PARALLEL AND DISTRIBUTED COMPUTATIONS

(Syllabus)

Реквізити навчальної дисципліни

Level of higher education	First (undergraduate)
Branch of knowledge	12 Information technologies
Specialty	123 Computer engineering
Educational program	Computer systems and networks
Discipline status	Selective
Form of education	full-time (full-time) part-time
Year of training, semester	3rd year, spring semester
Scope of the discipline	4 credits 120 hours
Semester control/ control measures	Final score, MKR
Timetable	//rozklad.kpi.ua
Language of teaching	English
Information about	
head of the course / teachers	Lecturer: Ph.D., associate professor, Oleksandr Korochkin avcora@gmail.com

Program

1. Description of the educational discipline, its purpose, subject of study and learning outcomes

The discipline "Parallel and distributed computing" is aimed at students mastering technologies for developing parallel programs. Knowledge of the basics of parallel programming is necessary for creating software for parallel and distributed computer systems, real-time systems, Internet applications, mobile devices.

The discipline is a continuation of the normative discipline "Parallel programming".

The goal of teaching the discipline "Parallel and distributed computing" is to provide students with knowledge and skills in the field of practical software development for parallel computer and distributed systems. The educational discipline forms in students the competence of software development for computer systems with parallel or distributed architecture based on mastery of tools of modern languages and libraries of parallel programming at workplaces and units during future professional activities in the first position. According to the results of the study of the discipline, the student should be able to solve professional tasks and be able to evaluate the features of the structure of the PCS that are used; use methods of creating and evaluating parallel algorithms; to use tools of modern languages and parallel programming libraries to create software for modern PCs, to master the basic methods of creating programs based on the concept of processes, to be able to organize optimal interaction of flows.

ABILITY:

- development of software for computer systems with parallel or distributed architecture based on knowledge of modern languages and parallel programming libraries

- analyze the structure of a parallel computer system

- analyze the parallel properties of the problem being solved
- develop and evaluate parallel algorithms
- organize calculations in parallel and distributed computer systems
KNOWLEDGE:
- methods and means of organizing calculations in high-performance computer systems
- structures of modern parallel computer systems, organization of memory and connection of processors,
- methods of analysis, evaluation and presentation of parallel algorithms, concepts of development of parallel programs
- thread programming methods, thread concepts, thread operations,
- methods and means of organizing flow interaction,
- process interaction models,
- resolutions and methods of solving the task of mutual exclusion and synchronization of processes,
- methods of debugging parallel programs.
SKILLS:
- to create software for high-performance computer systems with various architecture,
- perform the construction of a parallel algorithm,
- analyze the effectiveness of the parallel algorithm, form flow algorithms,
- program streams and create a parallel program,
- to organize effective interaction of flows depending on the structure of the PCS,
- place streams on PCS processors (RCS nodes)
- perform debugging and execution of a parallel (distributed) program

2. Pre-requisites and post-requisites of the discipline (place in the structural and logical scheme of training according to the relevant educational program)

Necessary disciplines: "Programming", "Object-oriented programming", "System programming", "Data structures and algorithms", "Software engineering", "Algorithms and calculation methods", "Parallel programming"

Disciplines that are based on the learning outcomes of this discipline: System software, "Computer systems".

3. Content of the academic discipline

Chapter 1. Development of parallel programs. Life cycle

Topic 1.1 Development of a parallel mathematical algorithm

Topic 1.2 Development of flow algorithms

Topic 1.3 Development of flow interaction scheme

Topic 1.4 Program development

Chapter 2. Programming for PCS with shared memory (PCS SP). The use of low-level means of organizing the interaction of streams

Topic 2.1 The language of Hell. Semaphores. Atomis variables

Topic 2.2 Java language. Semaphores. Barriers

Topic 2.3 WinAPI library. Semaphores. Mutexes. Events

Topic 2.4 C# language. Semaphores. Mutexes. Events

Topic 2.5 Python language. Semaphores. Mutexes. Events

Chapter 3. OpenMP library.

