Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

КАФЕДРА КИБЕРНЕТИКИ

ОДОБРЕНО УМС ИИКС

Протокол № УМС-575/01-1

от 30.08.2021 г.

РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Направление подготовки (специальность) [1] 09.04.04 Программная инженерия

Семестр	Трудоемкость, кред.	Общий объем курса, час.	Лекции, час.	Практич. занятия, час.	Лаборат. работы, час.	В форме практической подготовки/ В	СРС, час.	КСР, час.	Форма(ы) контроля, экз./зач./КР/КП
3	4	144	16	16	0		112	0	З
Итого	4	144	16	16	0	0	112	0	

Москва, 2022

АННОТАЦИЯ

В курсе “Параллельные вычисления” изучаются теоретические основы построения параллельных алгоритмов и их реализации на базовых параллельных структурах.

Основное внимание уделено развитию “параллельного“ мышления, использованию каскадных функций и специфических механизмов для параллельных структур.

1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ

Учебная задача: Научить студентов выявлять параллелизм в задаче, распределять работу между параллельными процессами и осуществлять их взаимодействие, писать и отлаживать параллельные программы.

Целью освоения дисциплины является достижение следующих результатов образования:

Знания:

на уровне представлений: Развитие “параллельного “ мышления, использование каскадных функций и специфических механизмов для параллельных структур.

на уровне воспроизведения: Теоретические основы построения параллельных алгоритмов и их реализации на базовых параллельных структурах.

на уровне понимания: выявлять параллелизм в задаче, распределять работу между параллельными процессами и осуществлять их взаимодействие, писать и отлаживать параллельные программы.

Умения:

теоретические – Теоретические основы построения параллельных алгоритмов и их реализации на базовых параллельных структурах.

2. МЕСТО УЧЕБНОЙ ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВО

Дисциплина «Параллельные вычисления» является обязательной дисциплиной инженерной подготовки студента. Дисциплина не требует специальной начальной подготовки, выходящей за рамки курса математики и информатики программы среднего образования.

В свою очередь, дисциплина является необходимым дополнением курсов, демонстрируя методы и средства выполнения инженерных расчетов, связанных с общими методами математических вычислений, методами оптимизации, методами моделирования и проектирования сложных систем.

3. ФОРМИРУЕМЫЕ КОМПЕТЕНЦИИ И ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

Универсальные и(или) общепрофессиональные компетенции:

Код и наименование компетенции	Код и наименование индикатора достижения компетенции
ОПК-2 [1] – Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с	З-ОПК-2 [1] – Знать: современные интеллектуальные технологии, инструментальные среды, программно-технические платформы для решения профессиональных задач

использованием современных интеллектуальных технологий, для решения профессиональных задач	У-ОПК-2 [1] – Уметь: обосновывать выбор современных интеллектуальных технологий, разрабатывать оригинальные программные средства для решения профессиональных задач В-ОПК-2 [1] – Владеть: методами разработки оригинальных программных средств, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач
УКЦ-1 [1] – Способен решать исследовательские, научно-технические и производственные задачи в условиях неопределенности, в том числе выстраивать деловую коммуникацию и организовывать работу команды с использованием цифровых ресурсов и технологий в цифровой среде	З-УКЦ-1 [1] – Знать современные цифровые технологии, используемые для выстраивания деловой коммуникации и организации индивидуальной и командной работы У-УКЦ-1 [1] – Уметь подбирать наиболее релевантные цифровые решения для достижения поставленных целей и задач, в том числе в условиях неопределенности В-УКЦ-1 [1] – Владеть навыками решения исследовательских, научно-технических и производственных задач с использованием цифровых технологий

Профессиональные компетенции в соотвествии с задачами и объектами (областями знаний) профессиональной деятельности:

