МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«БРАТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра математики и физики

УТВЕРЖДАЮ:
Проректор по учебной работе
_____Е.И.Луковникова
«_____» декабря 2018 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ

Б1.В.08

НАПРАВЛЕНИЕ ПОДГОТОВКИ

01.03.02 Прикладная математика и информатика

ПРОФИЛЬ ПОДГОТОВКИ
Инженерия программного обеспечения

Программа академического бакалавриата

Квалификация (степень) выпускника: бакалавр

СОДЕРЖАНИЕ ПРОГРАММЫ **Стр.**

1. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ, СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Вид деятельности выпускника

Дисциплина охватывает круг вопросов, относящихся к организационно-управленческому виду профессиональной деятельности выпускника в соответствии с компетенциями и видами деятельности, указанными в учебном плане.

Цель дисциплины

Целью изучения дисциплины является: ознакомление обучающихся с различными методами, приемами разработки параллельных алгоритмов, проектированием параллельных вычислительных систем.

Задачи дисциплины

- обучение методам компьютерного формализованного представления знаний и реализации выводов для последующей выработки и принятия человеком вариантов принимаемого решения;
- формирование умения и навыков самостоятельного исследования и решения различного рода задач путем применения средств функционального программирования совместно с другими видами программного обеспечения;
- формирование и развитие умений и навыков, позволяющих применять современные математические методы и программное обеспечение для решения задач науки и техники.

Код компетенции	Содержание компетенций	Перечень планируемых результатов обучения по дисциплине
1	2	3
ОПК-3	Способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям	**знать:** - основные параллельные алгоритмы **уметь:** – разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; **владеть:** - приемами построения алгоритмических и программных решений в области параллельного программирования

ОПК-4	Способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности	**знать:** – основы информационно-коммуникационных технологий и информационной безопасности; **уметь:** - использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности **владеть:** – навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования.
ПК-9	Способность составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	**знать:** - принципы планирования работы по осуществлению разработки параллельных программ ; **уметь:** - планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; **владеть:** – навыками планирования, проектирования и оценки параллельных вычислительных систем.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Дисциплина Б1.В.08 Параллельное программирование относится к вариативной части.

Дисциплина Параллельное программирование базируется на знаниях, полученных при изучении таких учебных дисциплин, как: Языки и методы программирования, Теория алгоритмов.

Основываясь на изучении перечисленных дисциплин, Параллельное программирование представляет основу для преддипломной практики и подготовки к государственной итоговой аттестации.

Такое системное междисциплинарное изучение направлено на достижение требуемого ФГОС уровня подготовки по квалификации бакалавр.

3. РАСПРЕДЕЛЕНИЕ ОБЪЕМА ДИСЦИПЛИНЫ

3.1. Распределение объема дисциплины по формам обучения

Форма обучения	*Курс*	*Семестр*	*Трудоемкость дисциплины в часах*						*Контрольная работа*	*Вид промежуточной аттестации*
			Всего часов	*Аудиторных часов*	*Лекции*	*Лабораторные работы*	*Семинары Практические занятия*	*Самостоятельная работа*		
1	2	3	4	5	6	7	8	9	10	11
Очная	4	8	108	48	24	24	-	60	кр	Зачет
Заочная	-	-	-	-	-	-	-	-	-	-
Заочная (ускоренное обучение)	-	-	-	-	-	-	-	-	-	-
Очно-заочная	-	-	-	-	-	-	-	-	-	-

3.2. Распределение объема дисциплины по видам учебных занятий и трудоемкости

Вид учебных занятий	*Трудоемкость (час.)*	*в т.ч. в интерактивной, активной, инновационной формах, (час.)*	*Распределение по семестрам, час*
			8
1	2	3	4
I. Контактная работа обучающихся с преподавателем (всего)	48	6	48
Лекции (Лк)	24	6	24
Лабораторные работы (ЛР)	24	-	24
Контрольная работа*	+	-	+
Групповые (индивидуальные) консультации*	+	-	+
II. Самостоятельная работа обучающихся (СР)	60	-	60
Подготовка к лабораторным работам	20	-	20
Подготовка к зачету	20	-	20
Выполнение контрольной работы	20	-	20
III. Промежуточная аттестация зачет	+	-	+
Общая трудоемкость дисциплины час.	108	-	108
зач. ед.	3	-	3,0

4. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

4.1. Распределение разделов дисциплины по видам учебных занятий

- для очной формы обучения:

№ раз-дела и темы	*Наименование раздела и тема дисциплины*	*Трудоем-кость, (час.)*	*Виды учебных занятий, включая самостоятельную работу обучающихся и трудоемкость; (час.)*		
			учебные занятия		*самостоя тельная работа обучаю-щихся**
			лекции	*лабораторн ые работы*	
1	2	3	4	5	6
1.	**Основные компоненты параллельной вычислительной системы**	**48**	**12**	**8**	**28**
1.1.	Архитектура вычислительных систем	8	2	-	6
1.2.	Модели вычислительных процессов и систем	10	2	2	6
1.3.	Оценки производительности	14	4	2	8
1.4.	Построение и анализ информационного графа	16	4	4	8
2.	**Параллельные алгоритмы распространенных задач**	**60**	**12**	**16**	**32**
2.1	Обработка линейной последовательности	16	4	4	8
2.2	Умножение матрицы на вектор	14	2	4	8
2.3.	Интегрирование	16	4	4	8
2.4.	Сортировка	14	2	4	8
	ИТОГО	**108**	**24**	**24**	**60**

4.2. Содержание дисциплины, структурированное по разделам и темам

№ раздела и темы	*Наименование раздела и темы дисциплины*	*Содержание лекционных занятий*	*Вид занятия в интерактивной, активной, инновационной формах, (час.)*
1	**2**	**3**	**4**
1.	**Основные компоненты параллельной вычислительной системы**		
1.1.	Архитектура оп вычислительных систем	Основные понятия параллельного программирования. Классификация вычислительных систем. Обзор технологий параллельных вычислений.	-

1.2.	Модели вычислительных процессов и систем	Граф алгоритма «операции -операнды». Представление алгоритма в виде диаграммы расписания. Проблема отображения. Определение времени выполнения параллельного алгоритма.	Проблемная лекция (2 час)
1.3.	Оценки производительности	Модели параллельных вычислений . Построение соотношений для оценки производительности. Закон Густавсона – Барсиса. Производительность конвейерных систем. Масштабируемость параллельных вычислений.	-
1.4.	Построение и анализ информационного графа	Классификация алгоритмов по типу параллелизма. Декомпозиция по задачам. Умножение Карацубы. Декомпозиция в задачах с параллелизмом по данным. Декомпозиция по данным. Блочная декомпозиция с учетом локализации подобластей	Проблемная лекция (2 час)
2.	**Параллельные алгоритмы распространенных задач**		
2.1	Обработка линейной последовательности	Вычисление частных сумм последовательности числовых значений. Последовательный алгоритм суммирования. Каскадная схема суммирования. Модифицированная каскадная схема. Вычисление всех частных сумм.	-
2.2	Умножение матрицы на вектор	Блочное разбиение матрицы. Постановка задачи. Последовательный алгоритм. Умножение матрицы на вектор при разделении данных по строкам. Умножение матрицы на вектор при разделении данных по столбцам. Умножение матрицы на вектор при блочном разделении данных.	Проблемная лекция (2 час)
2.3.	Интегрирование	Конкуренция процессов за ресурсы. Критические области. Последовательный алгоритм интегрирования. Метод трапеций. Метод двойного пересчета. Адаптивный алгоритм.	-
2.4.	Сортировка	Алгоритм четной-нечетной перестановки. Параллельная реализация алгоритма. Анализ эффективности. Быстрая сортировка. Сортировка Шелла.	-

4.3. Лабораторные работы

№ п/п	*Номер раздела дисциплины*	*Наименование лабораторной работы*	*Объем (час.)*	*Вид занятия в интерактивной, активной, инновационной формах, (час.)*
1	**1.**	Обмен данными в MPI между двумя процессами	4	-
2		Коллективные операции передачи данных. Редукция	4	-
3	**2.**	Коммуникаторы и группы	4	-
4		Топологии сетей передачи данных	4	-
5		Синхронизация обмена данными	4	-
6		Сортировка последовательности	4	-
		ИТОГО	**24**	**-**

4.4. Семинары/ практические занятия
учебным планом не предусмотрено.

4.5. Контрольные мероприятия: контрольная работа
Контрольная работа «Параллельные алгоритмы»

Цель: Проверка теоретических знаний основных параллельных алгоритмов, умений использовать технологии параллельного программирования для решения задач.

Структура:
Задача 1. Законы Амдала;
Задача 2. Последовательная программа
Задача 3. Параллельная программа.
Задача 4. Анализ параллельной программы
Рекомендуемый объем: 4 задания.

Выдача задания , прием кр проводится в соответствии с календарным учебным графиком.

Оценка	**Критерии оценки контрольной работы**
отлично	Задания выполнены в срок и в полном объеме. Учащийся обнаруживает систематическое и глубокое знание учебного материала, свободно выполняет предусмотренные программой задания, владеет математической терминологией и символикой, может пояснить решения любой задачи.
хорошо	Задания выполнены в срок и в полном объеме. При этом ответ имеет один из недочетов: 1) в изложении решения допущены небольшие пробелы, не исказившие математическое содержание ответа; 2) допущены 1-2 недочета при решении задач, исправленные после замечания преподавателя..
удовлетвори тельно	Задания выполнены не полностью, но не менее, чем на 50%. 1) изложение решения содержит 1 грубую ошибку, искажающую

	математическое содержание ответа 2) в работе допущены недочеты, которые обучающийся не смог исправить после замечания преподавателя.
неудовлетворительно	Выполнено менее 50% работы, работа содержит более 2 грубых ошибок. Учащийся обнаруживает существенные пробелы в знаниях, препятствующие дальнейшему обучению..

5. МАТРИЦА СООТНЕСЕНИЯ РАЗДЕЛОВ УЧЕБНОЙ ДИСЦИПЛИНЫ К ФОРМИРУЕМЫМ В НИХ КОМПЕТЕНЦИЯМ И ОЦЕНКЕ РЕЗУЛЬТАТОВ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Компетенции / *№, наименование разделов дисциплины*	*Кол-во часов*	*Компетенции*			*Σ комп.*	$t_{ср}$, *час*	*Вид учебных занятий*	*Оценка результатов*
		ОПК		*ПК*				
		3	*4*	*9*				
1	2	3	4	5	6	7	8	9
1. Основные компоненты параллельной вычислительной системы	60	+	+	-	2	30	Лк, ЛР	кр, зачет
2. Параллельные алгоритмы распространенных задач	48	-	+	+	2	24	Лк, ЛР	кр, зачет
всего часов	**108**	**30**	**54**	**24**	**3**	**36**		

6. ПЕРЕЧЕНЬ УЧЕБНО-МЕТОДИЧЕСКОГО ОБЕСПЕЧЕНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

1. Шичкина, Ю. А. Организация вычислений на распределенных системах : монография / Ю. А. Шичкина. - Братск : БрГУ, 2013. - 176 с.

