portfolio_7

2023-03-09

Intel TBB

Before jumping into what intel TBB is and how we can use it we will first talk about some other stuff. This other stuff will let us write functions that are generalizable and more easily parrallizable, namely by using mapping and reducing functions. Then we will discuss how we can then parrallize these functions using intel TBB.

Preliminaries

First let's get into the preliminaries. We are going to print out the contents of a c++ file which will defne some usefull functions (namely map and reduce) and explain how it all works:

```
cat main.cpp
```

```
## #include <iostream>
## #include <fstream>
## #include <string>
## #include <vector>
## #include "part1.h"
##
##
## // A function that reads a text file and returns a vector of strings
## // we will use this to read in the odyssey text file
## std::string read_file(const std::string &filename)
## {
##     std::ifstream file(filename);
##     std::string result;
##
##     if (file.is_open())
##     {
##         std::string line;
##         while (std::getline(file, line))
##         {
##             result += line;
##         }
##     }
##
##     return result;
## }
##
## // A function that takes a long string and turns it into a vector of strings
## // we will use this to split the odyssey text file into a vector of words
## std::vector<std::string> split_string(const std::string &str)
## {
##     std::vector<std::string> result;
##     std::string word;
##     for (auto &c : str)
##     {
##         if (c == ' ')
##         {
##             result.push_back(word);
##             word = "";
##         }
##         else
##         {
##             word += c;
##         }
##     }
##     result.push_back(word);
##     return result;
## }
##
## // A function that checks if the word given is "wine"
## bool contains_wine(const std::string &word)
## {
##     return word == "wine";
## }
##
## // A function that check if the first word is "wine" and the second is "dark"
## bool contains_wine_dark(const std::string &word1, const std::string &word2)
## {
##     bool bool1 = (word1 == "wine");
##     bool bool2 = (word2 == "dark");
##     bool out = (bool1 && bool2);
##     return out;
## }
##
## // A generic function that given two words will check if the next two words are the same
## // we will use this to show how to use lambda functions
## bool contains(const std::string &word1, const std::string &word2, const std::string &word3, const st
## {
##     return (word1 == word3) && (word2 == word4);
## }
##
## // We will use all these contains functions in conjunction
## // with our map function when we get to main()
##
## // TODO: explain below:
## // Here we are creating a function that can map a function onto every element in a vector
## // TODO: explain templating -> allows being generic with static typing
## // template<class FUNC, class T>
## // auto map(FUNC func, const std::vector<T> &arg1, const std::vector<T> &arg2)
## // {
## //     int nvalues = std::min( arg1.size(), arg2.size() );
##
## //     auto result = std::vector<T>(nvalues);
##
## //     for (int i=0; i<nvalues; ++i)
## //     {
```

```
## //         result[i] = func(arg1[i], arg2[i]);
## //     }
##
## //     return result;
## // }
##
## // TODO: explain below
## // Here we now define a generic version of map,
## // this version can map a function to an arbitrary number of vectors,
## // we build in this functionality by using the "..." variadic template operator
## template<class FUNC, class... ARGS>
## auto map(FUNC func, const std::vector<ARGS>&... args)
## {
##     typedef typename std::result_of<FUNC(ARGS...)>::type RETURN_TYPE;
##     // TODO: whats detail?
##     int nargs=part1::detail::get_min_container_size(args...);
##
##     std::vector<RETURN_TYPE> result(nargs);
##
##     for (size_t i=0; i<nargs; ++i)
##     {
##         result[i] = func(args[i]...);
##     }
##
##     return result;
## }
##
## // A function which takes an int and a bool and adds one to the int if the bool is true
## // we will use this with reduce in a moment
## int add_one_if_true(int x, bool y)
## {
##     if (y)
##     {
##         return x+1;
##     }
##     else
##     {
##         return x;
##     }
## }
##
## // TODO: explain below
## template<class FUNC, class T>
## T reduce(FUNC func, const std::vector<T> &values)
## {
##     if (values.empty())
##     {
##         return T();
##     }
##     else
##     {
##         T result = values[0];
##
##         for (size_t i=1; i<values.size(); ++i)
##         {
##             result = func(result, values[i]);
##         }
##
##         return result;
##     }
## }
##
##
## int main(int argc, char **argv)
## {
##     // First we are going to read in the odyssey text file
##     // and we are going to turn it into a vector of strings
##     auto words = read_file("odyssey.txt");
##     auto vec_words = split_string(words);
##     // We are now going to print out the first 10 words
##     std::cout << "The First 10 words in the Odyssey:" << std::endl;
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << vec_words[i] << std::endl;
