Assignment 3: Loop scheduler with pthreads

The purpose of this assignment is for you to learn more about

- creating threads and passing parameters to them,
- loop schedulers,
- overhead associated with thread management and synchronization.

As usual all time measurements are to perform on the cluster.
Link with the pthread library by passing `-pthread`.
Loop schedulers are used essentially whenever the code reads like your stereotypical *for* loop. In the previous assignment, transform would be a stereotypical for loop, but reduce could also be written this way. Here we will use the numerical integration example.

1 Preliminary

Question: Before anything, we will need to get timing for sequential execution. Navigate to the `sequential/` directory. Compile the sequential code with `make sequential` and get sequential time by running `make bench` on mamba.
Question: Navigate in the `hello_thread/` directory. Write a program in `hello_thead.cpp` that takes one parameter `nbthreads` the number of threads and that starts `nbthreads` threads which simply print "I am thread i in *nbthreads*". Compile with `make hello_thread`. Run with `./hello_thread 4`.

2 Writing a static loop scheduler

A static loop scheduler is the simplest way to achieve some parallelism. We will use it to make the numerical integration problem parallel.

Essentially, with T threads, each thread will do exactly $\frac{1}{T}$ of the loop iterations. So that if the loop has 100 iterations and there are four threads, the first threads will execute loop iterations from 0 to 24, the second will execute iterations from 25 to 49, ...

With the pthread library, one can pass parameters to a thread by passing a pointer to a struct that contains all the information it needs. The simplest way to write this code is to have one instance of the struct per thread, to give each thread the different information it needs. Some are the same for all the threads, a, b, a pointer to the results. Some are different, such as the range of indices they are supposed to execute.

Note that all the threads will need access to a pointer to the float that stores the sum of all the values. To avoid race conditions, this variables needs to be placed under mutual exclusion and there are two ways to do this:

- `iteration`-level mutual exclusion essentially puts the mutual exclusion on each access to the variable
- `thread`-level mutual exclusion makes each threads locally compute the sum of values in its own variable, and only aggregates the value into the results once all the iterations have been computed.

Question: Write a static loop scheduler and use it to compute numerical integration. Navigate to the `static/` directory and use `static_sched.cpp` as a template. Write the code so that it outputs the integral value on `stdout` and the time it takes to make the computation on `stderr`.

The program should take the following command line parameters:

- `functionid`, same as in numerical integration assignment
- `a`, same as in numerical integration assignment
- `b`, same as in numerical integration assignment
- `n`, same as in numerical integration assignment
- `intensity`, same as in numerical integration assignment
- `nbthreads`, an integer indicating the number of threads that should perform the numerical integration
- `sync`, a string, either `iteration` or `thread` that indicate which synchronization method to use.

Question: Report time and speedup across a range of precision, intensity, and synchronization mode. Use `make test` to test your code. Your code MUST pass the test before you can use `make bench` to start the PBS jobs. Once they are complete, plot the charts with `make plot`. Note: you must run these commands from within the `static/` directory.
Question: Why do you think some measurements are so erratic?
Question: Why is the speedup of low intensity runs with iteration-level synchronization the way it is?
Question: Compare the speedup of iteration-level synchronization to thread-level synchronization. Why is it that way?

3 Writing a dynamic loop scheduler

A dynamic loop schedulers are essentially managed by distributing ranges of indices to threads when they request them.

The worker threads execute a code that looks like

```
while (!loop.done()) {
  int begin, end;
  loop.getnext(&begin, &end);
  execute_inner_loop(begin, end);
}
```

The implementation of a loop scheduler boils done to implementing the two functions `done` and `getnext`. They can be easily implemented using mutual exclusion.

The size of the interval `[begin;end]` is called the `granularity` and is usually a parameter of the scheduler.

Pay attention that managing the memory of the loop scheduler can be a bit tricky as you need to be sure that all threads coordinated by the loop scheduler are done.

To compute numerical integration, note that the program needs to make sure that the result is correct, to do so, you can enforce the mutual exclusion in three places:

- `iteration`, Within the most inner loop for each step of the numerical integration,
- `chunk`, By locally storing the value in `execute_inner_loop` and adding them to a shared variable once per call.
- `thread`, By storing one value per thread, and aggregating to the shared variable once it is `done`.

Question: Write a dynamic loop scheduler to compute numerical integration. Navigate to the `dynamic/` directory and use `dynamic_sched.cpp` as a base code. Write the code so that it outputs the integral value on `stdout` and the time it takes to make the computation on `stderr`.

The code should take the same parameter as the previous one except:

- `sync` can also take `chunk` as a synchronization method
- `granularity`, an integer indicating how many iterations does one thread take at a time.

Question: Report time and speedup across a range of precision, intensity, and synchronization mode. Use `make test` to test your code. Your code MUST pass the test before you can use `make bench` to start the PBS jobs. When complete, you should be able to plot with `make plot`. Note: you must run these commands from within the `dynamic/` directory.
Question: Compare performance at 16 threads across the different synchronization modes. Why are the speedup this way?
Question: For thread level synchronization, compare the performance at 16 threads of different n and *intensity*. Why are the plots this way?

4 Extra credit

4.1 Gantt Chart

Question: Instrument the task scheduler to extract for each task its start time and completion time. Use that to print a Gantt chart of the execution.

4.2 Thread Pool

A common overhead is the time it takes the system to create threads. Often, runtime systems will create the threads only once and re-use them from one parallel construct to the next.
Question: Implement a thread pool in the loop scheduler and perform multiple numerical integration. What is the impact on speedup for small computation?

4.3 Merge Sort

Question: Implement a parallel implementation of merge sort using this scheduler.
Question: Report speedup for arrays of size ranging from 10^4 to 10^{10}.