7. ПЕРЕЧЕНЬ ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ ЛИТЕРАТУРЫ, НЕОБХОДИМОЙ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

№	*Наименование издания*	*Вид занятия*	*Количество экземпляров в библиотеке*, шт.	*Обеспеченность*, (*экз./ чел.*)
1	2	3	4	5
	Основная литература			
1.	Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .	Лк, ЛР, СР	123 включая аналоги	1,0
2.	Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.	Лк, ЛР, кр, СР	10	0,5
	Дополнительная литература			
3.	Волкова, Т.И. Введение в программирование : учебное пособие / Т.И. Волкова. - Москва ; Берлин : Директ-Медиа, 2018. - 139 с. : ил., схем., табл. - Библиогр. в кн. - ISBN 978-5-4475-9723-8 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=493677	СР, Лк	ЭР	1,0
4.	Гагарина, Л. Г. Технология разработки программного обеспечения : учеб. пособие для вузов / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. - Москва : ИНФРА-М, 2009. - 400 с.	кр, СР	15	1,0

8. ПЕРЕЧЕНЬ РЕСУРСОВ ИНФОРМАЦИОННО-ТЕЛЕКОММУНИКАЦИОННОЙ СЕТИ «ИНТЕРНЕТ» НЕОБХОДИМЫХ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

1.Электронный каталог библиотеки БрГУ
http://irbis.brstu.ru/CGI/irbis64r_15/cgiirbis_64.exe?LNG=&C21COM=F&I21DBN=BOOK&P21DBN=BOOK&S21CNR=&Z21ID=.
2. Электронная библиотека БрГУ
http://ecat.brstu.ru/catalog .
3. Электронно-библиотечная система «Университетская библиотека online»
http://biblioclub.ru .
4. Электронно-библиотечная система «Издательство «Лань»
http://e.lanbook.com .
5. Информационная система "Единое окно доступа к образовательным ресурсам"
http://window.edu.ru .
6. Научная электронная библиотека eLIBRARY.RU http://elibrary.ru .
7. Университетская информационная система РОССИЯ (УИС РОССИЯ)
https://uisrussia.msu.ru/ .
8. Национальная электронная библиотека НЭБ
http://xn--90ax2c.xn--p1ai/how-to-search /.

Специальные тематические сайты

1. Электронный журнал “Типичный програмист” https://tproger.ru .
2. Сайт по программированию https://professorweb.ru
3. Сайт по программированию https://metanit.com

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Обучающийся должен разработать собственный режим равномерного освоения дисциплины. Подготовка студента к предстоящей лекции включает в себя ряд важных познавательно-практических этапов:

– чтение записей, сделанных в процессе слушания и конспектирования предыдущей лекции, вынесение на поля всего, что требуется при дальнейшей работе с конспектом и учебником;

– техническое оформление записей (подчеркивание, выделение главного, выводов, доказательств);

– выполнение практических заданий преподавателя;

– знакомство с материалом предстоящей лекции по учебнику и дополнительной литературе.

Успешность выполнения лабораторных работ определяется подготовкой к ним. Подготовка к лабораторным работам содержит:

- изучение теоретического материала, содержащегося в учебной литературе, изучение лекционного материала,

- знакомство с заданиями на лабораторную работу;

- составление плана выполнения лабораторной работы.

Наиболее продуктивной является самостоятельная работа в библиотеке, где доступны основные и дополнительные печатные и электронные источники.

При выполнении приведенных выше рекомендаций подготовка к экзамену сведется к повторению изученного и совершенствованию навыков применения теоретических положений и различных методов решения к стандартным и нестандартным заданиям.

9.1. Методические указания для обучающихся по выполнению лабораторных работ

Лабораторная работа №1

Обмен данными в MPI между двумя процессами

Цель: Изучить парные механизмы передачи данных в технологии MPI

Теоретические сведения

Все операции приема и передачи сообщение делятся на парные (точка-точка) и коллективные. Парные операции осуществляются между двумя процессами, коллективные позволяют осуществлять взаимодействие группы процессов. Для выполнения парных операций могут использоваться разные режимы передачи, среди которых синхронный, блокирующий и др.

1. Типы данных MPI

При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях MPI необходимо указывать тип пересылаемых данных.

Таблица 1. Базовые типы данных MPI для алгоритмического языка С

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float

MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_PACKED для пакетной пересылки данных	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

2. Передача сообщений (точка-точка)

Для передачи сообщения процесс-отправитель должен выполнить функцию:

int MPI_Send(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm),

где

- buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения,
- count – количество элементов данных в сообщении,
- type - тип элементов данных пересылаемого сообщения,
- dest - ранг процесса, которому отправляется сообщение,
- tag - значение-тег, используемое для идентификации сообщений,
- comm - коммуникатор, в рамках которого выполняется передача данных.

Следует отметить:

1. Отправляемое сообщение определяется через указание блока памяти (буфера), в котором это сообщение располагается. Используемая для указания буфера триада

(buf, count, type)

входит в состав параметров практически всех функций передачи данных,

2. Процессы, между которыми выполняется передача данных, в обязательном порядке должны принадлежать коммуникатору, указываемому в функции MPI_Send,

3. Параметр tag используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число (см. также описание функции MPI_Recv).

Сразу же после завершения функции MPI_Send процесс-отправитель может начать повторно использовать буфер памяти, в котором располагалось отправляемое сообщение. Вместе с этим, следует понимать, что в момент завершения функции MPI_Send состояние самого пересылаемого сообщения может быть совершенно различным - сообщение может располагаться в процессе-отправителе, может находиться в процессе передачи, может храниться в процессе-получателе или же может быть принято процессом-получателем при помощи функции MPI_Recv. Тем самым, завершение функции MPI_Send означает лишь, что операция передачи начала выполняться и пересылка сообщения будет рано или поздно будет выполнена.

Прием сообщений (точка-точка)

Для приема сообщения процесс-получатель должен выполнить функцию:

int MPI_Recv(void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm,

MPI_Status *status),

где

- buf, count, type – буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в MPI_Send,
- source - ранг процесса, от которого должен быть выполнен прием сообщения,
- tag - тег сообщения, которое должно быть принято для процесса,
- comm - коммуникатор, в рамках которого выполняется передача данных,
- status – указатель на структуру данных с информацией о результате выполнения операции приема данных.

Следует отметить:

1. Буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения,

2. При необходимости приема сообщения от любого процесса-отправителя для параметра source может быть указано значение MPI_ANY_SOURCE,

3. При необходимости приема сообщения с любым тегом для параметра tag может быть указано значение MPI_ANY_TAG,

4. Параметр status позволяет определить ряд характеристик принятого сообщения:

- status.MPI_SOURCE – ранг процесса-отправителя принятого сообщения,
- status.MPI_TAG - тег принятого сообщения.

Функция

MPI_Get_count (MPI_Status *status, MPI_Datatype type, int *count)

возвращает в переменной count количество элементов типа type в принятом сообщении.

Вызов функции MPI_Recv не должен согласовываться со временем вызова соответствующей функции передачи сообщения MPI_Send – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.

По завершении функции MPI_Recv в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция MPI_Recv является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то причинам ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.

4. Тупиковые ситуации (deadlock)

При обмене данными в некоторых случаях возможны вызванные взаимной блокировкой т.н. тупиковые ситуации (используются также термины 'deadlock', 'клинч"); в этом случае функции посылки и приема данных мешают друг другу и обмен не может состояться. Ниже рассмотрена deadlock-ситуация при использовании для пересылок разделяемой памяти.

Вариант 1.	Ветвь 1:	Ветвь 2:
	Recv (из ветви 2)	Recv (из ветви 1)
	Send (в ветвь 2)	Send (в ветвь 1)

Вариант 1 приведет к deadlock’у при любом используемом инструментарии, т.к. функция приема не вернет управления до тех пор, пока не получит данные; из-за этого функция передачи не может приступить к отправке данных, поэтому функция приема не вернет управление... и т.д. до бесконечности.

Вариант 2.	Ветвь 1:	Ветвь 2:
	Send (в ветвь 2)	Send (в ветвь 1)
	Recv (из ветви 2)	Recv (из ветви 1)

Казалось бы, что (если функция передачи возвращает управление только после того, как данные попали в пользовательский буфер на стороне приема) и здесь deadlock неизбежен. Однако при использовании MPI зависания во втором варианте не происходит: функция MPI_Send, если на приемной стороне нет готовности (не вызвана MPI_Recv), не станет дожидаться ее вызова, а скопирует данные во временный буфер и немедленно вернет управление основной программе. Вызванный далее MPI_Recv данные получит не напрямую из пользовательского буфера, а из промежуточного системного буфера (т.о. используемый в MPI способ буферизации повышает надежность – делает программу более устойчивой к возможным ошибкам программиста). Т.о. наряду с полезными качествами (см. выше) свойство блокировки может служить причиной возникновения (трудно локализуемых и

избегаемых для непрофессионалов) тупиковых ситуаций при обмене сообщениями между процессами.

Рассмотрим различные способы разрешения тупиковых ситуаций.

1. Простейшим вариантом разрешения тупиковой ситуации будет изменение порядка следования процедур посылки и приема сообщения на оном из процессов, как показано ниже.

Вариант 3.	Ветвь 1:	Ветвь 2:
	Send (в ветвь 2)	Recv (из ветви 1)
	Recv (из ветви 2)	Send (в ветвь 1)

2. Другим вариантом разрешения тупиковой ситуации может быть использование неблокирующих операций. Заменим вызов процедуры приема сообщения с блокировкой на вызов процедуры MPI_Irecv. Расположим его перед вызовом процедуры MPI_Send, т.е. преобразуем фрагмент следующим образом.

Вариант 4.	Ветвь 1:	Ветвь 2:
	Send (в ветвь 2)	IRecv (из ветви 1)
	Recv (из ветви 2)	Send (в ветвь 1) MPI_Wait

В такой ситуации тупик гарантированно не возникнет, поскольку к моменту вызова процедуры MPI_Send запрос на прием сообщения уже будет выставлен, а значит, передача данных может начаться. При этом рекомендуется выставлять процедуру MPI_Irecv в программе как можно раньше, чтобы раньше предоставить возможность начала пересылки и максимально использовать преимущества асинхронности.

Находим максимум wi.

