Assignment 2
SPM course a.a. 24/25

March 29, 2025

Collatz problem parallelization using modern C++

The Collatz problem is a well-known mathematical problem defined as follows: for a given positive integer n, if n is even, the next number is $\frac{n}{2}$, if n is odd, the next number is $3n + 1$. This process is repeated until the sequence reaches 1. Although the rule is simple, the number of iterations required (i.e., the Collatz sequence length) can vary significantly among different starting numbers. The C++ function computing the Collatz sequence length (steps) for a given input n is as follows:

```
using ull=unsigned long long;
ull collatz_steps(ull n) {
    ull steps = 0;
    while(n != 1) {
        n = (n % 2 == 0) ? n/2 : 3*n + 1;
        ++steps;
    }
    return steps;
}
```

We aim to develop a parallel version of the Collatz problem that accepts multiple input ranges. The program should be invoked as follows: ./collatz_par range1 [range2] [range3] …
For example, given the following three input ranges (format 'start-end'), the output should be as follows:

./collatz_par 1-1000 50000000-100000000 1000000000-1100000000
1-1000: 178
50000000-100000000: 949
1000000000-1100000000: 984

For each range, the program must compute the maximum Collatz sequence length required by any number in that range to reach 1.

Requirements

1. Provide parallel Implementations using only plain C++17 (or above). Implement your own parallel scheduling for processing the input ranges. The parallel implementation should aim to minimize overheads.
2. Provide two parallel implementations:
   - Static Scheduling: Implement at least the block-cyclic variant of static scheduling.
   - Dynamic Scheduling: Implement at least one dynamic scheduling approach to allow threads to pull work dynamically as they complete their current tasks.
3. Provide a brief theoretical performance analysis using the Work-Span model to analyze the expected performance of parallel implementations estimating the theoretical speedup.
4. Provide a performance analysis of your implementations, comparing the strong speedup of your static and dynamic scheduling approaches on one internal node of the SPM cluster.

Input options, as for example, the scheduling version to use (static or dynamic), the number of threads, and the block/task size, should be provided as additional input parameters, as follows:

./collatz_par **-d -n 16 -c 32** 1-1000 1000-2000
Dynamic scheduling.
Running with 16 threads and a task size of 32 numbers.

./collatz_par **-n 8 -c 64** 1-1000 1000-2000
Static scheduling.
Running with 8 threads and a block size of 64 numbers.

If the parameter -d is not specified, the default scheduling is block-cyclic. If -n and -c are not specified, the default values are 16 and 1, respectively.

Deliverables

Provide all source files, including your sequential and parallel implementations, in plain C++. The scripts to compile and execute your code on the cluster nodes. A PDF report (max 4-5 pages) including a description of your implementations and the theoretical and concrete performance analysis comparing the two versions. Mention the challenges encountered and the solutions adopted.

Submit your source code and PDF report in a single zip file named 'collatz_parallel_<YourName>.zip' **by April 8 EOB.**