Задача профессиональной деятельности (ЗПД)	**Объект или область знания**	**Код и наименование профессиональной компетенции; Основание (профессиональный стандарт-ПС, анализ опыта)**	**Код и наименование индикатора достижения профессиональной компетенции**
производственно-технологический			
организация обеспечения индустриального производство программного обеспечения для информационно-вычислительных систем различного назначения..	обеспечение внедрения усовершенствованных методов и алгоритмов обработки данных в информационно-вычислительных системах; - улучшение технологии параллельных, высокопроизводительных и распределенных информационно-вычислительных систем; - организация процесса промышленного тестирования программного обеспечения; - внедрение языков программирования и их трансляторов; -	ПК-14 [1] - способен применять навыки программной реализации систем с параллельной обработкой данных и высокопроизводительных систем *Основание:* Профессиональный стандарт: 06.017, 06.028	З-ПК-14[1] - Знать: технологии программной реализации систем с параллельной обработкой данных и высокопроизводительных систем ; У-ПК-14[1] - Уметь: применять навыки программной реализации систем с параллельной обработкой данных и высокопроизводительных систем ; В-ПК-14[1] - Владеть: навыками программной реализации систем с параллельной обработкой данных и высокопроизводительных систем

	усовершенствование сетевых протоколов и сетевых служб; - организация использования операционных систем.		
организация обеспечения индустриального производство программного обеспечения для информационно-вычислительных систем различного назначения..	обеспечение внедрения усовершенствованных методов и алгоритмов обработки данных в информационно-вычислительных системах; - улучшение технологии параллельных, высокопроизводительных и распределенных информационно-вычислительных систем; - организация процесса промышленного тестирования программного обеспечения; - внедрение языков программирования и их трансляторов; - усовершенствование сетевых протоколов и сетевых служб; - организация использования операционных систем.	ПК-18 [1] - способен применять навыки создания компонент операционных систем и систем реального времени *Основание:* Профессиональный стандарт: 06.017, 06.028	З-ПК-18[1] - Знать: технологии создания компонент операционных систем и систем реального времени ; У-ПК-18[1] - Уметь: применять навыки создания компонент операционных систем и систем реального времени ; В-ПК-18[1] - Владеть: навыками создания компонент операционных систем и систем реального времени
проектный			
обеспечение и организация проектирования, разработки и эксплуатации информационных систем и программных продуктов целевого назначения;	обеспечение внедрения усовершенствованных методов и алгоритмов обработки данных в информационно-вычислительных системах; - улучшение технологии параллельных, высокопроизводительных и распределенных информационно-вычислительных систем; - организация процесса промышленного тестирования программного обеспечения; - внедрение	ПК-8 [1] - способен проектировать системы с параллельной обработкой данных и высокопроизводительные системы, и их компоненты *Основание:* Профессиональный стандарт: 06.016, 06.028	З-ПК-8[1] - Знать: методы и инструменты проектирования систем с параллельной обработкой данных и высокопроизводительные систем, и их компоненты ; У-ПК-8[1] - Уметь: проектировать системы с параллельной обработкой данных и высокопроизводительные системы, и их компоненты ; В-ПК-8[1] - Владеть: методами и инструментами

	языков программирования и их трансляторов; - усовершенствование сетевых протоколов и сетевых служб; - организация использования операционных систем.		проектирования систем с параллельной обработкой данных и высокопроизводительные систем, и их компоненты

4. СТРУКТУРА И СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

Разделы учебной дисциплины, их объем, сроки изучения и формы контроля:

№ п.п	Наименование раздела учебной дисциплины	Недели	Лекции/ Практ. (семинары)/ Лабораторные работы, час.	Обязат. текущий контроль (форма*, неделя)	Максимальный балл за раздел**	Аттестация раздела (форма*, неделя)	Индикаторы освоения компетенции
	3 Семестр						
1	Часть 1	1-8		ДЗ-3,КР-4,ДЗ-7,КР-12	20	КИ-8	В-ПК-18, З-ПК-8, У-ПК-8, В-ПК-8, З-УКЦ-1, У-УКЦ-1, В-УКЦ-1, З-ОПК-2, У-ОПК-2, В-ОПК-2, З-ПК-14, У-