Пример 1. Передача и прием сообщений между двумя процессами

//В этой программе процесс 0 пересылает процессу 1 переменную b, и обратно получает переменную a

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv){
int rank;
float a, b;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
a = 0.0;
b = 0.0;
  if(rank = = 0){     адрес b
b = 1.81;
MPI_Send(&b, 1, MPI_FLOAT, 1, 0, MPI_COMM_WORLD);
MPI_Recv(&a, 1, MPI_FLOAT, 1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
}
if(rank = = 1){
a = 2.23;
MPI_Recv(&b, 1, MPI_FLOAT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
MPI_Send(&a, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD);
}
printf("process %d a = %f, b = %f\n", rank, a, b);
MPI_Finalize();
}
```

Пример 2. Передача элементов массива

Предположим, нам нужно сформировать и обработать массив buf из 20 случайных чисел. Если формировать этот массив вне MPI_Init, то каждый из распараллеленных процессов сформирует свой массив. Поэтому мы прописываем формирование массива в процессе 0, а затем рассылаем его другим процессам.

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char* argv[]){
const int n=20;     // размер массива
int buf [n];          //массив символов
int proc_num, proc_rank, recv_rank, i, x=0;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, & proc_num);
MPI_Comm_rank(MPI_COMM_WORLD, & proc_rank);
if ( proc_rank = = 0 ){
// Действия, выполняемые только процессом с рангом 0
// формирование массива
for (i=0; i<n; i++) {buf[i]= rand()%100; }
// пересылка массива всем процессам, кроме процесса с рангом 0
for ( int i=1; i< proc_num; i++ ) {
MPI_Send (buf, 20, MPI_INT, i, 0, MPI_COMM_WORLD); // buf –  это уже адрес
}
}
else // Сообщение, принимаемое всеми процессами от процесса с рангом 0
MPI_Recv(buf, 20, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
MPI_Finalize();
return 0;
}
```

Задания к лабораторной работе

Задание 1. Написать программу для трех процессов, выполняющих создание и передачу данных

Вариант	Процесс 0	Процесс1	Процесс 2
1	a=3 передача a процессу 2 прием c из проц 2 печать c	b=5 передача b на процесс 2	прием a и b печать a и b c=a+b передача на процесс 0
2	a=2 передача a процессу 1 прием c из проц 2 печать c	b=5 передача b на процесс 2 прием a a=a+b передача процессу 2	прием a и b печать a и b c=a*b передача на процесс 0
3	a=3 передача a процессу 2 прием b из проц 1 прием a из проц 2 b=a*b печать b	b=5 передача b на процесс 0 печать b	прием a из процесса 0 a++ передача a на проц 0 печать a
4	a=2 передача a процессам 1	прием a из проц 0 печать a	прием a из процесса 1 печать a

	и 2 прием из проц 2 и 1 печатьb и c	b=2a-1 передача b на процесс 0	c=a*a передача c процессу 0

Задание 2. Написать параллельную программу для трех процессов, которые выполняют операции с векторами. Постройте граф алгоритма.

Вариант 1.Программа , вычисляющая угол между векторами.

Процессы 1 и 2 генерируют координаты векторов a и b в трехмерном пространстве и пересылают их процессу 0. Затем они же вычисляют модули этих векторов и снова высылают процессу 0

Процесс 0 принимает векторы a и b, вычисляет их скалярное произведение, принимает модули и находит косинус угла и угол между векторами.

Каждый процесс выводит на экран результаты своих расчетов.

Вариант 2. Программа, вычисляющая площадь параллелограмма, построенного на векторах a и b.

Процесс 0 запрашивает у пользователя координаты двух векторов и сообщает их процессам 1 и 2. Далее, вычисляются координаты векторного произведения:

$\begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$ разложением по первой строке. (Нам нужны только миноры!).

Процесс 0 вычисляет минор M11=$\begin{vmatrix} a_y & a_z \\ b_y & b_z \end{vmatrix}$,

процесс 1 вычисляет минор M12, процесс 2 – M13 Формулы для миноров найдите сами

Каждый процесс вычисляет квадрат полученного минора и все процессы пересылают их нулевому.

Нулевой процесс находит площадь параллелограмма по полученным данным.

Вариант 3. Программа, проверяющая компланарность векторов a, b и c .

Процесс 0 запрашивает у пользователя координаты 3-х векторов a, b, c и сообщает их процессам 1 и 2. Далее, вычисляется смешанное произведение:

$\begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$ разложением по первому ряду.

Для этого процесс 0 вычисляет алгебраическое дополнениеA11=$(-1)^2\begin{vmatrix} b_y & b_z \\ c_y & c_z \end{vmatrix}$,

процесс 1 вычисляет A12, процесс 2 – A13 Формулы для алгебраических дополнений найдите сами. Каждый процесс умножает полученное алгебраическое дополнение на свой элемент и далее пересылает результат процессу 0.

Процесс 0 принимает данные, находит их сумму и делает выводы о компланарности векторов.

Вариант 4. Программа выполняет линейное преобразование, заданное матрицей F[3][3] над вектором x[3].

Процесс 0 генерирует матрицу размера 3x3 и вектор. Пересылает вектор процессам 1 и 2, Затем пересылает процессу 1 вторую , а процессу 2 – третью строку матрицы.

Каждый процесс умножает свою строку на вектор и отсылает результат нулевому.

Нулевой процесс печатает результат.

Порядок выполнения:

1. Изучить теоретические сведения.
2. Изучить примеры программ, приведенных в теоретической части.
3. Модифицировать пример 1.1 или 1.2 для выполнения задания 1 согласно Вашему варианту.
4. Оформить отчет.

Форма отчетности:
Отчет по работе содержит:
1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе

При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .
2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

Контрольные вопросы для самопроверки
1. Для чего нужны аргументы программы?
2. Какую смысловую нагрузку несет аргумент argc?
3. Что содержит argv[0]?
4. Можно ли в вызове функции main указать аргументы, отличные от int argc, char**argv и char**env?
5. Для чего используются опции программы?
6. Какие значении может возвращать функция getopt()?
7. Что означает строка "aht:s:", указанная в качестве третьего аргумента функции getopt()?
8. Какое значение сохранится в переменой optind при вызове программы с аргументами -t 23 -s qwerty -a?

Лабораторная работа 2.

Коллективные операции передачи данных. Редукция

Цель: Изучить коллективные операции передачи данных

Теоретические сведения

1. Широковещательная рассылка данных

Как уже отмечалось ранее, под коллективными операциями в MPI понимаются операции над данными, в которых принимают участие все процессы используемого коммуникатора. Достижение эффективного выполнения операции передачи данных от одного процесса всем процессам программы (широковещательная рассылка данных) может быть обеспечено при помощи функции:

int MPI_Bcast(*buf,count, type, rank, comm),

где

- buf, count, type – буфер памяти с отправляемым сообщением (для процесса с рангом 0), и для приема сообщений для всех остальных процессов,
- rank - ранг процесса, выполняющего рассылку данных,
- comm - коммуникатор, в рамках которого выполняется передача данных.

Следует отметить:

1. Функция MPI_Bcast определяет коллективную операцию и, тем самым, при выполнении необходимых рассылок данных вызов функции MPI_Bcast должен быть осуществлен всеми процессами указываемого коммуникатора (см. далее пример программы),

2. Указываемый в функции MPI_Bcast буфер памяти имеет различное назначение в разных процессах.

Для процесса с рангом root, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение. Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

```
// рассылка данных на все процессы
MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

2 Передача данных от всех процессов одному процессу. Операции редукции

В программе суммирования числовых значений имеющаяся процедура сбора и последующего суммирования данных является примером часто выполняемой коллективной операции передачи данных от всех процессов одному процессу. В этой операции над собираемыми значениями осуществляется та или иная обработка данных (для подчеркивания последнего момента данная операция еще именуется операцией редукции данных):

int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm),

- sendbuf - буфер памяти с отправляемым сообщением,
- recvbuf – буфер памяти для результирующего сообщения (только для процесса с рангом root),
- count - количество элементов в сообщениях,
- type – тип элементов сообщений,
- op - операция, которая должна быть выполнена над данными,
- root - ранг процесса, на котором должен быть получен результат,
- comm - коммуникатор, в рамках которого выполняется операция.

В качестве операций редукции данных могут быть использованы предопределенные в MPI операции

Таблица 1. Базовые типы операций MPI для функций редукции данных

Операция	Описание
MPI_MAX	Определение максимального значения
MPI_MIN	Определение минимального значения
MPI_SUM	Определение суммы значений
MPI_PROD	Определение произведения значений
MPI_LAND	Выполнение логической операции "И" над значениями сообщений
MPI_BAND	Выполнение битовой операции "И" над значениями сообщений
MPI_LOR	Выполнение логической операции "ИЛИ" над значениями сообщений
MPI_BOR	Выполнение битовой операции "ИЛИ" над значениями сообщений
MPI_LXOR	Выполнение логической операции исключающего "ИЛИ" над значениями сообщений
MPI_BXOR	Выполнение битовой операции исключающего "ИЛИ" над значениями сообщений

Следует отметить:

1. Функция MPI_Reduce определяет коллективную операцию и, тем самым, вызов функции должен быть выполнен всеми процессами указываемого коммуникатора, все вызовы функции должны содержать одинаковые значения параметров count, type, op, root, comm,

2. Передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом root,

3. Выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений. Так, например, если сообщения содержат по два элемента данных и выполняется операция суммирования MPI_SUM, то результат также будет состоять из двух значений, первое из которых будет содержать сумму первых элементов всех отправленных сообщений, а второе значение будет равно сумме вторых элементов сообщений соответственно.

Применим полученные знания для переработки ранее рассмотренной программы суммирования – как можно увидеть, весть выделенный программный код может быть теперь заменен на вызов одной лишь функции MPI_Reduce:

```
// сборка частичных сумм на процессе с рангом 0
MPI_Reduce(&proc_sum, &total_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Пример Программа, проверяющая число numb на простоту. Процесс с рангом 0 принимает число от пользователя и высылает (MPI_Bcast) его всем процессам. Каждый процесс, включая и 0-й проверяет наличие делителей числа numb в своем диапазоне [ibeg; iend]. Если делитель найден, процессу с рангом 0 посылается (MPI_Reduce) значение div=1, если нет, то div=0. Итоговый результат compound вычисляется как логическое ИЛИ частичных результатов.

```
#include<stdio.h>
#include <math.h>
#include<mpi.h>

int main(int argc, char **argv)
{
  int size, rank;
  int len,ibeg,iend,i;
  int numb, div=0, compound;
  MPI_Status *status;
  MPI_Request *request;
  MPI_Init(&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

 // Процесс с рангом 0 запрашивает значение у пользователя:
 if (rank==0){
    printf("give me a number: ");
    scanf ("%d",&numb);
  }
// Действия всех процессов с любым рангом:
  MPI_Bcast(&numb,1,MPI_INT,0,MPI_COMM_WORLD);

      len=(numb+1)/size+1;       //   * Вычисляем
      ibeg=rank*len;             //    диапазон
```

```
    if (ibeg<2) ibeg=2;             //   для поиска
    iend=(rank+1)*len;              //   возможных
    if (iend>numb)iend=numb; //   делителей *
   // Осуществляем проверку на делимость:
    div=0;
    for(i=ibeg;i<iend;i++)
        if(numb%i==0) {
          div=1;
          //printf ("proc rank=%d found div=%d \n", rank, i);
          break;
        }
  // В compound собираем результаты:
  MPI_Reduce(&div,&compound,1,MPI_INT,
                  MPI_LOR,0,MPI_COMM_WORLD);
// Печать результата:
if(rank==0){
   if (compound==0) printf("The number is prime\n");
   else printf("The number is compound\n");
 }
  MPI_Finalize();
 return 0;
}
```

3 Синхронизация вычислений

В ряде ситуаций независимо выполняемые в процессах вычисления необходимо синхронизировать.

