this same directory.

   (C) Sub-directory "**examples**" should contain the following files:
   a. "**vector.cpp**" (provided along with this assignment).
   b. "**matrix.cpp**" (provided along with this assignment).
   c. "**fib.cpp**" (provided along with this assignment).
   d. "**Makefile**". This file should compile all the above examples, link it with libstamp.so that you created above separately, and should create the executables "vector", "matrix", and "fib".
   e. "**README**". This should list bare minimum steps to use your StaMp implementation and run the above examples. You will loose marks if TA is unable to run the program by following the steps you mentioned in this README.

9) We will evaluate your implementation of StaMp using the three examples provided herewith without any changes. You should not do any changes to these examples as for evaluation the instructor will use his own copy of these examples. We will consider the correctness of your implementation only if all the examples execute successfully by following your README file.

10) StaMp must also print the total execution time for each call of a linguistic interface. Some sample execution of matrix.cpp on a dual core Intel processor by using StaMp is as mentioned below:
   a. $ ./matrix 1
      StaMp Statistics: Threads = 1, Parallel execution time = 8.69 seconds
      Test Success
   b. $ ./matrix 2
      StaMp Statistics: Threads = 2, Parallel execution time = 4.78 seconds
      Test Success