3.1 Creation of streams

3.2 Interaction of flows

Chapter 4. Programming for PCS with shared memory. Use of critical sections to organize flow interaction

4.1 WinAPI library. Critical sections

4.2 Java language. Synchronized blocks

4.3 Language The C# language. C# language.

Chapter 5. Programming for PCs with shared memory. Using monitors to organize the interaction of flows

Topic 5.1 The language of Hell. Protected module

Topic 5.2 Java language. Monitor

Chapter 6. Programming for PCS with shared memory. Using monitors to organize the interaction of flows

Topic 6.1 The language of Hell. Protected module

Topic 6..2 Java language. Monitor

Chapter 7. Programming for PCS with distributed memory.

7.1. The language of Hell. Rendezvous mechanism

7.2 MRI library

Chapter 8 Programming for distributed PCs

Topic 8.1 Using sockets

Topic 8.2 Use of remote methods

4. Guidelines

The discipline study methodology requires the use of modern hardware and software related to parallel and distributed processing. As for the hardware component, parallel computer systems with a multi-core architecture, as well as distributed (cluster) systems should be considered first of all. Attention should be paid to the structural organization of modern multi-core processors, the *organization of multi-level cache memory and the connection of cores and cluster nodes. Laboratory work should be performed in a classroom equipped with such multi-core computers that are connected by a powerful network.*

Modern software tools that allow programming for parallel (distributed) computer systems are based on the use of parallel programming languages and libraries. These include Java, C#, and Ada languages. OpenMP, MPI, PVM. It is necessary to consider all such systems, because they differ both in terms of the concept of creating flows and in the means of organizing the interaction of flows.

As the base language, you can choose Java, which supports the classic approach to creating streams through special task modules, and also implements both models of process interaction organization through shared variables and message links. In addition, the resources of the Java language allow you to create software for distributed systems.

5. **Educational materials and resources**

Базова:

1. *Loutsky G., Zhukov I., Korochkin A. Parallel Computing. – Kyiv, Kornechuk, 2007. - 216 pp.*

Додаткова:

2. *Korochkin O.Multicore programming in Ada. Навч. посібник для здобувачів ступеня бакалавр за спеціальністю 123 «Комп'ютерні системи та мережі» Київ, НТУУ-КПІ, 2018.- 114 с. Електронний ресурс. Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 6 від 31.01.2020 р.) за поданням Вченої ради ФІОТ (протокол № 4 від 25.11.2019 р.), https://comsys.kpi.ua/metodichni-vkazannya-po-disciplinam*

Education content

5. Methods of mastering an educational discipline (educational component)

Themas	*Hours*				
	Total				
		Lectures		*Labs*	*IW*

1	*2*	*3*	*4*	*5*	*6*
Chapter 1. Development of parallel programs. Life cycle					
Topic 1.1 Development of a parallel mathematical algorithm	*1*	*0,5*	*-*	*-*	*0,5*
Topic 1.2 Development of flow algorithms	*1*	*0,5*	*-*	*-*	*0,5*
Topic 1.3 Development of flow interaction scheme	*1*	*0,5*			*0,5*
Topic 1.4 Program development	*1*	*0,5*			*0,5*
Total in 1	***4***	***2***	***-***	***-***	***2***
Chapter 2. Programming for PCS SM. The use of low-level means of organizing the interaction of threads					
Topic 2.1 The language of Ada. Semaphores. Atomic variables	*3*	*2*	*-*	*-*	*1*
Topic 2.2 Java language. Semaphores. Barriers	*3*	*2*	*-*	*-*	*1*