Так, например, для измерения времени начала работы параллельной программы необходимо, чтобы для всех процессов одновременно были завершены все подготовительные действия, перед окончанием работы программы все процессы должны завершить свои вычисления и т.п.

Синхронизация процессов, т.е. одновременное достижение процессами тех или иных точек процесса вычислений, обеспечивается при помощи функции:

int MPI_Barrier(comm);

Функция MPI_Barrier определяет коллективную операцию и, тем самым, при использовании должна вызываться всеми процессами используемого коммуникатора. При вызове функции MPI_Barrier выполнение процесса блокируется, продолжение вычислений процесса произойдет только после вызова функции MPI_Barrier всеми процессами коммуникатора.

Рассмотрим далее оставшиеся базовые коллективные операции передачи данных.

4. Обобщенная передача данных от одного процесса всем процессам

Обобщенная операция передачи данных от одного процесса всем процессам (распределение данных) отличается от широковещательной рассылки тем, что процесс передает процессам различающиеся данные

Выполнение данной операции может быть обеспечено при помощи функции:

int MPI_Scatter(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm),

- sbuf, scount, stype - параметры передаваемого сообщения (scount определяет количество элементов, передаваемых на каждый процесс)
- rbuf, rcount, rtype - параметры сообщения, принимаемого в процессах,
- root – ранг процесса, выполняющего рассылку данных,
- comm - коммуникатор, в рамках которого выполняется передача данных.

При вызове этой функции процесс с рангом root произведет передачу данных всем другим процессам в коммуникаторе. Каждому процессу будет отправлено scount элементов.

Процесс с рангом 0 получит блок данных из sbuf из элементов с индексами от 0 до scount-1, процессу с рангом 1 будет отправлен блок из элементов с индексами от scount до 2* scount-1 и т.д. Тем самым, общий размер отправляемого сообщения должен быть равен scount * p элементов, где p есть количество процессов в коммуникаторе comm.

Следует отметить, поскольку функция MPI_Scatter определяет коллективную операцию, вызов этой функции при выполнении рассылки данных должен быть обеспечен в каждом процессе коммуникатора.

Отметим также, что функция MPI_Scatter передает всем процессам сообщения одинакового размера.

5. Обобщенная передача данных от всех процессов одному процессу

Операция обобщенной передачи данных от всех процессоров одному процессу (сбор данных) является обратной к процедуре распределения данных. Для выполнения этой операции в MPI предназначена функция:

int MPI_Gather(void *sbuf,int scount,MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm),

- sbuf, scount, stype - параметры передаваемого сообщения,
- rbuf, rcount, rtype - параметры принимаемого сообщения,
- root – ранг процесса, выполняющего сбор данных,
- comm - коммуникатор, в рамках которого выполняется передача данных.

При выполнении функции MPI_Gather каждый процесс в коммуникаторе передает данные из буфера sbuf на процесс с рангом root. Этот "ведущий" процесс осуществляет склейку поступающих данных в буфере rbuf. Склейка данных осуществляется линейно, положение пришедшего фрагмента данных определяется рангом процесса, его приславшего. В целом процедура MPI_Gather обратна по своему действию процедуре MPI_Scatter. Для того, чтобы разместить все поступающие данные, размер буфера rbuf должен быть равен scount * p элементов, где p есть количество процессов в коммуникаторе comm.

Функция MPI_Gather также определяет коллективную операцию, и ее вызов при выполнении сбора данных должен быть обеспечен в каждом процессе коммуникатора.

Следует отметить, что при использовании функции MPI_Gather сборка данных осуществляется только на одном процессе. Для получения всех собираемых данных на каждом из процессов коммуникатора необходимо использовать функцию сбора и рассылки:

int MPI_Allgather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm).

6. Общая передача данных от всех процессов всем процессам

Передача данных от всех процессов всем процессам является наиболее общей операцией передачи данных. Выполнение данной операции может быть обеспечено при помощи функции:

int MPI_Alltoall(void *sbuf,int scount,MPI_Datatype stype, void *rbuf,int rcount,MPI_Datatype rtype,MPI_Comm comm),

- sbuf, scount, stype - параметры передаваемых сообщений,
- rbuf, rcount, rtype - параметры принимаемых сообщений
- comm - коммуникатор, в рамках которого выполняется передача данных.

При выполнении функции MPI_Alltoall каждый процесс в коммуникаторе передает данные из scount элементов каждому процессу (общий размер отправляемых сообщений в процессах должен быть равен scount * p элементов, где p есть количество процессов в коммуникаторе comm) и принимает сообщения от каждого процесса.

Вызов функции MPI_Alltoall при выполнении операции общего обмена данными должен быть выполнен в каждом процессе коммуникатора.

Пример. Параллельная программа суммирования числовых значений элементов массива

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char* argv[]) {

    double massive[100], part_mass[100], total_sum, part_sum = 0.0;
    int rank, size, count, i;
     MPI_Status status;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&size);
     MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    // подготовка данных
    if ( rank == 0 ) {
          for (i=0; i<100; i++)
                massive[i]=2*i-7;
    }
    // рассылка данных на все процессы
    count=100/size;
    MPI_Scatter(massive, count, MPI_DOUBLE, part_massive, count, MPI_DOUBLE, 0,
                MPI_COMM_WORLD);

    // вычисление частичной суммы на каждом из процессов
    i=0;
    while (i<count) {
        part_sum=part_sum+part_mass[i];
       i++;
    }
    // сборка частичных сумм на процессе с рангом 0
    // все процессы отсылают свои частичные суммы
   MPI_Reduce(&proc_sum, &total_sum, 1, MPI_DOUBLE, MPI_SUM, 0,
                MPI_COMM_WORLD);

