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”

INTRODUCTION TO PARALLEL PROGRAMMING

Lecture 3. OpenMP-Based Parallel Programming.

Nizhni Novgorod

2014

Lecture _3_OpenMP-Based Parallel Programming

A common approach to performing computations on shared memory multiprocessor systems is creation of new parallel methods based on existing sequential programs where sections of independent computations are identified either by means of a compiler or by programmers themselves The automatic software analysis capabilities for the purpose of parallel computations are limited to a certain extent, so the second approach prevails. In this context, both new programming languages oriented to parallel programming and the existing ones with operator sets added to enable parallel computations, can be used for parallel software development.

One more popular approach consists in the use of various libraries to obtain an API intended for parallel programming. The best known examples of this approach are Windows Thread API [9] and PThead API [7]. However, the first one is applicable only to Microsoft Windows, while the second one is complex enough and has a lower-level character.

All the above approaches require considerable rework of the existing software thus making popularization of parallel computations much more difficult. This brought about rapid development of one more approach to parallel programming where programmer's instructions on parallel computations are added to the program by means of certain external features of programming language which may have the form of directives or comments to be handled by a preprocessor before the program is complied. At the same time, the source text remains the same and is used by the compiler to construct the sequential source code in absence of the preprocessor. The preprocessor, if used, replaces the parallelism directives with a certain additional program code (as a rule, such code has the form of accesses to the procedures of a parallel library). The above approach is the basis of *OpenMP*.

OpenMP was developed on the basis of past experience in parallel programming for shared memory systems. Based on X3Y5 [8] and taking into account the possibilities of PThreads API [7], OpenMP uses a considerably simpler directive format and has new functionalities. The OpenMP Architectural Review Board or ARB was established at the earliest project stages to engage key experts in OpenMP development and standardization. The first OpenMP specifications for Fortran were published in 1997 while the C standard was released in 1998.The last OpenMP specifications for C and Fortran were published in 2005 (for more details, visit www.openmp.org).

The opening part of this lecture indicates that OpenMP is now a primary approach to parallel programming for shared memory systems.

The second part of this lecture deals with a number of key OpenMP notions and definitions. It determines a parallel program as a set of sequential (*single-thread*) and parallel (*multi-thread*) program code sections.

The third part gives a concise introduction into OpenMP-based parallel programming. It describes the **parallel** directive and gives the example of the first OpenMP-based parallel program. This part defines the notions of parallel program *segments*, *regions* and *sections* important for further discussion.

The fourth part is devoted to the issues of computational load allocation to threads as illustrated by data *loop parallelization*. It describes the **for** directive and the ways to manage allocation of loop iterations to threads.

The fifth part features a detailed discussion of data management for parallel threads. It determines general and local variables for the threads. It also describes *reduction*, a very important and frequently used operation of common data processing.

The final part of this lecture contains additional information on OpenMP, i. .e. it describes the possibility of program compilation as an ordinary program code and gives a list of compilers supporting OpenMP.

Test questions

1. What computer platforms range with shared memory systems?
2. What approaches to parallel software development are used?
3. What is the essence of OpenMP basics?
4. Why is it important to standardize parallel program development tools?
5. What are the main advantages of OpenMP?
6. What is a parallel program in terms of OpenMP?
7. What is meant by the notion of thread?
8. What writing format is used for OpenMP directives?
9. What is the purpose of the **parallel** directive?
10. What is meant by the parallel program segment, region and section?
11. What minimum set of OpenMP directives is necessary for parallel software development?
12. How can one determine runtime for an OpenMP-based software?
13. How does one parallelize loops in OpenMP? What are the conditions of loop parallelization?
14. Which OpenMP capabilities manage allocation of loop iteration to threads?
15. How is the order of iterations determined for parallelized loops in OpenMP?
16. How can one parallelize program code segments with low computational complexity?
17. What is meant by reduction?
18. What tools does OpenMP have to manage the number of created threads?
19. What is meant by dynamic thread creation mode?
20. How is parallel segment nesting managed?
21. How can one ensure the program code uniqueness for both sequential and parallel program versions?
22. What compilers ensure OpenMP support?

Practice

1. Develop a program to find the minimum (maximum) vector element value.
2. Develop a program to find the scalar product of two vectors.
3. Develop a program to solve the problem of definite integral computation using the method of rectangles

$$y = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f_i,\ f_i = f(x_i),\ x_i = i\,h,\ h = (b-a)/N$$

(see for example [15] for a description of integration methods).

4. Develop a program to solve the problem of finding the maximum value among the matrix row minimum elements (this problem is part of matrix games)

$$y = \max_{1\le i\le N} \min_{1\le j\le N} a_{ij}\ .$$

5. Develop a program for problem 4 based on the use of special type matrices (band, triangular etc). Determine runtime and evaluate the speedup. Perform computational experiments for various rules of iteration allocation to threads and compare efficiency of parallel computations (such experiments are appropriate for the problems where loop iteration complexity may vary).

6. Implement the reduction operation subject to various ways of ensuring mutual exclusion (atomic operations, critical sections, lock-based synchronization). Evaluate efficiency of different approaches. Compare the results with reduction performance resulting from the use of the *reduction* parameter of the **for** directive.

7. Develop a program for scalar product computation for a sequential set of vectors (input data may be prepared in advance in a separate file). Introduction of vectors and computation of their product should be processed as two different problems whose parallelization requires the use of **sections.**

8. Perform computational experiments using the previously developed programs involving various number of threads (less/more than or equal to that of the existing computing elements). Determine runtimes and evaluate the speedup.

9. Make sure that your compiler supports the nested parallel fragments. If yes, develop programs that use/do not use nested parallelism. Perform computational experiments and evaluate efficiency of different approaches.

10. Develop a program for problem 4 based on parallelization of loops of various nesting depth. Perform computational experiments and compare their results. Estimate contingencies related to thread creation and termination.

References

1. **Amdahl, G.** (1967). Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings, Vol. 30, pp. 483-485, Washington, D.C.: Thompson Books.
2. **Bertsekas, D.P., Tsitsiklis, J.N.** (1989). Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.

3. **Grama, A.Y., Gupta, A. and Kumar, V.** (1993). Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel and Distributed technology. 1 (3). pp. 12-21.
4. **Gustavson, J.L.** (1988) Reevaluating Amdahl's law. Communications of the ACM. 31 (5). pp.532-533.
5. **Kumar V., Grama, A., Gupta, A., Karypis, G.** (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)
6. **Quinn, M. J.** (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
7. **Butenhof D.R.** (1007) Programming with POSIX Threads. Boston, MA: Addison-Wesley Professional., 1997.
8. **Chandra R., Dagum L., Kohr D., Maydan D., McDonald J., Melon R.** Parallel Programming in OpenMP. San-Francisco, CA: Morgan Kaufmann Publishers., 2000.
9. Addison-Wesley Microsoft Technology Series Addison-Wesley Professional; 4 edition (February 26, 2010)