1	2	3	4	5	6
Topic 2.3 WinAPI library. Semaphores. Mutexes. Events	2	2			
Topic 2.4 C# language. Semaphores. Mutexes. Events	3	2			1
Topic 2.5 Python language. Semaphores. Mutexes. Events	3	2			1
Total in 2	***14***	***10***	***-***	***-***	***4***
Chapter 3. OpenMP library					
3.1 Creation of streams	3	1	-	-	2
3.2 Interaction of flows	3	1	-	-	2
Total in 3	***6***	***2***			***4***
Chapter 4. Programming for PCS SM. Use of critical sections to organize flow interaction					
4.1 WinAPI library. Critical sections	10	2		3	5
4.2 Java language. Synchronized blocks	15	2	-	3	10
4.3 C# language. Castles	15	2		2	11
Total in 4	***40***	***6***	***-***	***8***	***26***
Chapter 5. Programming for PCs with shared memory. Using monitors to organize the interaction of flows					
Topic 5.1 The language of Ada. Protected module	4	2	-	2	2
Topic 5.2 Java language. Monitor	2	2	-	-	
Total in 5	***6***	***4***	***-***	***2***	***2***
Chapter 6. Programming for PCS SM. Using Atomic variablesл					
Topic 6.1 The language of Adal. Atomic variables	8	2		2	4
Topic 6.2 Java language. Atomic, Volatile variables	8	2	-	-	6
Total in 6	***16***	***4***	-	***2***	***12***
Chapter 7. Programming for the PCS SM					
7.1. The language of Hell. Rendezvous mechanism	5	2	-	-	6
7.2 MRI library	5	2	-	-	6
Total in 7	***10***	***4***	-	***3***	***3***
Chapter 8. Programming for the PKS LM					
Topic 8.1 Using sockets	7	2	-	-	3
Topic 8.2 Use of remote methods	7	2	-	-	4
Total in 8	***14***	***4***	-	***3***	***7***
MCW	***2***	***2***			***2***
Test	***8***				***8***
Total in semester:	***120***	***36***	***-***	***18***	***66***

Lectures

№	*The name of the topic of the lecture and a list of main questions (a list of didactic tools, references to literature and tasks on IW)*
1	***Development of parallel programs. Life cycle.*** *Peculiarities of the task of parallel programming: research on parallelism tasks, development of parallel algorithms, programming of parallel processes, interaction of processes, solution of the problem of mutual exclusion, synchronization of processes, placement of processes on processors, planning of execution of processes, debugging and execution [1, p.37-38.]* *Features of the development life cycle of programs for parallel systems.* *Stages of development: analysis of requirements, design, implementation, testing, production, modification, support [1, c.38-48].* *SRS: Perform an analysis of the life cycle of program development for the operation of finding the maximum element of the matrix [1, p. 66-67].*
2	***Analysis and construction of parallel algorithms*** *Peculiarities of the task of parallel programming: research on parallelism tasks, development of parallel algorithms, programming of parallel processes, interaction of processes, solution of the problem of mutual exclusion, synchronization of processes, placement of processes on processors, planning of execution of processes, debugging and execution [1, p.37-38.]* *Features of the development life cycle of programs for parallel systems.* *Stages of development: analysis of requirements, design, implementation, testing, production, modification, support [1, c.38-48].* *SRS: Perform an analysis of the life cycle of program development for the operation of finding the maximum element of the matrix [1, p. 66-67].*
3	***Programming for CS with shared memory. Low-level tools. The language Adal*** *Use of semaphores. Synchronized_Object type [1, c.38-44].* *SRS: Packages Asynchronous_Task_Control, Synchronous_Task_Control, [1, c. 76-77].*
4	***Programming for CS with shared memory. Low-level tools. Language C#*** *Use of semaphores. Semaphores class. Using mutexes. Mutex class. Using events. Event class. [1, c. 42-45].* *SRS: Develop parallel algorithms for given vector-matrix operations [1, c. 76-77].*
5	***Programming for CS with shared memory. Low-level tools. Python language. Use of*** *semaphores. Semaphires class. Using mutexes. Mutex class. Using events. Event class. [1, c. 42-45].* *SRS: Analyze different approaches to programming processes (flows) in modern parallel programming languages and libraries [1, c. 26-54].*
6	***Programming for CS with shared memory. Low-level means. WinAPI library****. Use of semaphores. Functions of Semaphires. Using mutexes. Mutex class. Using events. Event class. [1, c. 42-45].* *[1, c. 26-55]. SRS: Programming calculations in the Python language [1, p. 50]*
7	***Programming for CS with shared memory. Critical sections. WinAPI*** *Library Processes*