    // вывод результата
    if ( proc_rank == 0 )
      printf("\nTotal Sum = %10.2f",total_sum);
    MPI_Finalize();
}
```

Задания

1. Дан массив. Разработайте алгоритм и напишите MPI-программу вычисления максимального элемента, используя операцию MPI_Scatter передачи данных от одного процесса всем процессам и операцию редукции.

2. Разработайте алгоритм и напишите MPI-программу вычисления скалярного произведения векторов *a* и b, используя MPI_Bcast и операцию редукции.
3. Разработайте алгоритм и напишите MPI-программу вычисления произведения матрицы A(nxn) на вектор V(nx1)

.

Контрольные вопросы

1. Какие операции называют операциями ввода-вывода?
2. Какие функции называют низкоуровневыми?
3. Почему системные программы используют низкоуровневый ввод-вывод?
4. Каким значениям файлового дескриптора соответствует стандартный поток вывода?

Порядок выполнения работы

1. Изучить теоретические сведения.
2. Изучить примеры программ, приведенных в теоретической части.
3. Выполнить задания.
4. Оформить отчет.

Отчет должен включать:
– титульный лист;
– формулировку цели работы;
– формулировку задания;
– листинги программ;
– результаты выполнения программ;
– выводы, согласованные с целью работы;
– ответы на контрольные вопросы.

Форма отчетности:
Отчет по работе содержит:
1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе
При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .
2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

Лабораторная работа 3

Коммуникаторы и группы

Цель работы: Научиться выделять отдельные процессы в группы, освоить способы взаимодействии процессов в группах.

Коммуникаторы и группы

1 Коммуникаторы

Стандартный коммуникатор MPI_COMM_WORLD создается автоматически при запуске параллельной программы на выполнение. В MPI есть несколько стандартных коммуникаторов:

MPI_COMM_WORLD – включает все процессы параллельной программы (создается автоматически при запуске параллельной программы на выполнение);

MPI_COMM_SELF – включает только данный вызывающий процесс;

MPI_COMM_NULL – пустой коммуникатор, не содержит ни одного процесса.

В MPI имеются процедуры, позволяющие создавать новые коммуникаторы, содержащие подмножества процессов.

2 Управление группами

С понятием коммуникатора тесно связано понятие группы процессов. Под группой понимают упорядоченное множество процессов. Каждому процессу в группе соответствует уникальный номер - ранг. Каждому процессу в группе соответствует уникальный ранг. Группа - отдельное понятие MPI, и операции с группами, могут выполняться отдельно от операций с коммуникаторами, но операции обмена для указания области действия всегда используют коммуникаторы, а не группы. Таким образом, один процесс или группа процессов могут входить в несколько различных коммуникаторов. С группами процессов в MPI допустимы следующие действия:

- Объединение групп;
- Пересечение групп;
- Разность групп.

Новая группа может быть создана только из уже существующих групп. В качестве исходной группы при создании новой может быть использована группа, связанная с предопределенным коммуникатором MPI_COMM_WORLD.

MPI_GROUP_EMPTY – пустая группа, не содержащая ни одного процесса.

MPI_GROUP_NULL – значение, используемое для ошибочной группы.

Для получения группы, связанной с коммуникатором comm, используется функция

int MPI_Comm_group (MPI_Comm comm,MPI_Group *group).

Возвращаемый параметр group - группа, связанная с коммуникатором.

Далее, для конструирования новых групп могут быть применены следующие действия:

1) создание новой группы newgroup из группы oldgroup, которая будет включать в себя n процессов, ранги которых задаются массивом ranks

int MPI_Group_incl(MPI_Group oldgroup, int n, int *ranks, MPI_Group *newgroup);

Ранги процессов содержатся в массиве ranks. При n = 0 создается пустая группа MPI_GROUP_EMPTY. С помощью данной подпрограммы можно не только создать новую группу, но и изменить порядок процессов в старой группе.

2) создание новой группы newgroup из группы oldgroup, которая будет включать в себя n процессов, ранги которых не совпадают с рангами, перечисленными в массиве ranks

int MPI_Group_excl(MPI_Group oldgroup, int n, int *ranks, MPI_Group *newgroup)

При n = 0 новая группа тождественна старой.

3) создание новой группы newgroup как разности групп group1 и group2

int MPI_Group_difference(MPI_Group group1, MPI_Group
group2, MPI_Group *newgroup);

4) создание новой группы newgroup как пересечения group1 и group2

int MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup);

5) создание новой группы newgroup как объединения group1 и group2

int MPI_Group_union(MPI_Group group1, MPI_Group
group2, MPI_Group *newgroup);

6) Создание группы newgroup из группы group добавлением в нее n процессов, ранг которых указан в массиве ranks

int MPI_Group_range_incl(MPI_Group oldgroup, int n, int ranks[][3], MPI_Group *newgroup)

Массив ranks состоит из целочисленных триплетов вида (первый_1, последний_1, шаг_1), …, (первый_n, последний_n, шаг_n). В новую группу войдут процессы с рангами (по первой группе) первый_1, первый_1 + шаг_1, ….

7) Создание группы newgroup из группы group исключением из нее n процессов, ранг которых указан в массиве ranks

int MPI_Group_range_excl(MPI_Group group, int n, int ranks[][3], MPI_Group *newgroup)

Массив ranks устроен так же, как аналогичный массив в подпрограмме MPI_Group_range_incl.

Имеются и другие подпрограммы-конструкторы новых групп.

8) Уничтожение группы group

int MPI_Group_free(MPI_Group *group)

Получить информацию об уже созданной группе можно, используя следующие функции:

Определение количества процессов (size) в группе (group)

int MPI_Group_size(MPI_Group group, int *size)

Определение ранга (rank) процесса в группе group

int MPI_Group_rank(MPI_Group group, int *rank)

Если процесс не входит в указанную группу, возвращается значение MPI_UNDEFINED.

Управление коммуникаторами

В MPI различают два вида коммуникаторов – интракоммуникаторы, используемые для операций внутри одной группы процессов, и интеркоммуникаторы - для обмена между группами процессов. Создание коммуникатора - операция коллективная и должна вызываться всеми процессами коммуникатора).

1) Вызов MPI_Comm_dup копирует уже существующий коммуникатор oldcom

int MPI_Comm_dup (MPI_Comm oldcom,MPI_comm *newcomm),

в результате будет создан новый коммуникатор newcomm, включающий в себя те же процессы.

2) Для создания нового коммуникатора из группы служит функция MPI_Comm_create.

int MPI_comm_create (MPI_Comm oldcom, MPI_Group group, MPI_Comm *newcomm)

Вызов создает новый коммуникатор newcomm, который будет включать в себя процессы группы group коммуникатора oldcomm.

3) Создание нескольких коммуникаторов сразу методом расщепления

int MPI_Comm_split(MPI_Comm oldcomm, int split, int rank, MPI_Comm* newcomm)

Группа процессов, связанных с коммуникатором oldcomm, разбивается на непересекающиеся подгруппы, по одной для каждого значения аргумента split. Процессы с одинаковым значением split образуют новую группу. Ранг в новой группе определяется значением rank. Если процессы A и B вызывают MPI_Comm_split с одинаковым значением split, а аргумент rank, переданный процессом A, меньше, чем аргумент, переданный процессом B, ранг A в группе, соответствующей новому коммуникатору, будет меньше ранга процесса B. Если же в вызовах используется одинаковое значение rank, система присвоит ранги произвольно. Для каждой подгруппы создается собственный коммуникатор newcomm.

MPI_Comm_split должны вызвать все процессы из старого коммуникатора, даже если они не войдут в новый коммуникатор. Для этого в качестве аргумента split в подпрограмму передается предопределенная константа MPI_UNDEFINED. Соответствующие процессы вернут в качестве нового коммуникатора значение MPI_COMM_NULL. Новые коммуникаторы, созданные подпрограммой MPI_Comm_split, не пересекаются, однако с помощью повторных вызовов подпрограммы MPI_Comm_split можно создавать и перекрывающиеся коммуникаторы.

4) Пометить коммуникатор comm для удаления

int MPI_Comm_free(MPI_Comm *comm)

Обмены, связанные с этим коммуникатором, завершаются обычным образом, а сам коммуникатор удаляется только после того, как на него не будет активных ссылок. Данная операция может применяться к коммуникаторам интра- и интер-.

5) Сравнение двух коммуникаторов (comm1) и (comm2)

int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int *result)

Выходной параметр result – целое значение, которое равно MPI_IDENT, если контексты и группы коммуникаторов совпадают; MPI_CONGRUENT, если совпадают только группы; MPI_SIMILAR и MPI_UNEQUAL, если не совпадают ни группы, ни контексты. В качестве аргументов нельзя использовать пустой коммуникатор MPI_COMM_NULL.

6) Получение доступа к удаленной группе, связанной с интеркоммуникатором comm

int MPI_Comm_remote_group(MPI_Comm comm, MPI_Group *group)

Выходной параметр:group – удаленная группа.

7) Создание интеркоммуникатора

int MPI_Intercomm_create(MPI_Comm local_comm, int local_leader, MPI_Comm peer_comm, int remote_leader, int tag, MPI_Comm *new_intercomm)

Входные параметры:

local_comm – локальный интракоммуникатор;

local_leader – ранг лидера в локальном коммуникаторе (обычно 0);

peer_comm – удаленный коммуникатор;

remote_leader – ранг лидера в удаленном коммуникаторе (обычно 0);

tag – тег интеркоммуникатора, используемый лидерами обеих групп для обменов в контексте родительского коммуникатора.

Выходной параметр:

new_intercomm – интеркоммуникатор.

Вызов этой подпрограммы должен выполняться в обеих группах процессов, которые должны быть связаны между собой. В каждом из этих вызовов используется локальный интракоммуникатор, соответствующий данной группе процессов. При работе с MPI_Intercomm_create локальная и удаленная группы процессов не должны пересекаться, иначе возможны «тупики».

```
#include <mpi.h>
#include <stdio.h>

int main(int argc,char* argv[]){
MPI_Group systemgroup;
MPI_Group mygroup;
MPI_Comm newcomm;
int size,rank;
int ranks[2];
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_group(MPI_COMM_WORLD,&systemgroup);
// получили группу процессов, связанную с
// коммуникатором MPI_COMM_WORLD
// в эту группу входят все процессы
ranks[0] = 1;
ranks[1] = 3;
MPI_Group_incl(systemgroup,2,ranks,&mygroup);
//новая группа содержит 2 процесса - 1 и 3
MPI_Comm_create(MPI_COMM_WORLD,mygroup,&newcomm);
//коммуникатор newcomm полностью готов к работе
if (newcomm!= MPI_COMM_NULL)
MPI_Comm_free(&newcomm);