							ПК-14, В-ПК-14, З-ПК-18, У-ПК-18
2	Часть 2	9-16		ДЗ-11,КР-12,ДЗ-15,КР-16	30	КИ-16	З-ОПК-2, В-ОПК-2, З-ПК-14, У-ПК-14, В-ПК-14, З-ПК-18, У-ПК-18, В-ПК-18, З-ПК-8, У-ПК-8, В-ПК-8, З-УКЦ-1, У-УКЦ-1, В-УКЦ-1, У-ОПК-2
	Итого за 3 Семестр		16/16/0		50		
	Контрольные				50	З	З-

	мероприятия за 3 Семестр						ОПК-2, У-ОПК-2, В-ОПК-2, З-ПК-14, У-ПК-14, В-ПК-14, З-ПК-18, У-ПК-18, В-ПК-18, З-ПК-8, У-ПК-8, В-ПК-8, З-УКЦ-1, У-УКЦ-1, В-УКЦ-1

* – сокращенное наименование формы контроля

** – сумма максимальных баллов должна быть равна 100 за семестр, включая зачет и (или) экзамен

Сокращение наименований форм текущего контроля и аттестации разделов:

Обозначение	**Полное наименование**
ДЗ	Домашнее задание
КИ	Контроль по итогам
З	Зачет

КАЛЕНДАРНЫЙ ПЛАН

Недели	Темы занятий / Содержание	Лек., час.	Пр./сем., час.	Лаб., час.
	3 Семестр	16	16	0
1-8	**Часть 1**	8	8	
1 - 2	**Параллельные вычисления и распределенная обработка данных.** Классификация вычислительных систем параллельной обработки данных.	Всего аудиторных часов		
		2	2	
		Онлайн		
3 - 4	**Управление оперативной памятью ЭВМ.** Адаптация виртуальной памяти. Обеспечение когерентности данных в параллельных вычислителях.	Всего аудиторных часов		
		2	2	
		Онлайн		
5 - 6	**Архитектура микропроцессоров.** Параллелизм на уровне машинных команд.	Всего аудиторных часов		
		2	2	
		Онлайн		
7 - 8	**Вычислительные кластеры.** Характеристики коммуникационных технологий.	Всего аудиторных часов		
		2	2	
		Онлайн		
9-16	**Часть 2**	8	8	
9 - 12	**Языки и системы параллельного программирования.** Система MPI. Стандарты Open MP. Фортран DVM.	Всего аудиторных часов		
		4	4	
		Онлайн		
13 - 14	**Прикладное параллельное программирование.** Система Норма, дискретное моделирование.	Всего аудиторных часов		
		2	2	
		Онлайн		
15 - 16	**Параллельные алгоритмы.** Редукционные алгоритмы. Распараллеливание алгоритма общей рекурсии 1-го порядка.	Всего аудиторных часов		
		2	2	
		Онлайн		

Сокращенные наименования онлайн опций:

Обозначение	Полное наименование
ЭК	Электронный курс
ПМ	Полнотекстовый материал
ПЛ	Полнотекстовые лекции
ВМ	Видео-материалы
АМ	Аудио-материалы
Прз	Презентации
Т	Тесты
ЭСМ	Электронные справочные материалы
ИС	Интерактивный сайт

ТЕМЫ ЛАБОРАТОРНЫХ РАБОТ

Недели	Темы занятий / Содержание
	3 Семестр
1	Занятие 1. Эффективное программирование для микропроцессоров. Учет локальных вычислений.
2	Занятие 2. Векторное программирование. Распараллеливание циклов методом координат.
3	Занятие 3. Контрольная работа.
4	Занятие 4. Программирование для мультипроцессорных систем. Образование параллельных процессов. Протоколы передачи сообщений. Семафоры.

5. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Лекции
Семинарские занятия
Домашние задания
Контрольные работы

6. ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

Фонд оценочных средств по дисциплине обеспечивает проверку освоения планируемых результатов обучения (компетенций и их индикаторов) посредством мероприятий текущего, рубежного и промежуточного контроля по дисциплине.

