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 6. Data Flow Graph

Nizhni Novgorod

2014

Lecture 6. Flow Graph

Objectives

The purpose of this lecture is to study parallel application development based on representing an algorithm as a flow graph.

Abstract

This lecture describes parallel application development based on representing an algorithm as a flow graph. It contains examples demonstrating the reviewed approach. Main types of graph nodes are reviewed in detail.

Guidelines

There are some applications which are best represented as messages communicated among graph nodes. These messages may contain data or be just completion signals. The tbb::flow::graph class and the related node representation classes may be used to illustrate data flow graphs of a parallel program (Fig. 1).

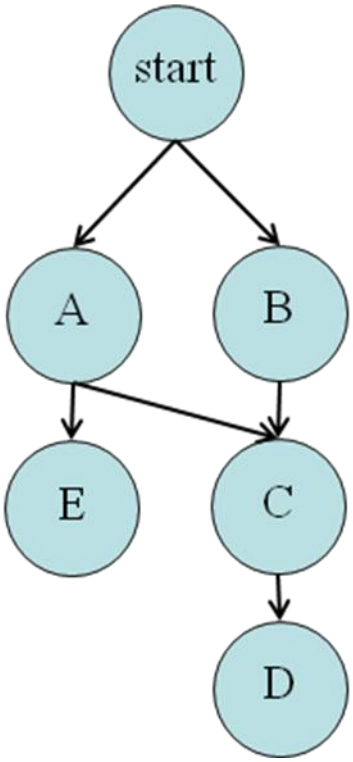

Fig. 1 Data flow graph example

tbb::flow::graph owns all threads which process the graph nodes. The graph nodes implement the user-defined handlers and manage the messages exchanged within the graph. The graph edges determine the node network.

Among the variety of graph node types there are nodes that require custom data processing functor (or lambda expression) and nodes that implement a certain functionality (e. g. sending the received message to all descendants).

Graph construction is based on creation of a tbb::flow::graph object. Then define the graph nodes for this object and connect them using the make_edge function.

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. Develop a program for parallel vector addition.
2. Develop a program for parallel scalar vector multiplication.
3. Implement a program that strops every number in a text file. Save the resulting text to a new file.
4. Implement a parallel algorithm of Fibonacci sequence computation based on a recursive algorithm.

Test questions

1. The tbb::flow::graph class
   a. Is intended for representation of weighted oriented graphs and running path algorithms.
   b. (+) Is intended for representation of the parallel algorithm data flow graph
2. Graph nodes:
   a. (+) Implement the user-defined handlers.
   b. Are used for software operation algorithm visualization.
   c. (+) Manage messages exchanged within the graph.
3. Graph edges:
   a. Are used to visualize software operation algorithm dependencies
   b. (+) Are used to determine the graph node network.
   c. Are used to set the cost of transition within the graph.
4. The graph node functionality can be set:
   a. By implementing the overloaded method class do().
   b. (+) By implementing the functor.
   c. (+) By implementing the lambda expression.
5. continue_node:
   a. (+) Waits for completion of backward nodes and does not accept input data.
   b. Receives a message to the single input port and generates a message at the single output port to be sent to all node descendants.
   c. Generates a message to be sent to all node descendants. Performed by strictly one thread.

d. Does not contain the user-defined code. Transfers the incoming message to all node descendants.

e. Does not contain the user-defined code. Based on numerous input messages, creates a common message to be sent to all descendants.

6. broadcast_node:

a. Waits for completion of backward nodes and does not accept input data.

b. Receives a message to the single input port and generates a message at the single output port to be sent to all node descendants.

c. Generates a message to be sent to all node descendants. Performed by strictly one thread.

d. (+) does not contain the user-defined code. Transfers the incoming message to all node descendants.

e. Does not contain the user-defined code. Based on numerous input messages, creates a common message to be sent to all descendants.

7. source_node:

a. Waits for completion of backward nodes and does not accept input data.

b. Receives a message to the single input port and generates a message at the single output port to be sent to all node descendants.

c. (+) Generates a message to be sent to all node descendants. Performed by strictly one thread.

d. Does not contain the user-defined code. Transfers the incoming message to all node descendants.

e. Does not contain the user-defined code. Based on numerous input messages, creates a common message to be sent to all descendants.

8. function_node:

a. Waits for completion of backward nodes and does not accept input data.

b. (+) Receives a message to the single input port and generates a message at the single output port to be sent to all node descendants.

c. Generates a message to be sent to all node descendants. Performed by strictly one thread.

d. Does not contain the user-defined code. Transfers the incoming message to all node descendants.

e. Does not contain the user-defined code. Based on numerous input messages, creates a common message to be sent to all descendants.

9. join_node:

a. Waits for completion of backward nodes and does not accept input data.

b. Receives a message to the single input port and generates a message at the single output port to be sent to all node descendants.

c. Generates a message to be sent to all node descendants. Performed by strictly one thread.

d. Does not contain the user-defined code. Transfers the incoming message to all node descendants.

e. (+) does not contain the user-defined code. Based on numerous input messages, creates a common message to be sent to all descendants.

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.