MPI_Group_free(&mygroup);
```

```
MPI_Finalize();
}
```

В программе создается новый коммуникатор, включающий в себя процессы с рангами 1 и 3 из "старого" коммуникатора - MPI_COMM_WORLD. Стоит заметить, что теперь у 3-го процесса два ранга – один глобальный (в рамках коммуникатора MPI_COMM_WORLD), а другой – в рамках только что созданного коммуникатора newcomm.

Зачем нужны созданные пользователем коммуникаторы? Одна причина, побуждающая их использовать, уже называлась - без них не обойтись разработчикам библиотек. Вторая причина, по которой приходится создавать собственные коммуникаторы, состоит в том, что только с их помощью можно управлять более сложными, чем "точка - точка" типами обменов

Вычисление суммы последовательности числовых значений

Простейшим и вместе с тем наиболее широко применяемым в разнообразных задачах является алгоритм вычисления суммы числовой последовательности:

$$S = \sum_{i=1}^{n} x_i \,, \qquad (1)$$

где n – количество суммируемых значений.

Граф-схемы традиционных (линейного и каскадного) алгоритмов решения этой задачи в качестве примеров были приведены в разделе 3 на рис. 3.1 и 3.2 соответственно. Оценим необходимое для реализации этих схем число вычислительных операций.

Очевидно, что для реализации алгоритма последовательного суммирования необходимо $n-1$ операций. Общее количество операций суммирования в каскадной схеме такое же, как в последовательном алгоритме:

$$K_{посл} = \frac{n}{2} + \frac{n}{4} + \ldots + 1 = n - 1. \qquad (2)$$

Если все операции на каждой итерации каскадной схемы выполняются параллельно, количество параллельных операций равно числу итераций k каскадной схемы:

$$K_{пар} = k = log_2 n. \qquad (3)$$

Полагая время выполнения всех вычислительных операций одинаковым и равным τ, имеем $T_1 = K_{посл} \cdot \tau$, $T_s = K_{пар} \cdot \tau$ и с учетом (9.2), (9.3) получаем оценки ускорения и эффективности:

$$R = \frac{T_1}{T_s} = \frac{n-1}{log_2 n}, \qquad (4)$$

$$E_s = \frac{n-1}{s \cdot \log_2 n} = \frac{2 \cdot (n-1)}{n \cdot \log_2 n}. \qquad (5)$$

Равенство (5) записано в предположении, что число процессоров, необходимых для реализации каскадной схемы, выбрано равным $s = n/2$. Нетрудно заметить, что эффективность каскадной схемы падает с ростом числа слагаемых:

$$\lim_{n \to \infty} p_s = 0.$$

Указанный недостаток преодолевается применением модифицированной каскадной схемы. Граф-схема соответствующего этой схеме алгоритма для случая $n = 2^k$, $k = 2^s$, s=2 приведена на рис. 1. Здесь цифрами 1-16 обозначены операции ввода, а цифрами 17-31 – операции суммирования. В этом варианте каскадной схемы вычисления проводятся в два этапа:

- на первом этапе все суммируемые значения подразделяются на $n / log_2 n$ групп по $log_2 n$ элементов в каждой группе, вычисления внутри группы выполняются последовательно, а вычисления для групп осуществляются параллельно на $s = n / \log_2 n$ процессорах;

- на втором этапе к полученным $n/\log_2 n$ суммам применяется каскадная схема.

На первом этапе требуется $log_2 n$ операций (при использовании $s = n / log_2 n$ процессоров). Для выполнения второго этапа необходимо

$$log_2\left(n / log_2 n\right) \le log_2 n \qquad (6)$$

параллельных операций, выполняемых на

$$s_2 = \left(n / log_2 n\right) / 2$$

процессорах. С учетом неравенства (9.6) для описанной схемы при

$$s = n / log_2 n \qquad (7)$$

имеем

$$T_s \cong 2 \cdot log_2 n. \qquad (8)$$

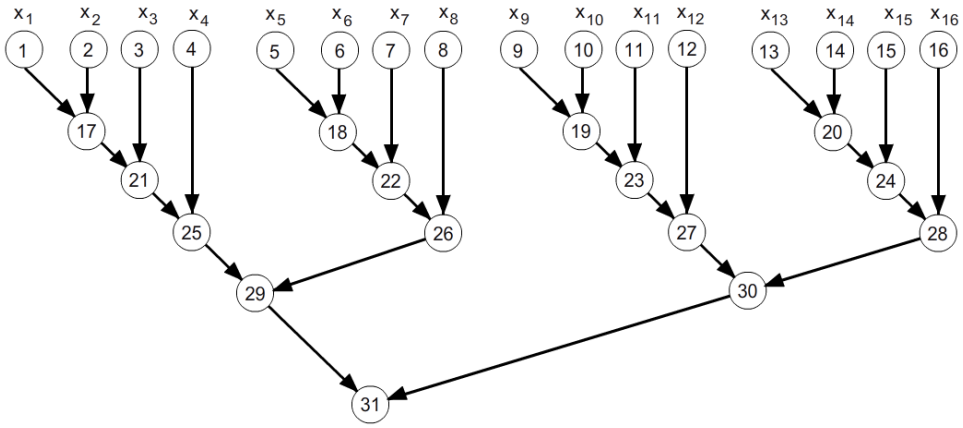


Рис. Модифицированная каскадная схема суммирования

С учетом приведенных оценок (7), (8) показатели ускорения и эффективности модифицированной каскадной схемы определятся соотношениями

$$R = \frac{T_1}{T_s} = \frac{n-1}{2 \cdot log_2 n}, \qquad (9)$$

$$E_s = \frac{R}{s} = \frac{n-1}{2 \cdot \log_2 n \cdot \left(n / \log_2 n\right) \cdot} = \frac{n-1}{2 \cdot n}. \qquad (10)$$

Сравнивая оценки (9), (10) с показателями обычной каскадной схемы (4), (5), нетрудно заметить, что ускорение в данном случае уменьшилось в 2 раза, зато имеет место ненулевая оценка снизу для эффективности:

$$\lim_{n\to\infty} E_s = \lim_{n\to\infty} \frac{n-1}{2 \cdot n} = 0{,}5.$$

В отличие от обычной каскадной схемы, модифицированный каскадный алгоритм является *оптимальным по стоимости*, поскольку вычислительные затраты в данном случае определяются как

$$C_s = sT_s = \left(n / log_2 n\right)\left(2 log_2 n\right) = 2n,$$

т.е. пропорциональны времени выполнения последовательного алгоритма.

2. Вычисление определенного интеграла

Пусть требуется вычислить с точностью ε значение определенного интеграла

$$J(A,B) = \int_A^B f(x)dx$$

Пусть на отрезке [*a*,*b*] задана равномерная сетка, содержащая *n+1* узел:

$$x_i = a + \frac{b-a}{n}$$

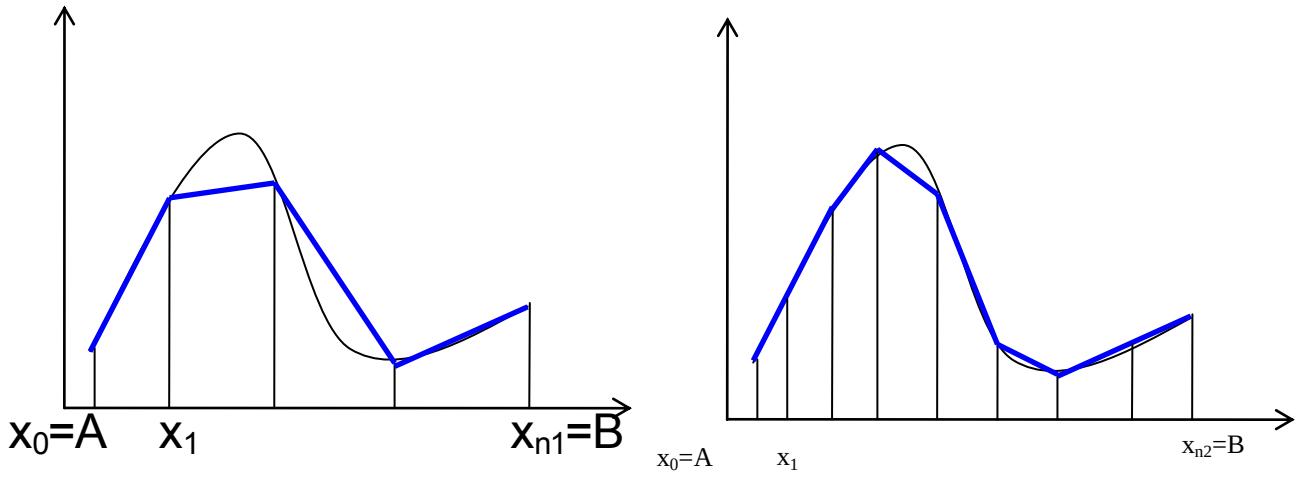
Тогда, согласно методу трапеций, можно численно найти определенный интеграл от функции на отрезке [a,b]:

$$J_n(a,,b) = \frac{b-a}{n}\left(\frac{f(x_0)+f(x_n)}{2}\right) + \sum_{i=1}^{n-1} f(x_i)$$

Метод двойного пересчета Будем полагать значение J найденным с точностью ε,
если выполнено условие:

$$\left|J_{n1} - J_{n2}\right| \le \varepsilon \left|J_{n2}\right|$$

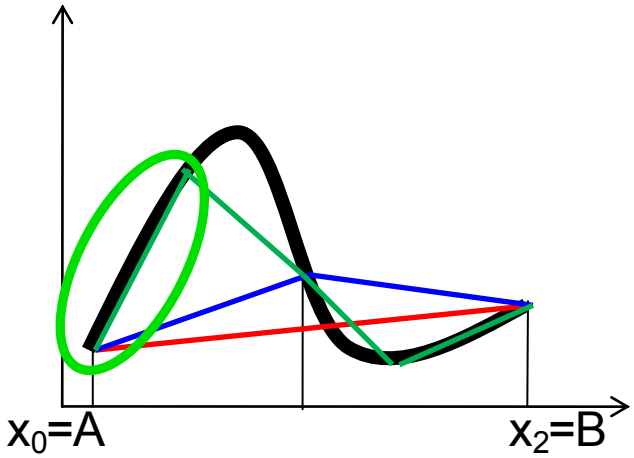
n1<n2



Недостатки:

- в некоторых точках значение подынтегральной функции вычисляется более одного раза
- на всем интервале интегрирования используется равномерная сетка, тогда как число узлов сетки на единицу длины на разных участках интервала интегрирования, необходимое для достижения заданной точности, зависит от вида функции

```
IntTrap01(A,B){
n=1
J2n=(f(A)+f(B))(B-A)/2
  do {
Jn= J2n
n=2n
 s=f(A)+f(B)
for(i=1;i<n;i++)
          s+=2f(A+(B-A)i/n);
J2n=s(B-A)/n;
}while(|J2n- Jn|ε≥ J2n)
return J2n
```




```
main()
{
J= IntTrap03(  A, B, f(A),f(B)  )
}
```

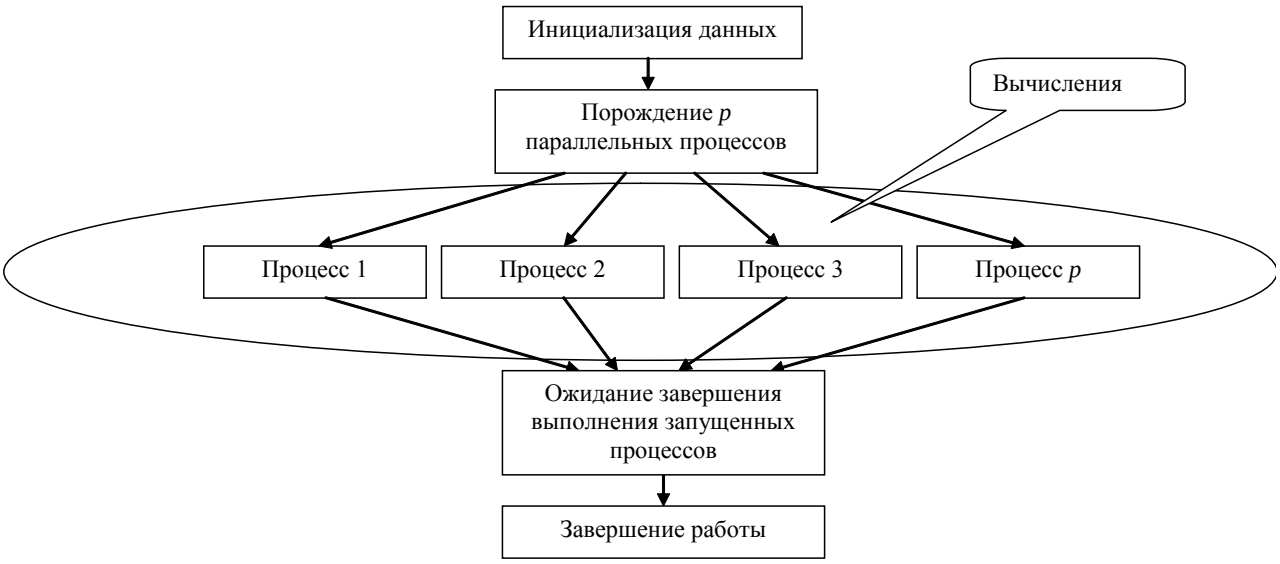
Адаптивный алгоритм

```
IntTrap03(A,B,fA,fB) {
J=0
C=(A+B)/2
fC=f(C)
sAB=(fA+fB)*(B-A)/2
sAC=(fA+fC)*(C-A)/2
```

