World Applied Sciences Journal 28 (6): 842-846, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.28.06.13810

# Effective Teaching of "Parallel Computing" Course by Using Microlearning Technique

*Nurbek Saparkhojayev*

Senior Lecturer at Suleyman Demirel University, Ablayhan Street No 1/1,
Kaskelen, Almaty, Kazakhstan, 040900

**Abstract:** In current time, there are many types of problems in the High Education System. Among of these problems, the most important one is an exceeding number of information that teachers and professors provide students. And, after this, it takes unbelievable amount of time to read this information, understand it and get prepared to exams. Beside these constraints, the deadline for submitting any assignment or homework is also important. So, most students cannot finish all assigned home works or programming tasks at a time. This paper describes a research that was done in "Parallel Computing" course by use of microlearning approach. The research paper consists of introduction, purpose of work and conclusion. All research was done in Suleyman Demirel University, at Faculty of Engineering and Natural Sciences.

**Key words:** Microlearning · Parallel Computing · MPI · OpenMP · Education

## INTRODUCTION

In learning and developing environments the following parts called macro, meso and micro are used. The main one is macro, which contains meso parts and meso itself contains micro parts which present the smallest part among of them and finally all of them construct together any products. As an example, it is possible to use computer science example: in order to start and finish a big project, this project (macro) is divided into subprojects (meso) and after this, each subproject contains its own modules (micro). This technology is well-known as "divide and conquer" technique. From this example, it is possible to notice that for learning it is easier to start from small parts and then at the end construct big part. Microlearning is a new methodology of teaching which becomes more popular day by day.

According to the authors of these research papers [1-2], there are 7 dimensions of microlearning:

**Time:** Each topic can be taught in 5-7 minutes maximum. If topic is complex, then it should be divided into small parts. For example, MPI topic is separated into 4 small parts, represented in 4 microlessons below.

**Content:** Each topic is represented in A4 page, where topic description, simple test or simple tasks are given.

**Curriculum:** As shown in microlessons below, topics are divided into small subtopics.

**Form:** Mostly each parallel computing technology is taken simpler as it can be, for example to explain OpenMP technology, there were used some examples from real life.

**Process:** Learning "Parallel Computing" discipline is the same as learning to speak foreign language; it needs a lot of practice. So, for this reason, each topic contains simple test questions and tasks, which can be solved by students in around of 2-10 minutes.

**Mediality:** Microlearning in Parallel Computing is prepared as electronic book in PDF format with hyperlinks. It can be also used in cell phones because of PDF format.

**Learning type:** In the class there is a projector used for the "Parallel Computing" discipline, on which lecturer shows A4-formatted pages and then after explanation of

**Corrsponding Author:** Nurbek Saparkhojayev Senior Lecturer at Suleyman Demirel University, Ablayhan Street No 1/1, Kaskelen, Almaty, Kazakhstan, 040900.

each material, there are tests for students and simple tasks to do. It is used to make sure that students have understood all the materials correctly and appropriate.

To pass through all subjects it will be better to use literature in microlearning style which provides understanding of "Parallel Computing" terminology in basic way by dividing information into small micro parts which have a log of graphics and pictures and not too many words for explanation. These examples show that in learning it is important to begin from small parts to construct big parts.

**Microlearning in Parallel Computing:** In this section, the demonstration of 4 microlessons is discussed.

**Lesson 1:** The System performance degradation factors.

- Consists of 4 factors: starvation, latency, overhead and waiting for contention. Also known as SLOW;
- Starvation occurs when there is no work to do for processor;
- Latency is measure of the time delay experienced by a system. Occurs when a lot of processors want to send and receive at the same time;
- Overhead is non-useful work to manage the parallel resources and concurrent abstract tasks.
- Waiting on contention is about network-based time delay;

All of them make our system to perform inadequately and at the end, instead of speedup, the execution time is much bigger than in Sequential Computing.

**Test for 1st Lesson:**

What is SLOW?

- Starvation Latency Overhead Waiting on Contention;
- Super Level operating windows;
- Serial Leveled optimistic Work;
- Small low operating work;

What is the reason for starvation to occur?

- Too many work;
- Not correctly assignment of work;
- System does not want to calculate;
- Just for fun;

Latency occurs when..

- A lot of processors want to send at the same time;
- System is full;
- System is empty;
- Starvation occurred;

Overhead is a type of work which is

- Useful;
- Non-useful;
- Good for system;
- Not efficient;

**Lesson 2:** Processes vs. Threads

- Differences in their definitions
- Using technique

**Differences in their Definitions:** Parallel programming with directives (OpenMP) offers many advantages over programming with the message passing paradigm (MPI):

- Simple to program, with incremental path to full parallelization;
- Shared memory model, no need for explicit data distribution;
- Scalability achieved by taking advantage of hardware cache coherence; and
- Portability via standardization activities [4].