Связь между формируемыми компетенциями и формами контроля их освоения представлена в следующей таблице:

Компетенция	Индикаторы освоения	Аттестационное мероприятие (КП 1)
ОПК-2	З-ОПК-2	З, КИ-8, КИ-16
	У-ОПК-2	З, КИ-8, КИ-16
	В-ОПК-2	З, КИ-8, КИ-16
ПК-14	З-ПК-14	З, КИ-8, КИ-16
	У-ПК-14	З, КИ-8, КИ-16
	В-ПК-14	З, КИ-8, КИ-16
ПК-18	З-ПК-18	З, КИ-8, КИ-16
	У-ПК-18	З, КИ-8, КИ-16
	В-ПК-18	З, КИ-8, КИ-16
ПК-8	З-ПК-8	З, КИ-8, КИ-16
	У-ПК-8	З, КИ-8, КИ-16
	В-ПК-8	З, КИ-8, КИ-16
УКЦ-1	З-УКЦ-1	З, КИ-8, КИ-16
	У-УКЦ-1	З, КИ-8, КИ-16
	В-УКЦ-1	З, КИ-8, КИ-16

Шкалы оценки образовательных достижений

Шкала каждого контрольного мероприятия лежит в пределах от 0 до установленного максимального балла включительно. Итоговая аттестация по дисциплине оценивается по 100-балльной шкале и представляет собой сумму баллов, заработанных студентом при выполнении заданий в рамках текущего и промежуточного контроля.

Итоговая оценка выставляется в соответствии со следующей шкалой:

<table>
<tr><th>Сумма баллов</th><th>Оценка по 4-ех балльной шкале</th><th>Оценка ECTS</th><th>Требования к уровню освоению учебной дисциплины</th></tr>
<tr><td>90-100</td><td>5 – «отлично»</td><td>A</td><td>Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно увязывать теорию с практикой, использует в ответе материал монографической литературы.</td></tr>
<tr><td>85-89</td><td rowspan="3">4 – «хорошо»</td><td>B</td><td rowspan="3">Оценка «хорошо» выставляется студенту, если он твёрдо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос.</td></tr>
<tr><td>75-84</td><td>C</td></tr>
<tr><td>70-74</td><td rowspan="2">D</td></tr>
<tr><td>65-69</td><td rowspan="2">3 – «удовлетворительно»</td><td rowspan="2">Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала.</td></tr>
<tr><td>60-64</td><td>E</td></tr>
<tr><td>Ниже 60</td><td>2 – «неудовлетворительно»</td><td>F</td><td>Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала, допускает существенные ошибки. Как правило, оценка «неудовлетворительно» ставится студентам, которые не могут продолжить обучение без дополнительных занятий по соответствующей дисциплине.</td></tr>
</table>

Оценочные средства приведены в Приложении.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

ОСНОВНАЯ ЛИТЕРАТУРА:

1. 004 П18 Параллельные вычисления на GPU. Архитектура и программная модель CUDA : учебное пособие, Москва: Издательство Московского университета, 2012

2. ЭИ В12 Основы программирования МРР-архитектур : учебно-методическое пособие, А. Б. Вавренюк, В. В. Макаров, Е. В. Чепин, Москва: НИЯУ МИФИ, 2010

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА:

1. 004 Н50 Параллельное программирование для многопроцессорных вычислительных систем : , С.А. Немнюгин, О.Л. Стесик, СПб: БХВ-Петербург, 2002

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ:

Специальное программное обеспечение не требуется

LMS И ИНТЕРНЕТ-РЕСУРСЫ:

https://online.mephi.ru/

http://library.mephi.ru/

8. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

Специальное материально-техническое обеспечение не требуется

9. УЧЕБНО-МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ СТУДЕНТОВ

Целью практикума является усвоение и закрепление теоретического материала курса и приобретение практических навыков разработки параллельных алгоритмов и написания параллельных программ.

Параллельных программы реализуются на языке АДА. Этот паскалеподобный язык содержит высокоуровневые средства для описания параллельных процессов, что облегчает написание и отладку параллельных программ.