```
sCB=(fC+fB)*(B-C)/2
sACB=sAC+sCB
if(|sAB-sACB|ε≥|sACB|)
        J=IntTrap03(A,C,fA,fC)+IntTrap03(C,B,fC,fB)
else
        J=      sACB
return J
}
```

1)Порождение p параллельных процессов, каждый из которых выполняет процедуру slave
2) Пока есть отрезки, не переданные для отработки, следует дождаться сообщения от любого из процессов slave,вычислившего частичную сумму на переданном ему отрезке, Получить значение этой суммы, прибавить к общему значению Интеграла и передать освободившемуся процессу очередной отрезок
3) Получить результаты вычислений переданных отрезков и прибавить их к общей сумме

Преимущества:
нет повторных вычислений функции
малое число вычислений на гладких участках
Недостаток:
Большой дисбаланс загрузки процессоров
Метод глобального стека
Вычислительные системы с общей памятью
Динамическая балансировка загрузки
Отсутствие централизованного управления



Задания:

Задание 1

Написать MPI-программу, в которой из имеющегося числа процессов создается две группы №1 и №2, содержащие N и M процессов соответственно. Для каждой группы определить собственный коммутатор. Ранги процессов задать произвольно.

Создать новую группу №3 путем [операция] двух ранее созданных групп № 1 и № 2 соответственно. Распечатать ранги процессов в новых группах

Создать новую группу №4 путем исключения из исходной группы №3 2 процессов с произвольными рангами.

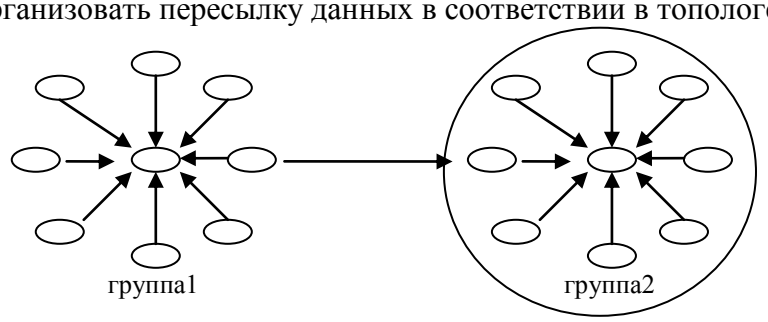
Произвести сравнение групп а) №4 и №2 б) №4 и №3.

Удалить ранее созданные коммутаторы и группы.

Вариант №	N	M	операция
1	3	3	разность
2	2	3	объединение
3	4	2	пересечение

Задание 2

Организовать пересылку данных в соответствии в топологей зведа+звезда:



Сначала в каждой группе собираются данные на одном процессе, используя редукцию, а затем данные с группы 1 передаются в группу 2.

Для передачи данных между группами создать интеркоммуникатор.

2) Расщепить группу2 на 3 части. Организовать передачу данных в них.

Контрольные вопросы

1. Какие функции предназначены для работы с файлами?
2. Каким правам доступа соответствует обозначение 0352?
3. Что такое файловый дескриптор? Для чего он нужен?
4. Какое значение возвращает функция open() при удачном завершении?
5. Зачем нужно проверять код возврата из функций ввода-вывода?
6. Что означают флаги O_CREAT|O_APPEND при открытии файла?
7. Если файл открыт для чтения и записи с флагом O_APPEND, можно ли читать данные из произвольного места в файле с помощью функции lseek()?
8. Можно ли воспользоваться функцией lseek() для изменения данных в произвольном месте в файле?
9. Что произойдёт, если пользователь создаст файл функцией open("pathname", 0466), запишет в него какую-либо информацию, а затем попробует открыть через файловый менеджер или терминал?

Порядок выполнения работы

1. Изучите теоретические сведения.
2. Изучите примеры программ, приведенных в теоретической части.
3. Модифицируйте примеры для выполнения задания
4. Оформите отчет.

Форма отчетности:

Отчет по работе содержит:

1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе

При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .

2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

Лабораторная работа 4

Топологии сетей передачи данных

Цель работы: **:** Получение навыков применения различных топологий.

Теоретические сведения

1. Линейная топология. Каждый процесс с рангом, не превосходящим size-1 пересылает массив a[] процессу с рангом, большим на 1.

```
#include<stdio.h>
#include<mpi.h>
int main(int argc, char **argv)
{
  int  size, rank  a[10], i;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank==0)
 {
    for (i=0; i<10; i++)          a[i]=2*i;
     MPI_Send(a,10,MPI_INT,1,0,MPI_COMM_WORLD);
 }
 else if (rank>0 &&rank<(size-1)) {
     MPI_Recv(a,10,MPI_INT, rank-1, 0, MPI_COMM_WORLD, &status);
     printf("\n rank=%d  data recived \n" , rank);
     MPI_Send(a,10,MPI_INT,rank+1,0,MPI_COMM_WORLD);
   }
   else {
          MPI_Recv(a,10,MPI_INT, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
         for (i=0;i<10;i++)      printf("a[%d]= %d  ", i, a[i]);
   }
  MPI_Finalize();
 return 0;
}
```

2. Кольцевая топология

```
#include<stdio.h>
#include<mpi.h>
int main(int argc, char **argv)
{
  int  size, rank, N=5, a[N],b[N], i, m;
  MPI_Status status;
  MPI_Init(&argc, &argv);
```

