HPC - TBB - Parallel for and tasks

Table of Contents

1 Introduction

Threading Building Blocks (TBB) is a popular C++ library for efficient, multithreaded computation on both multicore and manycore (e.g., Intel Xeon Phi) processors. TBB is, in principle, similar to OpenMP. Compared to OpenMP, TBB focuses on more complex parallelism, such as data flow graphs or task-based parallelism (heavily inspired by Charles Leiserson's Cilk).

Files for TBB lab 1: tbb-lab-01.zip

2 Installing, intializing and compiling with TBB

2.1 Binary distribution

We will use a binary distribution of TBB (but it is relatively easy to build it from source).

- download a binary release from https://github.com/01org/tbb/releases
- unpack to some directory, e.g., `$HOME/tbb`
- initialize environmental variables:

```
cd $HOME/tbb/oneapi-tbb-2021.2.0/env
. ./vars.sh # note space between dot and vars.sh
# for 2020 TBB version: tbbvars.sh auto_tbbroot
# older TBB versions required: . tbbvars.sh intel64 linux auto_tbbroot
```

- verify that the variables are set: `env | grep TBB` should return a line with `TBBROOT=...`
- verify manual compilation through:

```
g++ parfor.cpp -std=c++14 -ltbb_debug -o parfor
```

- we will use `cmake` for compilation. The sources for this lab include `FindTBB.cmake`, an add on for `cmake` to find the TBB library.

2.2 Homebrew

Alternative on OSX with ARM processors (or when no binary distribution is available):

- install tbb through brew: `brew install tbb`
- find the installation path by `brew list tbb` (should be somewhere in `/opt/homebrew/Cellar/tbb/`)
- verify manual compilation through:

```
g++ parfor.cpp --std=c++14  -I/opt/homebrew/Cellar/tbb/2021.5.0_1/include/ -L/opt/homebrew/Cellar/tbb/2021.5.0_1/lib -ltbb -o parfor
```

- manually set TBBROOT:

```
export TBBROOT=/opt/homebrew/Cellar/tbb/2021.5.0_1/
```

3 Parallelization of simple for loops

While we will focus on more complex parallelization schemes, TBB also has a powerful `tbb::parallel_for` construct. Given a range of loop indices, `parallel_for` partitions it into chunks and then executes in parallel a function for each chunk. We will use a lambda-expression version of `parallel_for`.

`parfor.cpp` shows an example:

```
tbb::parallel_for(  // execute a parallel for
    tbb::blocked_range<long>(1, limit),  // pon a range from 1 to limit
    [](const tbb::blocked_range<long>& r) { // inside a loop, for a partial range r,
        // run this lambda
        for (long i = r.begin(); i != r.end(); ++i)
            is_prime(i);
    }
    );
```

Exercises:

- Figure out the algorithm used by TBB to partition the range into chunks.
- Add a counter that shows the total number of primes found. Hint: the counter has to be of type `std::atomic<long>` (check what happens when the counter is not atomic).

4 Task-based parallelism

Intuitively, a task is a unit of computation coupled with the data and the dependencies. Task-based parallelism is an approach fundamentally different from thread-based parallelism. In classic thread-based parallelism, a programmer specifies not only what has to be done, but also how — in a separate thread. In task-based parallelism, a programmer specifies tasks and their dependencies that are then automatically scheduled by a scheduler. Thus, tasks are excellent if you don't know the load of a particular unit of work (for instance, when you explore a graph with an unknown structure).

Task-based parallelism is a concept used e.g., in Charles Leiserson's Cilk programming language, or in CPP11 (but the CPP implementation is "not quite there yet", https://bartoszmilewski.com/2011/10/10/async-tasks-in-c11-not-quite-there-yet).

`tasks-fib.cpp` shows an example of computing Fibonacci numbers with tasks:

```
int par_fib(long n) {
    if (n < 2) {
        return n;
    }
    else {
        long x, y;
        tbb::task_group g;
        g.run([&]{x = par_fib(n - 1);});  // spawn a task
        g.run([&]{y = par_fib(n - 2);});  // spawn another task
        g.wait();  // wait for both tasks to complete
        return x + y;
    }
}
```

Exercise: Compute Tribonacci numbers: Tri(n) = Tri(n-3) + Tri(n-2) + Tri(n-1)

5 Exercise: The n-queens problem

Use task-based parallelism to implement a parallel version of the n-queens problem: how to place n queens on an n x n checkerboard so that no two queens attack each other. A sequential solution is in `nqueens.cpp`.

- Extend the solution to a parallel version. Avoid the race condition when modifying `partial_board`.

- Start with a version that only counts the number of possible solutions (instead of generating all of them).
- Make sure you will get a decent speed-up compared to the sequential version (unlike our Fibonacci implementation). Think about the "grain size", or when to stop spawning new tasks.
- To return all solutions in a thread-safe way, use `tbb::concurrent_queue<board>`. Measure the performance degradation compared with just counting the solutions.

6 Bibliography

- https://www.threadingbuildingblocks.org/
- https://software.intel.com/en-us/node/506058 parallel for
- https://www.threadingbuildingblocks.org/tutorial-intel-tbb-task-based-programming task based programming
- Introduction to Algorithms, 3rd edition, chapter 27: "Multithreaded Algorithms"

Date: 2022/05/27
Author: Krzysztof Rządca
Created: 2024-05-24 Fri 15:24
Emacs 25.3.50.1 (Org mode 8.2.10)
Validate