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 <algorithm>
## #include <regex>
## #include "part1.h"
##
##
## // A function that reads a text file and returns it as a string
## // 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))
##         {
##             // Replace punctuation with spaces
##             std::replace(line.begin(), line.end(), ',', ' ');
##             std::replace(line.begin(), line.end(), '.', ' ');
##             std::replace(line.begin(), line.end(), ';', ' ');
##             std::replace(line.begin(), line.end(), ':', ' ');
##             std::replace(line.begin(), line.end(), '!', ' ');
##             std::replace(line.begin(), line.end(), '?', ' ');
##             std::replace(line.begin(), line.end(), '"', ' ');
##             std::replace(line.begin(), line.end(), '\'', ' ');
##             std::replace(line.begin(), line.end(), '(', ' ');
```

```
##             std::replace(line.begin(), line.end(), ')', ' ');
##             std::replace(line.begin(), line.end(), '-', ' ');
##             line = std::regex_replace(line, std::regex("-"), " ");
##             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;
##     bool prev_space = false;
##     for (auto &c : str)
##     {
##         if (c == ' ')
##         {
##             // this makes sure we don't add empty strings to the vector
##             if (!prev_space)
##             {
##                 result.push_back(word);
##                 word = "";
##             }
##             prev_space = true;
##         }
##         else
##         {
##             word += c;
##             prev_space = false;
##         }
##     }
##     result.push_back(word);
##     return result;
## }
##
## // A function that checks if the word given is "the"
## int contains_the(const std::string &word)
## {
##     int out = (word == "the") || (word == "The") || (word == "THE");
##     return out;
## }
##
## // A function that check if the first word is "the" and the second is "gods"
## int contains_the_gods(const std::string &word1, const std::string &word2)
## {
##     int bool1 = (word1 == "the") || (word1 == "The") || (word1 == "THE");
##     int bool2 = (word2 == "gods") || (word2 == "GODS") || (word2 == "Gods");
##     int 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
## int contains(const std::string &word1, const std::string &word2, const std::string &word3, const std
## {
##     // Let's turn all the words to lower case
##     std::string word1_lower = word1;
##     std::string word2_lower = word2;
##     std::string word3_lower = word3;
##     std::string word4_lower = word4;
##     std::transform(word1_lower.begin(), word1_lower.end(), word1_lower.begin(), ::tolower);
##     std::transform(word2_lower.begin(), word2_lower.end(), word2_lower.begin(), ::tolower);
##     std::transform(word3_lower.begin(), word3_lower.end(), word3_lower.begin(), ::tolower);
##     std::transform(word4_lower.begin(), word4_lower.end(), word4_lower.begin(), ::tolower);
##     int out = ((word1_lower == word3_lower) && (word2_lower == word4_lower));
##     return out;
## }
##
## // We will use all these contains functions in conjunction
## // with our map function when we get to main()
##
## // We will now define a generic version of map,
## // this map function can map a function to an arbitrary number of vectors.
## // The first thing to explain is templating,
## // templating is a way to define a function that can work with any type,
## // we will use this to define a map function that can work with any type,
## // with the template here we are saying that the function we define
## // can take some arbitrary type FUNC and some arbitrary number of vectors of type ARGS,
## // we build in this functionality by using the "..." variadic template operator.
## // Map will return a vector of the type returned by the function we pass in,
## // where it will apply the function to the first element of each vector,
## // then the second element of each vector, etc.
## // note: the function we pass in must take the same number
## // of arguments as the number of vectors we pass in
## // We will put this function into action later on in main()
## template<class FUNC, class... ARGS>
## auto map(FUNC func, const std::vector<ARGS>&... args)
## {
##     // Here we define the type of the return value of the function we pass in
##     typedef typename std::result_of<FUNC(ARGS...)>::type RETURN_TYPE;
##     // Here we use a function defined in the file part1
##     // what it does is get the minimum size of the vectors we pass in
##     int nargs=part1::detail::get_min_container_size(args...);
##     // Here we define the vector we will return
##     // with the same type as the return type of func
##     std::vector<RETURN_TYPE> result(nargs);
##
##     // We now loop over the vectors and apply func to each element
##     for (size_t i=0; i<nargs; ++i)
##     {
##         result[i] = func(args[i]...);
##     }
##
##     return result;
```