Практикум состоит из четырех (трех – в зависимости от усвоения материала) работ, каждая из которых содержит индивидуальные задания. Гипотетически редполагается, что в распоряжении студента имеется вычислительная система с большим количеством процессоров и блоков памяти. Требуется разработать такой алгоритм решения предложенных задач и реализовать такую параллельную программу, которая сможет максимально загрузить процессоры полезной работой по решению задач и по возможности равномерно распределить нагрузку. Другими словами, нужно построить высокопараллельный алгоритм, который радикально повышает скорость решения задачи.

В большинстве случаев требуется, чтобы разработанная программа являлась общим решением задачи и имела, по крайней мере, два независимых параметра P- число процессоров и N –размерность обрабатываемых данных (например, размерность матриц или одномерных массивов).

Предъявляемый для сдачи вариант программы может обрабатывать входные данные небольшого объема, но, тем не менее, программа должна осуществлять общее решение поставленной задачи. Так не допускается, чтобы программа правильно работала только при выполнении условия N=P.

Предполагается, что программа реализуется на вычислительной системе одного из следующих трех видов

1. Параллельная система с непосредственными связями между процессорам и.
2. Гиперкубическая система
3. Векторный процессор.
4. Систоллическая структура.

В первом случае процессоры могут работать по индивидуальной программе и обрабатывать свои собственные данные. Процессоры имеют доступ друг к другу, к локальной памяти друг друга и к главной памяти системы. Для краткости, мы будем называть эту структуру параллельные процессоры, хотя все три архитектуры содержат параллельные процессоры.

Во второй архитектуре процессоры находятся в узлах гиперкуба k-ого порядка и имеют непосредственные связи только с k процессорами, ближайшими по одной из k размерностей гиперкуба. Таким образом, не любые два процессора имеют непосредственные связи друг с другом, и, в общем случае, передача данных из одного процессора в другой проходит через промежуточные процессоры. Эта система должна иметь средства коммутации и управления для прокладывания пути из любого процессора в любой.

В третьей архитектуре процессоры работают под жестким управлением и в данный момент времени все выполняют одну и ту же команду. Процессоры, входящие в процессорную матрицу, получают данные из локальных регистров или локальных регистров и не имеют непосредственного доступа к главной памяти и другим ресурсам системы в частности к системе ввода-вывода. В большинстве задач данной работы она будет пониматься более узко, а именно как векторный процессор.

В четвёртом случае система представляет собой сеть процессоров, каждый из которых может обмениваться информацией только с соседними процессорами, причем этот обмен реализуется при помощи специальной аппаратуры, в результате чего он не занимает процессорного времени. Процессоры сети наделены локальной памятью, но не имеют доступа к главной памяти системы. Связь с главной памятью системы осуществляет управляющий компьютер (host), который связан с первым процессором сети.

Предполагается, что задания первой работы реализуются на системе с архитектурой параллельные процессоры (первый вариант). Вид системы, на которой реализуется задания второй и третьей работы, задается в соответствующих заданиях.

Первая работа содержит задачи, в решении которых еще не используются специальные функции и алгоритмы. На ней студент осваивает программирование параллельных процессов на языке АДА и основные подходы к построению параллельных алгоритмов. В основном задания посвящены различным приближенным методам нахождения корней функции, методам приближенного вычисления интегралов и некоторым классическим задача по программированию.

Вторая работа связана с методами параллельного умножения матриц, механизмами бесконфликтного доступа к векторной памяти, с моделированием работы параллельных вычислительных систем различных типов и использованием специальных функций.

Третья работа посвящена параллельным и векторным методам решения систем линейных уравнений и другим задачам линейной алгебры.

Примеры параллельных программ приводятся на лекциях и семинарах.

Разработка параллельного алгоритма и программы включает следующие основные этапы.

1. Декомпозиция задачи на подзадачи меньшей размерности, которые можно выполнять параллельно (Partition). Это разбиение должно обеспечить достижение определенного порядка временной сложности. Если размерность задачи N, то оценками временной сложности для практически используемых алгоритмов могут служить следующие выражения o(N2), o(N), o(Nlog N), o(log N). Декомпозиция задачи снижает ее реальную размерность до величины (N/P) и позволяет достичь требуемой оценки временной сложности.