Process can create threads, not vice versa;
To create process, the space must be allocated;

To create threads, there is no need to allocate space for them;

Each thread will share a space with others;

**Using Technique:** For processes, in "Parallel Computing", MPI approach is used;

For threads, you may use pthreads or OpenMP approaches;

You should have at least few processors or cores in your machine for the correct and appropriate use when you do parallelization of your code by using processes;

For threads, there are no such requirements;

Execution times of programs with processes are less when you have enough system resources and big problems, whereas in problems with small dataset, the use of threads is more convenient;

**Task for 2nd Lesson:**

Which one is true regarding the creation of processes?

- When process is created, there should be space allocated for this process;
- No need to allocate, it will share with others;
- Process doesn't need any memory space;
- All answers are correct;

OpenMP Approach is used for:

- Process creation and manipulation;
- Threads creation and manipulation;
- For both;
- None of the answers is correct;

**Lesson 3- MPI:** How to use this approach

- Basic calls
- MPI communicators and inter-process communication
- Collective calls

To start with the explanation of MPI, we need to give a brief info about MPI. The idea of using MPI came up in 1990's, when a community representing both vendors and users decided to create a standard interface to message passing calls in the context of distributed memory parallel computers. And after some time, MPI-1 was introduced. And this was just API, not programming language. Moreover, it had just two bindings: for C and Fortran programming languages. And, nowadays with a high influence and use of high-performance computing in every aspect and field of industry, there are new versions of MPI are presented.

**Basic calls:** Every MPI program must contain the preprocessor directive.

#include "mpi.h". This mpi.h header file contains the definitions and declarations necessary for compiling any MPI program and mpi.h is usually found in the "include" directory of most MPI installations.

int MPI_Init(int *argc, char ***argv) function is necessary for initializing the MPI execution environment. MPI_init() must be called before any other MPI functions can be called and it should be called only once.

int MPI_Finalize() function is needed for terminating MPI execution environment. All MPI processes must call this routine before exiting.

After students know the basic two functions of MPI, they need to know how compile a program. There are 2 possible ways; however, for the save of time, usually this easiest way is used.

For compiling any program, this command is used:

mpicc -o your_executable mpicode.c

After compilation, the user need to run and for running any program this command is used:

mpiexec -n 2./a.out'

In this specific example, command runs two copies of /a.out where the system specifies number of processes to be 2.

**Mpi Communicators and Inter-process Communication:** Communicator is an internal object, which is used in MPI. MPI Programs are made up of communicating processes and each process has its own address space containing its own attributes such as rank, size (and argc, argv, etc.). MPI provides functions to interact with it. Default communicator is MPI_COMM_WORLD and here is how it is called within any program:

int MPI_Comm_size ( MPI_Comm comm, int *size) and all processes are its members and it has a size (the number of processes). Each process has a rank within it. You can think of it as an ordered list of processes.

For defining the rank of any process in a communicator, the following function is applied:

int MPI_Comm_rank ( MPI_Comm comm, int *rank), which returns the rank of the calling process in the group underlying the comm.

Message passing in MPI program is carried out by 2 main MPI functions.

- MPI_Send – sends message to a designated process;
- MPI_Recv – receives a message from a process;

A basic communication mechanism of MPI between a pair of processes in which one process is sending data and the other process receiving the data is called "*point to point communication*".

**Collective Calls:** A communication pattern that encompasses all processes within a communicator is known as collective communication. There are few of them, the most frequently used are:

- Synchronization: Barrier;
- Communication: a. Broadcast; b. Gather & Scatter; c. AllGather;
- Reduction: a. Reduce; b. AllReduce;

Barrier- creates barrier synchronization in a communicator group *comm*.

MPI_Bcast() - A collective communication call where a single process sends the same data contained in the *message* to every process in the communicator.

MPI_Scatter() - splits the data referenced by the *sendbuf* on the process with rank root into *p* segments each of which consists of *send_count* elements of type *send_type*.

MPI_Gather() - collects the data referenced by *sendbuf* from each process in the communicator *comm* and stores the data in process rank order on the process with rank *root* in the location referenced by *recvbuf*.

MPI_Allgather() - gathers the content from the send buffer (*sendbuf)* on each process.

MPI_Reduce() - A collective communication call where all the processes in a communicator contribute data that is combined using binary operations (MPI_Op) such as addition, max, min, logical and, etc.