```
## }
##
## // A function which takes two int's and returns their sum
## // we will use this with reduce in a moment
## int sum(int x, int y)
## {
##     return x+y;
## }
##
## // Here we define the reduce function,
## // reduce takes a function and a vector of values,
## // it then applies the function to the first two elements of the vector,
## // then applies the function to the result of the previous function
## // and the third element of the vector,
## // and so on until it has applied the function to all the elements of the vector.
## // To implement reduce again we use templating,
## // allowing us to define a function that can work with any type,
## // here we have to arbitrary types FUNC and T,
## // where the function takes in an argument func of type FUNC
## // and an argument values of type std::vector<T> (a vector of type T)
## template<class FUNC, class T>
## T reduce(FUNC func, const std::vector<T> &values)
## {
##     // Deal with the case where the vector is empty
##     if (values.empty())
##     {
##         return T();
##     }
##     // If the vector is not empty...
##     else
##     {
##         // We start with the first element of the vector
##         T result = values[0];
##         // We then loop over the rest of the vector
##         // and apply the function to the result and the next element
##         for (size_t i=1; i<values.size(); ++i)
##         {
##             result = func(result, values[i]);
##         }
##
##         return result;
##     }
## }
##
## // Let's now use the functions we explained above
## 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 "the"
##     // this will return a vector of bools
##     auto has_the = map(contains_the, 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 the?" << std::endl;
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_the[i] << std::endl;
##     }
##
##     // What if we now wanted to find out all the places where the word the is followed by gods?
##     // we can do this by using our map function again
##     // First let's create a vector where the words are shifted by one to the left
##     std::string temp = vec_words[0];
##     std::vector<std::string> vec_words_shift(vec_words.size());
##     for (int i=0; i<vec_words.size()-1; ++i)
##     {
##         vec_words_shift[i] = vec_words[i+1];
##     }
##     vec_words_shift[vec_words.size()-1] = temp;
##
##     // Now we can use our map function
##     std::vector<int> has_the_gods = map(contains_the_gods, 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 the followed by gods?" << std::endl;
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_the_gods[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_the_gods
##     auto contains_the_gods_lambda = [](const std::string &word1, const std::string &word2)
##     {
##         return contains(word1, word2, "the", "gods");
##     };
##     // Here we name the lambda function contains_the_gods_lambda
##     // we can also use it in-line so we don't have to name it
##     // let's do that now
##     has_the_gods = map([](const std::string &word1, const std::string &word2)
##     {
##         return contains(word1, word2, "the", "gods");
##     }, 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 the followed by gods? this time we us
```

```
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_the_gods[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 "the gods" appears in the oddyssey
##     // we can do this by using our reduce function
##     int n_the_gods = reduce(sum, has_the_gods);
##     // Let's print out the result
##     std::cout << "" << std::endl;
##     std::cout << "The number of times the phrase the gods appears in the oddyssey is: " << n_the_gods
##
##     return 0;
## }
```

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:
## THE
## GODS
## IN
## COUNCIL
## MINERVA'S
## VISIT
## TO
## ITHACA
## THE
## CHALLENGE
##
## Which of the first ten words contain the word the?
## 1
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 1
## 0
##
## Which of the first ten words contain the word the followed by gods?
## 1
## 0
## 0
## 0
## 0
```

```
## 0
## 0
## 0
## 0
## 0
##
## Which of the first ten words contain the word the followed by gods? this time we used a lambda functi
## 1
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 0
##
## The number of times the phrase the gods appears in the oddyssey is: 120
```