2. Определение способа получения общего результата из частичных результатов, полученных путем выполнения подзадач.

3. Выбор схемы коммуникаций и способа синхронизации параллельных процессов. Принятые здесь решения должны обеспечить корректность выполняемых вычислений и исключение тупиковых ситуаций.

4. Склеивание (agglomeration). Если необходимо, подзадачи объединяются в более крупные с тем, чтобы удовлетворить требованиям по производительности и доступному объему аппаратуры.

5. Раскладка процессов по процессорам (mapping).

Декомпозицию (или распределение работы между процессами) подразделяют на статическую и динамическую. При статической декомпозиции работу делят между процессами заранее, до начала обработки, стремясь равномерно загрузить процессы и как можно дольше сохранять большую их загрузку. Для этого этапа существует другое название-балансировка нагрузки. При динамической балансировке нагрузки задания выдает некий центральный процесс, который обычно называют Master, в ответ на запросы освободившихся рабочих задач (workers).

Студент должен изучить теоретический материал, относящийся к теме задания, разработать параллельные алгоритмы решения полученных им задач, написать и отладить параллельную программу и сдать ее преподавателю в течение семестра. Варианты программ, приведенные на лекциях и семинарах могут использоваться только как фрагменты заданий. но не как задания целиком.

Требования к построению и оформлению программы

1. Программа должна использовать такие конструкции языка АДА, которые обеспечивают параллелизм работы процессов.

2. Построение программы должно следовать общему принципу, заложенному в модели параллелизма языка АДА, согласно которому пассивные процессы не потребляют ресурсов процессоров. Поэтому не допускаются реализации синхронизации процессов за счет введения временных задержек и процессов ожидания событий путем циклического опроса некоторых признаков.

3. Программа должна выводить протокол работы основных процессов, отражающий следующие этапы : старт процесса, получение входных данных, вывод результатов, сообщение о завершении работы. Протокол должен подтверждать наличие параллелизма в работе программы.

4. Работа программы должна демонстрироваться на специально подготовленных тестовых данных небольшого объема. Форма вывода результатов должна облегчать установление факта их правильности.

5. При проведении приближенных вычислений программа должна выводить как полученные результаты, так и эталонные значения с достаточным количеством значащих разрядов. Это является подтверждением достижения необходимой точности.

6. При сдаче программы студент должен суметь объяснить как принцип работы программы, так и назначение, и работу отдельных ее конструкций.

7. Студент должен обосновать достижение уровня временной сложности, на которую ориентирована данная реализация программы. Поскольку в языке АДА отсутствуют средства одновременного запуска процессов, и одновременной пересылки данных многим процессам (режим broadcast), имеющиеся во многих других системах, то соответствующие циклы в программе можно не учитывать при оценке ее временной сложности.

Примеры программ.

Приведенные примеры демонстрируют основные подходы к решению задач и использование средств языка АДА. Рассмотренные задачи являются заведомо упрощенными и не могут быть использованы в качестве примеров решения домашних заданий.

Задача 1. Построить параллельный алгоритм и написать параллельную программу для нахождения с заданной точностью корня функции f(x) на отрезке [a, b] методом дихотомии (бисекции), если известно, что на данном отрезке существует точно один корень.

Последовательный алгоритм циклически делит интервал пополам, определяет в какой половине лежит корень и уменьшает интервал путем соответствующего сдвига одной из его границ. Цикл продолжает свою работу до тех пор, пока длина интервала не станет меньше заданной, или корень не окажется на границе интервала.

Декомпозиция. Пусть имеется P процессов. Разобьем интервал [a, b] на P равных отрезков. Каждый процесс обрабатывает свой отрезок и определяет, находится ли корень в пределах его отрезке. Для этого он вычисляет значения функции на границах отрезка и проверяет, отличаются ли знаки этих значений.

Синхронизация. Если некий процесс определил, что корень находится в пределах его отрезка, то он передает процессу Master значения границ отрезка. Поскольку такой отрезок по условию задачи только один, то процесс Master принимает только одно сообщение в данном цикле. Но, если корень окажется на границе отрезка, то сообщения могут прислать два соседних процесса.