	in special parallel programming libraries. Win32 library. MPI library. PVM library. OpenMP library [1, c.26-55]. *SRS: Programming calculations in the PVM library [5, p. 30]*
8	***Programming for CS with shared memory. Critical sections. Java language.*** *Tasks of process interaction: synchronization and data exchange. Peculiarities of synchronization mechanisms. [1, c. 50-54]* *SRS: Peculiarities of interaction of processes [1, c.28-29]*
9	***Programming for CS with shared memory. Critical sections. C# language.*** *Construction lock.. [Additional 3, c. 124-130].* *SRS: Task Factory-Store [1, c.76-79]*
10	***Programming for CS with shared memory. Critical sections. Python language..*** *[1, c.78-82].*
11	***Programming for CS with shared memory. Monitors. The language of Hell.*** *Protected modules. Organization of flow interaction. [1, c. 99, 126].* *SRS: Implementation and application of monitors in the Python language [16, c. 131-167].*
12	***Programming for CS with shared memory. Monitors. Java language*** *Monitors in Java. Modifiers private, synchronized. wait(), notify(), notifyAll(), [1, c. 99, 126]* *SRS: Implementation and use of monitors in the C# language [1, c. 131-167].*
13	***Programming for CS with shared memory. OpenMP library*** *Barrier, shared, lock pragmas. Application. [1, c. 82-90].* *SRS: Implementation of the mechanism of critical sections in the OpenMP library [1, c. 127]*
14	***Programming for CS with shared memory. Atomic variables*** *The general concept of semaphores. Semaphore type. Operations with semaphores. Use of semaphores. Realization. Examples. [1, c. 91-100].* *SRS: atomic variables in Python [1, c.76-79]*
15	***Programming for CS with distributed memory. The language Ada*** *The general concept of the rendezvous mechanism. Constructions entry, accep [1, c.102-104].* *SRS: Using the select operator [1, c.76-79]*
16	***Programming for CS with distributed memory. Library of MPI*** *Implementation of the message model.* *[1, c. 104-110] SRS: Task Plant - Shop [1, c.76-79]*
17	*Programming for RKS .Sockets* *The concept of sockets. Types of sockets. Data transfer using sockets. [1, c.152-177]* *SRS: Implementation of sockets in the Java language [1, c.168-177]*
18	***Programming for RCS .Remote Procedures (RPC)*** *Concept of remote methods. Implementation of sockets in the library [1, c.153-176]* *SRS: PVM Library [1, 162-177] , [5, 117-122]*

Labs

The main tasks of the cycle of laboratory classes are for students to acquire the necessary practical skills for developing parallel (distributed) programs based on the concept of interacting streams, using the mechanisms of organizing the interaction of streams of modern languages and parallel programming libraries Java, Ada, C#, WinAРИ, OpenMP, MPI on the basis of modern computer equipment in the form of powerful multi-core systems.

№	*Labs*	*Hours*
1	*Programming of calculations for PCS SM. Use of low-level means of interaction of flows*	*3*
2	*Programming calculations for PCS SM Using monitors*	*3*
3	*Programming calculations for PCS SM OpenMP library*	*3*
4	*Programming calculations for PCS LM Message model*	*4*
5	*Programming calculations for RCS*	*5*
	Total:	*18*

Independent work of the student

№	*Themes*	*Hours*
1	*Development of parallel and distributed programs. Python language[11. p 153]*	*22*
2	*Development of distributed programs. C# language*	*22*
3	*Development of distributed Ada programs [1. p. 30-34, 117-123.]*	*22*

Policy and control

6. Policy of academic discipline (educational component)

The following factors are taken into account when enrolling and evaluating laboratory works:

• Complete completion of the task for laboratory work according to the individual option;

• Timeliness of laboratory work according to the schedule;

• Independent performance of laboratory work and absence of signs of plagiarism;

• Answers to questions about the content of laboratory work during its defense.