##     }
##
##     // Let's now use our map function to find every occurence of the word "wine"
##     // this will return a vector of bools
##     auto has_wine = map(contains_wine, vec_words);
##     // We can now print out the first 10 bools
##     std::cout << "" << std::endl;
##     std::cout << "Which of the first ten words contain the word wine?" << std::endl;
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_wine[i] << std::endl;
##     }
##
##     // What if we now wanted to find out all the places where the word wine is followed by dark?
##     // we can do this by using our map function again
##     // First let's create a vector where the words are shifted by one
##     std::vector<std::string> vec_words_shift = vec_words;
##     vec_words_shift.insert(vec_words_shift.begin(), vec_words.back());
##     vec_words_shift.pop_back();
##
##     // Now we can use our map function
##     std::vector<bool> has_wine_dark = map(contains_wine_dark, vec_words, vec_words_shift);
##     // We can now print out the first 10 bools
##     std::cout << "" << std::endl;
##     std::cout << "Which of the first ten words contain the word wine followed by dark?" << std::endl
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_wine_dark[i] << std::endl;
##     }
##
##     // Now let's do the same but we will use a lambda function
##     // a lambda function allows us to define a function on the fly
##     // recall the generic contain function we defined above
##     // let's write a lambda function that does the same thing as contains_wine_dark
##     auto contains_wine_dark_lambda = [](const std::string &word1, const std::string &word2)
##     {
##         return contains(word1, word2, "wine", "dark");
##     };
##     // Here we name the lambda function contains_wine_dark_lambda
##     // we can also use it in-line so we don't have to name it
##     // let's do that now
##     has_wine_dark = map([](const std::string &word1, const std::string &word2)
##     {
##         return contains(word1, word2, "wine", "dark");
##     }, vec_words, vec_words_shift);
##     // Let's check this is the same as before:
##     std::cout << "" << std::endl;
##     std::cout << "Which of the first ten words contain the word wine followed by dark? this time we 
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_wine_dark[i] << std::endl;
##     }
##     // Note that there is something called capture in lambda functions
##     // this allows us to capture variables from the outside scope
##     // the three main ways to capture are by value (writing [=]),
##     // by reference (writing [&]), and to not capture at all! (writing [])
##
##     // Let's now find out the number of times the phrase "wine dark" appears in the oddyssey
##     // we can do this by using our reduce function
##     int n_wine_dark = reduce(add_one_if_true, has_wine_dark);
##     // Let's print out the result
##     std::cout << "" << std::endl;
##     std::cout << "The number of times the phrase wine dark appears in the oddyssey is: " << n_wine_da
##
##     return 0;
## }
##
## // TODO: change from wine dark to some other phrase that actually exists in the oddyssey
```

Let's now run the above to see if it works (notice we specify we want to use C++ version 14 and that we want to optimize the compilation (-O3)):

```
g++ -O3 --std=c++14 main.cpp -o main
./main
```

```
## The First 10 words in the Odyssey:
## PREFACE
## TO
## FIRST
## EDITIONThis
## translation
## is
## intended
## to
## supplement
## a
##
## Which of the first ten words contain the word wine?
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
##
## Which of the first ten words contain the word wine followed by dark?
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
##
## Which of the first ten words contain the word wine followed by dark? this time we used a lambda func
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
##
## The number of times the phrase wine dark appears in the oddyssey is: 0
```

Parrallelizing

Ok, now we've covered the preliminaries let's now talk about how we would go about paralelizing some of the above using intel TBB. Intel TBB is an efficient task-based scheduler that supports multi-level parallelism.