Чтобы преодолеть эту неоднозначность, сохранив условие приема только одного сообщения процессом Master, поступим следующим образом. Пусть все процессы проверяют наличие корня только на одной из границ отрезка, например, на левой границе. Но тогда правую границу интервала не проверяет ни один из рабочих процессов. Это может на себя взять процесс Master. Таким образом, задача синхронизации решена.

Получив границы нового интервала, процесс Master проверяет достигнута ли необходимая точность и, если нет, то вновь делит полученный интервал на P отрезков и передает их рабочим процессам.

Это решение позволяет загрузить произвольно большое число процессоров, причем все они работают в течение всего времени работы программы, что дает возможность существенно уменьшить время решения задачи. Текст программы приведен ниже.

```
with ADA.text_io,ada.integer_text_io,ada.float_text_io;
use ADA.text_io,ada.integer_text_io,ada.float_text_io;
-- with Gnat.IO ; use Gnat.IO;

procedure dechot1 is
z:float;
nprog:integer := 9;
-- a,b: float;
function dechotomy(l,r,epsilon:float) return float is

res: float;

task type solver is
entry ranges(l,r: float);
end solver;

solvers:array(0..nprog-1) of solver;

task master is
entry interval(l_r,r_r: float);
entry answer(l_range,r_range:float);
entry result(res:out float);
end master;

task body solver is
-- x,x1,x2,x3,x4,a:float;
RR,LL : float;

function f(x:float) return float is
begin
return (2.0*x-15.0);
end f;

function g(x:float) return float is
begin
return (2.0*x-110.0);
end g;

begin
loop
select
accept ranges (l,r:float) do
LL:=l;
RR:=r;
end ranges;
```

```
put("LL= "); put(LL,10,9);new_line;
put("RR= "); put(RR,10,9);new_line;
if f(LL)=0.0 then master.answer(LL,LL); end if;
if (f(LL)<0.0) XOR (f(RR)<0.0) then
master.answer(LL,RR); end if;

OR
terminate;
end select;
end loop;
end solver;

task body master is
left_r,rigth_r:float;
d,eps:float;
k:integer;

function f(x:float) return float is
begin
return (2.0*x-15.0);
end f;

begin
accept interval (l_r,r_r:float)  do
left_r:=l_r;
rigth_r:=r_r;
end interval;
put("!-left_r:");put(left_r);
put("!-rigth_r");put(rigth_r);
while (abs(rigth_r-left_r)>epsilon) and (abs(f(rigth_r))>epsilon)
loop
d:=abs(rigth_r-left_r)/float(nprog);
for k in 0..nprog-1 loop
solvers(k).ranges(left_r+d*float(k),left_r+d*float(k+1));
end loop;

accept answer(l_range,r_range:float) do
left_r:=l_range;
rigth_r:=r_range;
end answer;

put(";-=-;");new_line;
put("left_r:");put(left_r,10,9);
put("rigth_r");put(rigth_r,10,9);
put("_!_");new_line;
end loop; -- while
```

```
put("!!!!");
accept result(res: out float) do
res:=left_r;
end result;
end master;

begin
--put("-! start");new_line;
master.interval(l,r);
--put("-! interval done");new_line;
master.result(res);
--put("-! exit");new_line;
return res;
end dechotomy;
begin
z:=dechotomy (0.0,10.0,0.00001);
put("Answer:");
put(z);
end dechot1;
```

Рабочие задачи представлены в программе массивом Solvers задач Solver. Поскольку число итераций заранее не известно, прием отрезков задачами Solver осуществляется в цикле с использованием оператора Select. Выход из цикла произойдет по срабатыванию альтернативы terminate, когда задача Master найдет решение, выйдет из цикла, и перестанет обращаться к рабочим задачам.

Задача Master работает в цикле, проверяя условие достижения заданной точности или появления корня на правой границе интервала. Она определяет длину отрезка d и передает начала и концы отрезков всем задачам Solver, затем принимает по входу answer величины начала и конца отрезка, содержащего корень, от одной из задач Solver. После выхода из цикла она возвращает результат по входу result функции dechotomy.

