The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology

and high-performance computing”

TBB-Based Parallel Programming

Lecture 2. Parallelizing Simple Loops

Nizhni Novgorod

2014

Lecture 2. Parallelizing Simple Loops

Objectives

The purpose of this lecture is to study the TBB libary tools that enable parallelizing simple loops.

Abstract

The lecture describes the tbb::parallel_for template function that enables implementation of parallelized simple loops as illustrated by the matrix-vector multiplication problem. It reviews possible function variants and its parameters.

Guidelines

As a rule, computations with a predetermined number of iterations involve the use of the for cycle. The TBB library enables parallelization of such computations. To do this, the library offers the tbb::parallel_for template function. This function has a number of prototypes.

An example of good parallelism is a problem where loop iterations may not require mutual synchronization. This can be illustrated by the matrix-vector multiplication problem.

tbb::parallel_for has two mandatory template parameters. The first parameter is the Range, i. e. a special type class determining the number of loop iterations. The second parameter is the Functor, i. e. a class implementing loop computations via the body::operator() method.

The TBB library contains a number of implemented ranges: the tbb::blocked_range single-dimension half-open interval, tbb::blocked_range2d two-dimension half-open interval and tbb::blocked_range3d three-dimension half-open interval. User-defined Range can also be implemented. The the tbb::blocked_range sets a range in the form of a half-open interval [begin, end), where the type of the begin and end elements is set using the template. Int types, pointers, STL direct access iterators etc can be used as the template parameters.

The main operation for any Range is its splitting. The splitting operation involves the use of a splitting constructor which has to split the range into two subsets. For tbb::blocked_range, the interval is split into two equal subsets (accurate to the decimal place). The 2D range tbb::blocked_range2d is a complete analog of the single-dimension one except that it sets a 2D half interval of the form $[x1, x2)\times[y1, y2)$.

Functor is a special type class that performs the required computations using the operator() method. The Functor may be considered to be resulting from transformation of the loop body into class. operator() is the main method of the Functor. The method is declared constant as it does not require changing the values in the Functor fields, if any.

The Functor for tbb::parallel_for must include the following methods:

- copy constructor required for correct operation of tbb::parallel_for that copies the Functor according to the parallelism implementation algorithm approved by the library developers;

- destructor;
- operator() method that performs computation.

The use of lambda expressions instead of functors helps shorten the program code substantially without declaring the Functor class.

The tbb::parallel_for operation algorithm works in such a way that computations are subject to dynamic scheduling, i. e. at the execution stage. The determining factor of scheduling is the way how the Range is implemented and what scheduling strategy is used. The scheduling strategy is set using the third parameter of the function tbb::parallel_for. The TBB library features three classes that determine scheduling strategies, simple_partitioner, auto_partitioner, affinity_partitioner. auto_partitioner is the default one. In most cases, the strategy of automatic computation amount selection will be the most efficient. The auto_partitioner strategy will select the computation grain size automatically thus reducing parallelism-related contingencies. This will guarantee that the grain size to be received by each thread is not less than [grainsize/2]. The affinity_partitioner strategy is very similar to auto_partitioner except for the fact that the computation grain size is selected to ensure optimum CPU cache utilization.

Recommendations for Students

The information is mainly sourced from the official TBB web page https://www.threadingbuildingblocks.org/. The site features numerous documents and examples. A free library version for non-commercial use is also downloadable.

Andrews (2000) is a recommended introduction into parallel programming.

Quinn (2004) is also recommended as a description of typical problems of parallel programming.

Practice

1. Implement a parallel program using parallel_for that outputs the numbers of performed iterations. Determine how the output varies depending on the grainsize values and partitioners types.

2. Implement your own Range which splits the set of iterations as 2:1 and use it for the previous problem.

3. Implement parallel matrix addition.

4. Implement parallel matrix multiplication.

Test questions

1. tbb::parallel_for:

   a. Has one type of call that has two input values, Range, Functor.

   b. Has several types of call and can be called without indicating parameters (all parameters are default ones).

c. (+) Has several types of call; computation Range and Functor has to be indicated for each of them

2. tbb::blocked_range:
   a. Must always be used when computation range is set.
   b. (+)Is an example of the Range implementation and can be used to set computation ranges.
3. tbb::blocked_range3d:
   a. (+) Is used to set the range as a rectangular prism.
   b. Is a 3D vector to be used for visual 3D application development
4. tbb::blocked_range(3, 7):
   a. Sets the computation range within [3, 7].
   b. (+) Sets the computation range within [3, 7).
   c. Sets the computation range within (3, 7].
   d. Sets the computation range within (3, 7).
5. tbb::blocked_range(3, 7, 2):
   a. Sets grainsize = 2 and the following iterations for computation: 3, 4, 5, 6, 7.
   b. (+) grainsize = 2 and the following iterations for computation: 3, 4, 5, 6.
   c. Set the next iterations for computations: 3, 5, 7.
   d. Set the next iterations for computations: 3, 5.
6. is_divisible method of the Range class:
   a. (+) Returns true if split is possible, otherwise false
   b. Returns false if split is possible, otherwise true
   c. Returns true if the number of iterations in the iterative space is even, otherwise false
7. Split constructor Range(R& r, split) of the Range class:
   a. Creates a copy of the range and divides the range set by the interval into two parts (only the resulting space range is changed).
   b. Creates the exact copy of the range.
   c. (+) Creates a copy of the range and divides the range set by the interval into two parts (both the parent space and the resulting space range are changed).
8. Computation scheduling in parallel_for:
   a. Is static and predetermined by the TBB library developers.
   b. Is dynamic; the computation grain size is determined by the grainsize parameter.
   c. (+) Is determined by the iterative space used.
9. In case of Functor implementation for parallel_for:
   a. One must not call operator() directly for the Functor object.
   b. The result of Functor operation must be saved in the functor fields.
   c. (+) operator() accepts the iterative space as an input parameter
10. operator() of the Functor sent to parallel_for:
   a. (+) Accepts the Range as an input parameter
   b. (+) Has to be constant.
   c. Accepts two inputs - iterative space and computation grain size.
11. The simple_partitioner strategy:
   a. Is used in parallel_for by default.

b. Selects the computation grainsize automatically thus reducing parallelism-related contingencies.

c. (+) Uses the Range to ensure parallel computations.

References

1. Intel® Threading Building Blocks Home Page: https://www.threadingbuildingblocks.org/

2. Intel® Threading Building Blocks Reference Manual: https://software.intel.com/en-us/node/506130

3. Intel® Threading Building Blocks User Guide: https://software.intel.com/en-us/node/506045

4. Andrews, G. R. (2000). Foundations of Multithreaded, Parallel, and Distributed Programming. – Reading, MA: Addison-Wesley.

5. Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.