MPI_Allreduce is used exactly like MPI_Reduce, except that the result of the reduction is returned on all processes, as a result there is no *root* parameter. For more information, please see the following reference [3].

**Test for 3rd Lesson:**

What are the functions for initializing and finalizing MPI environment?.

- MPI_Init() & MPI_Finalize();
- MPI_Start() & MPI_End();
- Start() & End();
- MPI_Open() & MPI_Close();

What are the functions for message-passing in MPI?.

- MPI_Send() - MPI_Receive();
- Send-Receive;
- Mail-Get;
- MPI_Deliver – MPI_ Return;

**Lesson 4:** Threads: POSIX threads(pthreads) vs. OpenMP threads.

The idea of using threads is trivial: developers try to achieve parallelism with insufficient resources, which means you use multi-thread programming to achieve high throughput, however you have just one shared memory for all threads. There are many types of them, like Java threads and others, however, in this paper, pthreads and OpenMP are discussed.

**Pthreads:** As it was already mentioned, threads are well-known as lightweight processes. They work very similar to processes, except the fact that they have shared resources. The header file #include <pthread.h> should be included in the code; however, comparing to MPI, there is no need to install package and prepare everything before running a program. Pthreads does not need any package to be installed in machine.

Pthread_create() -Creates a new thread within a process;

Pthread_exit() - Terminates the calling thread and makes the *value_ptr* available to any successful join with the terminating thread.

With a knowledge of these two commands, any developer who has some programming skills can start working on Pthreads. However, other commands need more details and because of this fact, are not covered.

**OpenMP:** What needs to be covered in this topic, are followings:

OpenMP [5] is designed to support portable implementation of parallel programs for shared memory multiprocessor architectures. OpenMP is an API (Application Programming Interface), it is not a programming language. It consists of a set of compiler directives that help the application developer to parallelize their workload. OpenMP is composed of the following main components:

- Directives;
- Runtime library routines;
- Environment variables;

Runtime libraries can be accessed by including omp.h in applications that use OpenMP.

Omp_get_num_threads() - returns the total number of threads currently in the group executing the parallel block from where it is called.

Omp_get_thread_num() - for the master thread, this function returns zero. For the child nodes the call returns an integer between 1and omp_get_num_ threads()-1inclusive. OpenMP provides four main environment variables for controlling execution of parallel codes:

- OMP_NUM_THREADS–controls the parallelism of the OpenMP application;
- OMP_DYNAMIC–enables dynamic adjustment of number of threads for execution of parallel regions;
- OMP_SCHEDULE–controls the load distribution in loops such as *do, for;*
- OMP_NESTED–Enables nested parallelism in OpenMP applications;

**Task for 4th Lesson:**

Which one of the followings is not main component of OpenMP?.

- Directives;
- Runtime library routines;
- Environment variables;
- Compilers;

Which one of the followings is not main environment variable of OpenMP?.

- OMP_NUM_THREADS;
- OMP_DYNAMIC;
- OMP_SCHEDULE;
- OMP_NESTED;
- OMP_STATIC;

## CONCLUSION

Microlearning gives a great opportunity for humanbeing in order to be more educated. At the current time, everyone unconsciously uses the microlearning approach while reading tutorials, forums, wiki, blogs and other resources. This approach was started to be used for teaching students in our university by using microlearning method and we have good feedback from them. There was another research done in [6], where authors showed that by the using microlearning approach, it is possible to teach "Computer Networks" course for students. It is an interesting fact, but now students provide more interest for course discussed above and students started learning more materials comparing to previous years. This paper shows how to apply microlearning principles in teaching "Parallel Computing" course.

## REFERENCES

1. Hug, Theo, Lindner, Martin; Bruck and A. Peter, 2006. Microlearning: Emerging Concepts, Practices and Technologies after e-Learning. Proceedings of Microlearning 2005. Innsbruck: Innsbruck University.
2. Micro Learning and Narration, Theo Hug, Innsbruck, Paper presented at the fourth Media in Transition conference, May 6-8, 2005, MIT, Cambridge (MA), USA.
3. MPI Standard official web-site for developers: http://www.mpi-forum.org/.
4. Jin H., M. Frumkin and J. Yan, 1999. "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance", NAS Technical Report NAS-99-011.
5. OpenMP Fortran Application Program Interface, http://www.openmp.org/.
6. Zhamanov, A. and M. Zhamapor, 2013. "Computer Networks teaching by microlearning principles", in Journal of Physics: Conference Series, 423: 012028.