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. We will try and do the same thing we did before, but this time we will parallelize everything, let's print out the code which explains how to use intel tbb:

```
cat main_tbb.cpp
```

```
## #include <iostream>
## #include <fstream>
## #include <string>
## #include <vector>
## #include <cmath>
## #include <algorithm>
## #include <regex>
## #include "part1.h"
## #include <tbb/parallel_for.h>
##
##
##
## // Everything from here till the big break is the same as in main.cpp
## // ------------------------------------------------------------------------
## // ------------------------------------------------------------------------
##
## 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))
##         {
```

```
##             // Replace punctuation with spaces
##             std::replace(line.begin(), line.end(), ',', ' ');
##             std::replace(line.begin(), line.end(), '.', ' ');
##             std::replace(line.begin(), line.end(), ';', ' ');
##             std::replace(line.begin(), line.end(), ':', ' ');
##             std::replace(line.begin(), line.end(), '!', ' ');
##             std::replace(line.begin(), line.end(), '?', ' ');
##             std::replace(line.begin(), line.end(), '"', ' ');
##             std::replace(line.begin(), line.end(), '\'', ' ');
##             std::replace(line.begin(), line.end(), '(', ' ');
##             std::replace(line.begin(), line.end(), ')', ' ');
##             std::replace(line.begin(), line.end(), '-', ' ');
##             line = std::regex_replace(line, std::regex("-"), " ");
##             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;
##     bool prev_space = false;
##     for (auto &c : str)
##     {
##         if (c == ' ')
##         {
##             // this makes sure we don't add empty strings to the vector
##             if (!prev_space)
##             {
##                 result.push_back(word);
##                 word = "";
##             }
##             prev_space = true;
##         }
##         else
##         {
##             word += c;
##             prev_space = false;
##         }
##     }
##     result.push_back(word);
##     return result;
## }
##
## // A function that checks if the word given is "the"
## int contains_the(const std::string &word)
## {
##     int out = (word == "the") || (word == "The") || (word == "THE");
##     return out;
```

```
## }
##
## // A function that check if the first word is "the" and the second is "gods"
## int contains_the_gods(const std::string &word1, const std::string &word2)
## {
##     int bool1 = (word1 == "the") || (word1 == "The") || (word1 == "THE");
##     int bool2 = (word2 == "gods") || (word2 == "GODS") || (word2 == "Gods");
##     int 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
## int contains(const std::string &word1, const std::string &word2, const std::string &word3, const std
## {
##     // Let's turn all the words to lower case
##     std::string word1_lower = word1;
##     std::string word2_lower = word2;
##     std::string word3_lower = word3;
##     std::string word4_lower = word4;
##     std::transform(word1_lower.begin(), word1_lower.end(), word1_lower.begin(), ::tolower);
##     std::transform(word2_lower.begin(), word2_lower.end(), word2_lower.begin(), ::tolower);
##     std::transform(word3_lower.begin(), word3_lower.end(), word3_lower.begin(), ::tolower);
##     std::transform(word4_lower.begin(), word4_lower.end(), word4_lower.begin(), ::tolower);
##     int out = ((word1_lower == word3_lower) && (word2_lower == word4_lower));
##     return out;
## }
##
## // A function which takes two int's and returns their sum
## // we will use this with reduce in a moment
## int sum(int x, int y)
## {
##     return x+y;
## }
##
## // ----------------------------------------------------------------------------
## // ----------------------------------------------------------------------------
## // From here on things will be a little different
##
## // This is the same map as in main.cpp
## // but now it is parrallelized using tbb
## template<class FUNC, class... ARGS>
## auto map(FUNC func, const std::vector<ARGS>&... args)
## {
##     // Here we define the type of the return value of the function we pass in
##     typedef typename std::result_of<FUNC(ARGS...)>::type RETURN_TYPE;
##     // Here we use a function defined in the file part1
##     // what it does is get the minimum size of the vectors we pass in
##     int nargs=part1::detail::get_min_container_size(args...);
##     // Here we define the vector we will return
##     // with the same type as the return type of func
##     std::vector<RETURN_TYPE> result(nargs);
##
##     // We now loop over the vectors and apply func to each element
```

