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 15. OpenMP-Based Parallel Programming (Continued)

Nizhni Novgorod

2014

Lecture_15_. OpenMP-Based Parallel Programming (Continued).

This lecture is a follow-up of lecture 3 which studies basic OpenMP directives. The purpose of this lecture is to review basic ways to ensure thread communication in parallel programs developed using OpenMP and to study OpenMP functions and environment variables enabling setup of OpenMp-based program runtime environment.

As it was mentioned earlier, threads are execute within the common address space of a parallel program. As a result, parallel thread communication can be ensured by means of shared data accessible by all threads. The simplest situation is when the shared data are read-only. When the shared data can be modified by numerous threads, some effort is required to guarantee proper communication. Indeed, let two threads execute the same program code

```
n=n+1;
```

for the shared variable *n*. Then, depending on execution conditions this operation may be executed sequentially (thus yielding a correct result); alternatively, both threads may simultaneously read the *n* value, increase it and write a new value in it (thus obtaining a wrong result). Such conditions when the result of computations depends on the thread execution rate are called *race conditions*. To avoid race conditions, one has to make sure that shared variables are modified by only one thread at a time, or, in other words, that in case of processing shared data threads are subject to *mutual exclusion*. In OpenMP, mutual exclusion can be ensured by *atomic operations*, *critical sections* or special semaphores (*locks)*.

It should be noted that mutual exclusion cuts down the possibility of parallel thread execution: in case of simultaneous access to shared variables only one thread will be able to continue operation while the others will be locked and wait for the shared data to be released. One can say that implementation of thread communication requires the skills of programming on shared memory systems: mutual exclusion is mandatory for operating shared data, but the resulting thread delays (locks) must be minimized.

Besides mutual exclusion, parallel program execution requires a cetrain degree of synchronization of computations executed by different threads: for example, data processing in one thread can start only after such data has been formed in another thread (this is the classical producer–consumer problem). In OpenMP, synchronization may be ensured by locks or the **barrier** directive.

This lecture is dedicated to methods for synchronizing access to shared data on shared memory systems based on OpenMP. It describes a number of new directives (**master**, **single**, **barrier**, **flush**, **threadprivate**, **copyin**).

See [13] for the most complete information on OpenMP parallel programming on shared memory systems. A concise description of OpenMP can be found in [2, 14]; [1, 7, 8] also contain some useful information.

There is yet more to read on OpenMP in the Internet. We would advise to visit www.parallel.ru and www.openmp.org.

For more information on multithread programming see 11] (for OC Windows) and [12] (for POSIX Threads).

To study general issues of parallel programming on shared memory systems, one can refer to [10].

Test questions

1. What are the rules of computation synchronization for parallelized loops in OpenMP?
2. What ways to ensure mutual exclusions can be used in OpenMP?
3. What is meant by an atomic operation?
4. How are critical sections determined?
5. What operations are offered by OpenMP for semaphore variables (locks)?
6. In what cases should we apply barrier synchronization?
7. How is task parallelism ensured in OpenMP (**sections** directive)?
8. How are single-thread parts of parallel fragments identified (**single** and **master** directives)?
9. How is memory state synchronized (using **flush** directive)?
10. How are persistent local variables of threads used (**threadprivate** and **copyin** directives)?

Practice

1. 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)

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

2. Develop a program for the same problem 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).

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)