Задача 2. Построить параллельный алгоритм и написать параллельную программу для вычисления с заданной точностью интеграла функции f(x) на интервале

[a,b], используя метод Ньютона-Котеса третьего порядка (метод Симпсона).

Приближенное интегрирование относится к задачам с так называемым геометрическим параллелизмом. Интервал интегрирования можно разбить на произвольное число отрезков и выполнить интегрирование функции на этих отрезках с помощью отдельных процессов, результаты работы которых, потом сложить.

Декомпозиция. Пусть имеется 3P процессов. Разделим [a,b] на P отрезков, с каждым из которых свяжем 3 процесса. Составная квадратурная формула для метода Симпсона имеет следующий вид :

где h - величина шага, а fi значение функции в i-ой точке отрезка. Из формулы видно, что за исключением крайних точек все нечетные компоненты имеют одинаковые коэффициенты, равные 4, а все четные компоненты имеют коэффициенты, равные 2.

Пусть дин процесс вычисляет сумму всех нечетных компонент S1, а второй процесс – сумму всех четных компонент S2 и оба передают значения своих сумм третьему процессу, который вычисляет выражение S=h(4S1+2 S2+f(a)+f(b))/3. Полученная величина является приближенным значением интеграла, если две соседние итерации отличаются больше, чем на заданную величину (eps), то величина шага уменьшается вдвое и начинается новая итерация. Такие вычисления продолжаются до тех пор, пока не будет достигнута заданная точность.

Синхронизация процессов заключается в том, чтобы накопленные суммы передавались управляющему (третьему) процессу, когда суммирование компонент полностью завершено.

Такой подход существенно снижает количество операций умножения, что увеличивает точность и снижает объем вычислений.

Ниже приведен текст программы, которая реализует приближенное интегрирование функции, но только для одного отрезка.

```
with ADA.text_io,ada.integer_text_io,ada.float_text_io;
use ADA.text_io,ada.integer_text_io,ada.float_text_io;
with ada.Numerics.Elementary_Functions;
use  ada.Numerics.Elementary_Functions;
with ada.Numerics; use ada.Numerics;

procedure Sim is

a, b, eps,Tmp: float;
Sum,k,h : float;
Mas : array (1..2) of float := (0.0,0.0);

function f (x: float) return float is -- test function
begin return 1.0/(1.0+x*x);
end f;

task type one_proc is
entry Set (i: integer);
entry  Start_p;
entry  End_p(MAS : out float);
end one_proc;

Event : array (1..2) of one_proc;

task body one_proc is
j : integer;
x, s : float;
begin --one_proc
accept Set (i: integer) do
j :=i;
```

```
end Set;
new_line; put(“ Starting  “);  put(j);
loop
select
accept Start_p;
x := a + h*float(j);  --Start point
s :=0.0;
while x<b  loop
s := s + f(x);
x :=x + 2.0*h;
end loop;
or
accept End_p(Mas : out float) do
Mas := s;
end End_p;
or
terminate
end select;
end loop;
end one_proc;


begin  --Main
put(“Simpson integral “);  new_line;
put(“First value “);  Get(a);  new_line;
put(“Second value “);  Get(b); new_line;
put(“Epsilon “);  Get(Eps);  new_line;
for i in 1..2 loop
Event(i).Set(i);
end loop;
k:= 1.0;  Tmp :=10.0; Sum :=0.0;
while ( abs(Tmp-Sum) > Eps ) loop
h :=( b-a)/(2.0* k);  k:= k + 1.0;
Tmp := Sum;
for i in 1..2 loop
Event(i).Start_p;
end loop;
for i in 1..2 loop
Event(i).End_p(Mas(i));
end loop;
Sum := (h/3.0)* (f(a) +f(b) + 4.0*Mas(1) +2.0*Mas(2));
end loop;
new_line;
put(“Result : “);
put(Sum,5,6 );
end Sim;
```

Автор(ы):

Черняев Валентин Викторович, к.т.н., доцент