```
##     // But this time we do it in a parralelized fashion
##     // by using the parrallelized for loop for tbb,
##     // we define a range we want the for loop to work over in the first argument
##     // in the second argument we define a lambda function which defines the for loop
##     // ie. we have the following form when using the tbb parallel for loop:
##     // tbb::parallel_for( range, kernel );
##     tbb::parallel_for(
##         tbb::blocked_range<int>(0,nargs),
##         [&](tbb::blocked_range<int> r)
##         {
##             for (int i=r.begin(); i<r.end(); ++i)
##             {
##                 result[i] = func(args[i]...);
##             }
##         }
##     );
##
##     return result;
## }
##
## // TODO: change this to parralel verison?
## template<class FUNC, class T>
## T reduce(FUNC func, const std::vector<T> &values)
## {
##     // Deal with the case where the vector is empty
##     if (values.empty())
##     {
##         return T();
##     }
##     // If the vector is not empty...
##     else
##     {
##         // We start with the first element of the vector
##         T result = values[0];
##         // We then loop over the rest of the vector
##         // and apply the function to the result and the next element
##         for (size_t i=1; i<values.size(); ++i)
##         {
##             result = func(result, values[i]);
##         }
##
##         return result;
##     }
## }
##
## // Let's now use the functions we explained above
## 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 now parralelized map function to find every occurence of the word "the"
##     // this will return a vector of bools
##     auto has_the = map(contains_the, 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 the?" << std::endl;
##     for (int i=0; i<10; ++i)
##     {
##         std::cout << has_the[i] << std::endl;
##     }
##
##     // Now we've used our parallelized map function
##     // let's use the tbb reduce function to find the total number of occurences of "the gods"
##
##     // First let's shift our word vector by one
##     // so that we can compare the current word with the next word
##     // and use our map function on the shifted vector
##     std::string temp = vec_words[0];
##     std::vector<std::string> vec_words_shift(vec_words.size());
##     for (int i=0; i<vec_words.size()-1; ++i)
##     {
##         vec_words_shift[i] = vec_words[i+1];
##     }
##     vec_words_shift[vec_words.size()-1] = temp;
##
##     auto has_the_gods = map([](const std::string &word1, const std::string &word2)
##     {
##         return contains(word1, word2, "the", "gods");
##     }, vec_words, vec_words_shift);
##
##     // Now we can use our reduce function to find the total number of occurences of "the gods"
##     // the reduce function has the following form:
##     // tbb::parallel_reduce(range, identity_value, kernel, reduction_function);
##     // where:
##     // range is a range of values over which to iterate
##     // identity_value is the value to start with
##     // kernel is a lambda function which defines the for loop,
##     // its used to run over a subrange of iterations of the total range
##     // reduction_function is a our reduction function
##     auto result = tbb::parallel_reduce(
##         tbb::blocked_range<int>(0, vec_words.size()),
##         0,
##         [&](tbb::blocked_range<int> r, double running_total)
##         {
##             for (int i=r.begin(); i<r.end(); ++i)
##             {
##             running_total += has_the_gods[i];
##             }
##
```

```
##             return running_total;
##         },
##         sum
##     );
##
##     // Let's now print our result
##     std::cout << "Number of times \"the gods\" appears in the oddysey is: " << result << std::endl;
##
##
##     return 0;
## }
```

Let's now run the above code to see if it works:

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

```
## The First 10 words in the Odyssey:
## THE
## GODS
## IN
## COUNCIL
## MINERVA'S
## VISIT
## TO
## ITHACA
## THE
## CHALLENGE
##
## Which of the first ten words contain the word the?
## 1
## 0
## 0
## 0
## 0
## 0
## 0
## 0
## 1
## 0
## Number of times "the gods" appears in the oddysey is: 120
```

It works! and we can see "the gods" appears 120 times! which is in agreement with what we got previously.