When evaluating control works, the following are taken into account:

• Correctness and completeness of tasks;

• Number of completed tasks under limited time conditions;

• Independent performance of tasks and absence of signs of plagiarism;

• The number of test execution attempts that precede the one being evaluated.

To prepare for the tests, students receive a list of theoretical questions and the content of typical problems that will be in the tests.

During the first and second attestation, the number of laboratory works and control works enrolled at the time of the attestation is taken into account.

8. Types of control and rating system for evaluating learning outcomes (RSV)

The RSV in the discipline, the semester control of which is provided in the form of a credit, for full-time education is developed according to the RSO-1 type and includes the evaluation of measures of current control in the discipline during the semester.

The rating evaluation of the acquirer consists of the points received by the acquirer based on the results of current control measures, incentive and penalty points.

Applicants who have fulfilled all the conditions for admission to the test and have a rating of 60 or more points receive a rating corresponding to the rating without additional tests.

With students who have fulfilled all the admission requirements and have a rating score of less than 60 points, as well as with those students who wish to improve their rating score, the teacher conducts a semester control in the form of a credit test or interviews

Types of control in the academic discipline include:

Laboratory work

Independent performance of three to five laboratory works (optional) is planned. The topics of laboratory works are coordinated in time and content with the topics of lectures. Carrying out laboratory work in its entirety allows you to acquire practical skills in thread programming using modern parallel programming languages and libraries.

Current control

It is planned to carry out modular control work (MKR)

Semester control

Assessment is conducted in the form of an interview with the student to objectively determine the level of knowledge, skills and practical skills acquired during the semester

The student's semester rating consists of the points he receives for the types of work in accordance with Table 1.

Table 1

Assessment of individual types of student academic work (in points)

Educational work		**Total by type of work**
Performance and protection of laboratory work No. 1		*7..16*
Performance and protection of laboratory work No. 2		*7..16*
Performance and protection of laboratory work No. 3		*7..16*
Performance and protection of laboratory work No. 4		*7..16*
Performance and protection of laboratory work No. 5		*7..16*
Performance of laboratory work Rl		*35..80*
Modular control work (MCW) Rk		*5..20*
Total i sesmester	***Rп= Rл + Rк***	40..100
		100
Credit in the form of an interview with the student (optional)		20

The student's individual current rating (Rп) consists of the points he receives for performing laboratory work and MKR. During the semester, students perform a selected number (3-5) of laboratory works. The maximum number of points for each laboratory work is 16. Points are awarded for:

- theoretical component – 8 points,

- practical component - 8 points.

The maximum possible score for laboratory work is 16 points. The maximum number for all laboratory works is 5x12=60 points.

Calculation of the scale size (R) of the rating.

The sum of the weighted points of control measures during the semester is:

Rp= Rl + Rk,

where Rp is the student's semester rating (MCR, laboratory work).

A necessary condition for a student's admission to credit is his individual semester rating (Rp), not less than 40 points, and the absence of arrears from laboratory work and MKR. If the mentioned requirements are not met, the student will not be admitted to the credit.

Table of correspondence of rating points to grades on the university scale:

Points	*Rating*
100-95	*Perfectly*
94-85	*Very well*
84-75	*Okay*
74-65	*Satisfactorily*
64-60	*Enough*
<60	*Unsatisfactorily*
Admission conditions not met	*Not allowed*

7. Additional information on the discipline (educational component)

As part of the study of the discipline "Parallel and distributed computing", it is allowed to credit the points obtained as a result of distance courses on the "Coursera" platform, provided that the program of this course has been approved in advance with the teacher and provided that an official certificate has been obtained.

Working program of the academic discipline (syllabus):

Compiled by Oleksandr Volodymyrovych Korochkin, Associate Professor, Ph.D.,
Approved by the Department of Computing (Protocol No. 10 dated 05/25/2022)
Agreed by the Methodical Commission of the faculty (protocol No. 10 dated 06.9.2022).......