```
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if (rank==0)
{
  for (i=0; i<N; i++)         a[i]=i;
for (i=1; i<size; i++)
    MPI_Send(a,N,MPI_INT,i,0,MPI_COMM_WORLD);
}
   else {
        MPI_Recv(b,N,MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
     printf("\n Process with rank %d recived: \n", rank);
       for (i=0;i<N;i++)            printf("b[%d]= %d  ", i, b[i]);
 }
 MPI_Finalize();
return 0;  }
```

Умножение матрицы на вектор, способы декомпозиции

Операция умножения матрицы на вектор – одна из самых часто встречающихся в самых различных задачах. Поэтому многие стандартные библиотеки программ содержат процедуры для выполнения матричных операций. Тем не менее изучение основных схем их распараллеливания полезно, т.к. во-первых, это дает необходимые сведения для обоснованного выбора наиболее подходящей для системы с данной топологией процедуры из имеющегося набора стандартных, во-вторых, матричные операции являются классическими примерами для демонстрации многих приемов и методов распараллеливания задач. Далее для общности будем полагать, что матрицы являются плотными, т.е. число нулевых элементов в них мало по сравнению с общим количеством элементов матриц.

При распараллеливании задачи умножения матрицы на вектор обычно используются два типа декомпозиции, показанные на рис.

1. Ленточное разбиение – на полосы по горизонтали или вертикали.
2. Разбиение данных на прямоугольные фрагменты (блочное разделение).

При ленточном разбиении каждому процессору выделяется некоторое количество (обычно подряд идущих) строк или столбцов. При этом в случае горизонтального разбиения матрица A представляется в виде

$$A = (A_0, A_1, ..., A_{s-1})^T, \qquad A_i = (a_{i0}, a_{i1}, ..., a_{ik-1}),$$

$$ij = ik + j, 0 \le j < k, k = m/s,$$

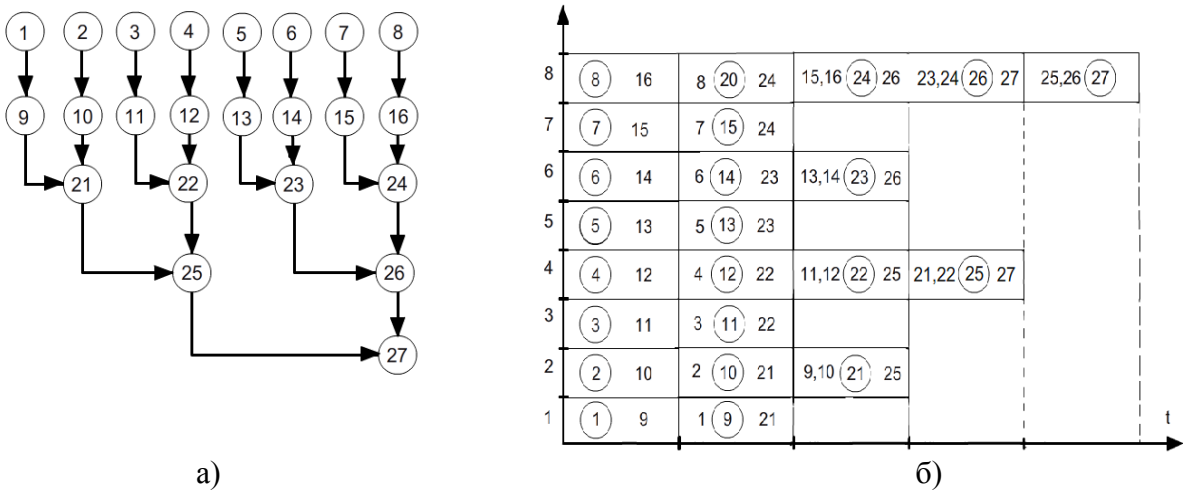
где $a_i = (a_{i,1}, a_{i,2}, ..., a_{i,n}), 0 \le i \le m$ – *i*-я строка матрицы *A* (предполагается, что количество строк *m* кратно числу процессоров: $m=k \cdot s$). Иногда с точки зрения балансировки процессоров для конкретной топологии более предпочтительным является циклическое чередование строк или столбцов. В этом случае

$$ij = i + js, 0 \le j < k, k = m/s.$$

На рис. 9.3 в качестве примера приведены граф-схема и временная диаграмма параллельного алгоритма умножения матрицы на вектор на 8 процессорах при ленточном разбиении по строкам. Здесь цифрами 1-8 обозначены операции ввода данных, цифрами 9-16 – операции умножения выделенных каждому процессору полос на вектор. В результате выполнения этих операций на каждом процессоре будет получена 1/8 часть искомого вектора. Цифрами 21-27 обозначены операции «сборки». Сборка осуществляется за три шага. После первого шага (выполнения операций 21-24) на 4 процессорах окажется по 1/4 части искомого вектора. На следующем шаге (операции 25-26) на двух процессорах будет сформировано по 1/2 части искомого вектора. Наконец на завершающем 3-м шаге в результате выполнения операции 27 будет получен результирующий вектор.

При ленточном разбиении по столбцам схема формирования результирующего вектора существенно отличается. Сборке результирующего вектора в этом случае предшествует

формирование промежуточных векторов. Соответствующий пример будет приведен в разделе 9.5.



а) б)

Рис. 9.3 Граф алгоритма (а) и временная диаграмма (б)
при разделении матрицы по строкам

При блочной декомпозиции матрица делится на прямоугольные фрагменты обычно из подряд идущих элементов. Если количество процессоров $s = p \cdot q$, количество строк матрицы кратно *p*, а количество столбцов – *q* ($m = k \cdot p$, $n = l \cdot q$), матрицу *A* можно представить в виде набора прямоугольных блоков следующим образом:

$$A = \begin{bmatrix} A(1,1) & A(1,2)\ldots & A(1,q) \\ \ldots. & \ldots & \ldots \\ A(p,1) & A(p,2)\ldots & A(p,q) \end{bmatrix},$$

где $A(i,j)$ — блок матрицы, состоящий из элементов:

$$A(i,j) = \begin{bmatrix} a_{1,1}(i,j) & a_{1,2}(i,j)\ldots & a_{1,l}(i,j) \\ \ldots. & \ldots & \ldots \\ a_{K,1}(i,j) & a_{K,2}(i,j)\ldots & a_{K,l}(i,j) \end{bmatrix}.$$

При блочном разбиении целесообразна топология системы в виде решетки из *p* строк и *q* столбцов. При этом вычислительный процесс следует организовать так, чтобы соседние в структуре решетки процессоры обрабатывали смежные блоки исходной матрицы. Для блочной схемы может быть применено также циклическое чередование строк и столбцов.

4 Умножение матрицы на вектор, разделение по строкам

В данном случае умножение *m*×*n*-матрицы на *n*×1-вектор сводится к вычислению *m* скалярных произведений векторов длины *n*. Каждое такое произведение требует *n* операций умножения и $n-1$ операций сложения. Таким образом, вычислительная сложность последовательного алгоритма составит $T_1 = m(2n-1)$, а в случае квадратной *n*×*n*-матрицы

$$T_1 = n(2n-1). \qquad (9.14)$$

Если умножение *n*×*n*-матрицы выполняется параллельно (см. рис. 9.3), за каждым процессором закрепляется не более

$$m_s =]n/s[\qquad (9.15)$$

строк, где $]*[$ – здесь и далее означает операцию округления до целого в большую сторону. Количество процессоров, за которыми будет закреплено меньше чем m_s строк, определяется конкретными значениями *n* и *s*.

Для построения оценок ускорения и эффективности с учетом затрат на вычисления и коммуникации предполагаем, что все $(2n-1)$ операций умножения и сложения имеют одинаковую длительность τ, а вычислительная система однородна, т.е. все процессоры обладают одинаковой производительностью. Тогда временные затраты параллельного алгоритма, связанные непосредственно с вычислениями, с учетом (9.15) составят

$$T_s = \left\lceil \frac{n}{s} \right\rceil (2n-1)\cdot\tau. \qquad (9.16)$$

Если сеть передачи данных имеет структуру гиперкуба или полного графа, операция сбора данных может быть выполнена за $\lceil log_2 s \rceil$ итераций. На первой итерации взаимодействующие пары процессоров обмениваются сообщениями объемом $w(n/s)$, где w – размер одного элемента вектора в байтах. На второй итерации этот объем увеличивается вдвое и оказывается равным $2w\lceil n/s \rceil$ и т.д. Общая длительность сбора данных

$$T_s = \sum_{i=1}^{\lceil \log_2 s \rceil} \left(\alpha + 2^{i-1} w\right) \lceil n/s \rceil / \beta = \alpha \lceil \log_2 s \rceil + w \lceil n/s \rceil \left(\sum_{i=1}^{\lceil \log_2 s \rceil} 2^{i-1} \right) / \beta, \qquad (9.17)$$

где α– латентность сети передачи данных, β – пропускная способность сети. Можно показать, что

$$\sum_{i=1}^{\lceil \log_2 s \rceil} 2^{i-1} = \left(2^{\lceil \log_2 s \rceil} - 1\right) = s - 1. \qquad (9.18)$$

С учетом (9.16), (9.17) и (9.18) общее время выполнения параллельного алгоритма

$$T_s = \left\lceil \frac{n}{s} \right\rceil (2n-1)\cdot\tau + \alpha \lceil \log_2 s \rceil + w \lceil n/s \rceil (s-1)/\beta. \qquad (9.19)$$

Если n/s и $log_2 s$ целые числа, в соответствии с (9.19) оценки для ускорения и эффективности принимают вид

$$R = \frac{T_1}{T_s} = \frac{n(2n-1)\cdot\tau}{(n/s)(2n-1)\cdot\tau + \alpha(log_2 s) + w(n/s)(s-1)/\beta}, \qquad (9.20)$$

$$E_s = \frac{R}{s} = \frac{n\cdot(2n-1)\cdot\tau}{n\cdot(2n-1)\cdot\tau + s\{\alpha(\log_2 s) + w(n/s)(s-1)/\beta\}}. \qquad (9.21)$$

Из соотношений (9.21) видно, что для сохранения требуемого уровня эффективности при увеличении числа используемых процессоров соответственно должна возрастать вычислительная сложность задачи (размерности матрицы и вектора). Заметим также, что если в (9.20), (9.21) пренебречь затратами на передачу данных, будем иметь

$$R = T_1/T_s = s,$$

$$E_s = R/s = 1.$$

Высокий внутренний параллелизм задачи связан с тем, что в данном случае отсутствуют операции, которые не поддаются распараллеливанию.

5 Умножение матрицы на вектор, разделение по столбцам

При параллельном умножении матрицы на вектор с разделением данных по столбцам каждый столбец матрицы A умножается на один (соответствующий ему) элемент вектора b. Если за процессором закреплена полоса, элементы столбцов, из которых она составлена, умножаются на соответствующие элементы вектора b, которые также должны быть закреплены за этим процессором. После умножения полученные значения суммируются для каждой m-й строки матрицы A в отдельности:

$$c'_m(i) = \sum_{j=j_0}^{j_l-1} a_{mj} b_j, \quad 0 \le m < n,$$

где (j_0 и j_{l-1} – начальный и конечный индексы столбцов подзадачи i, $0 \le i < n$).

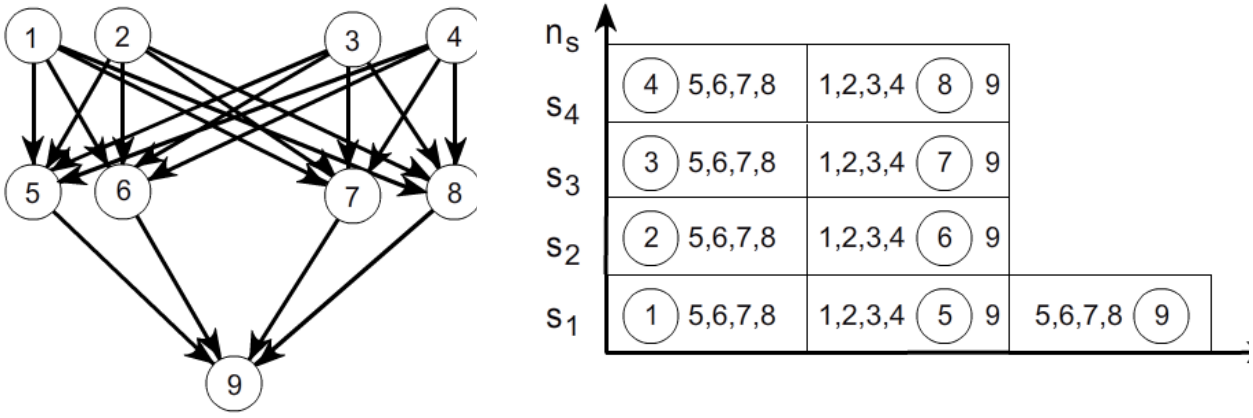
Вычислительные затраты для вычисления одного элемента промежуточного вектора на этом – первом этапе могут быть оценены как

$$2 \cdot]n/s[- 1. \qquad (9.22)$$

В результате реализации первого этапа на каждом процессоре получается столбец промежуточных значений.

Для получения элементов результирующего вектора (второй этап) подзадачи должны обменяться этими промежуточными данными между собой. Притом процессор-отправитель должен отослать на процессор-получатель только ту часть промежуточного вектора (]*m*/*s*[), за вычисление которой в результирующем векторе «отвечает» процессор-получатель. После обмена данными между подзадачами каждый процессор суммирует полученные значения для своего блока результирующего вектора. Завершающая операция – «сборка» результирующего вектора.

На рис. 9.4 в качестве примера приведены граф-схема и временная диаграмма умножения матрицы на вектор при разбиении матрицы по столбцам на 4 процессора. Здесь цифрами 1-4 обозначены операции умножения столбца или выделенной полосы (операции ввода данных для простоты не показаны) на соответствующий элемент (фрагмент) вектора, цифрами 5-8 – операции формирования (пересылка и суммирование) закрепленных за каждым процессором фрагментов искомого вектора, а операция 9 – завершающий этап «сборки» результирующего вектора.



а) б)

Рис. 9.4 Граф алгоритма (а) и временная диаграмма (б) вычислений при разделении матрицы по столбцам на 4 процессора

С учетом сказанного общее время выполнения вычислительных операций параллельного алгоритма на первом и втором этапах вместе составит:

$$T_s =]m \cdot (2 \cdot]n/s[- 1) + (s - 1)]m/s[[\cdot \tau . \qquad (9.23)$$

Заметим, что здесь в соответствии с (9.22)

$$m \cdot (2 \cdot]n/s[- 1)$$

– оценка максимального числа операций (умножения и сложения) для вычисления всех элементов одного промежуточного $m \times 1$-вектора.

Количество суммируемых значений для каждого элемента результирующего вектора на втором этапе равно числу промежуточных векторов, т.е. совпадает с числом процессоров *s*. Максимальный размер блока результирующего вектора, закрепленный за одним процессором-получателем в общем случае равен]*m*/*s*[. Следовательно, оценка числа операций на этом (втором) этапе равна *s*]*m*/*s*[.

Оценим теперь ускорение и эффективность параллельного алгоритма с учетом затрат времени на пересылку данных. Временные затраты на передачу данных, необходимых процессору-получателю для вычисления одного блока результирующего вектора составят

$$\left(\alpha + w\right]m/s\left[/\beta\right), \tag{9.24}$$

где α – латентность сети передачи данных, β – пропускная способность сети, w – размер элемента данных в байтах.

Общие временные затраты на коммуникации зависят от топологии сети. Если имеется возможность одновременно отправлять и принимать сообщения между любой парой процессоров, тогда временные затраты с учетом (9.24) составят

$$T_s^1 = \left(s-1\right)\left(\alpha + w\right]m/s\left[/\beta\right). \tag{9.25}$$

Если топология вычислительной системы представлена в виде гиперкуба, передача всех данных может быть выполнена за $log_2 s$ шагов, на каждом из которых максимально возможная (т.к. в общем случае размеры закрепляемых за процессором блоков различаются) длина передаваемых и получаемых сообщений $\left]m/s\right[$ элементов

$$T_s^2 = \left]\log_2 s\right[\left(\alpha + w\right]m/s\left[/\beta\right). \tag{9.26}$$

К временным затратам, определяемым соотношениями (9.25), (9.26), следует еще добавить время, необходимое на окончательную «сборку» результирующего вектора:

$$\left(s-1\right)\left(\alpha + w\right]m/s\left[/\beta\right).$$

С учетом этих затрат общее время выполнения параллельного алгоритма для указанных способов передачи данных соответственно

$$T_s^1 = \left]m\cdot\left(2\cdot\right]n/s\left[-1\right)+\left(s-1\right)\right]m/s\left[\right[\cdot\tau + 2\cdot\left(s-1\right)\left(\alpha + 2\cdot w\right]m/s\left[/\beta\right),$$

$$T_s^2 = \left[m\cdot\left(2\cdot\left[n/s\right]-1\right)+s\left[m/s\right]\right]\cdot\tau + \left(s-1+\left[log_2 s\right]\right)\left(\alpha + 2\cdot w\left[m/s\right]/\beta\right).$$

Если матрица квадратная размерности $n\times n$, а n кратно числу процессоров, т.е. n/s – целое число, ускорение и эффективность для первого типа топологии сети соответственно

$$R = \frac{n\cdot\left(2n-1\right)\cdot\tau}{\frac{n\left(2\cdot n-1\right)}{s}\cdot\tau + 2\cdot\left(s-1\right)\left(\alpha + 2\cdot w\left(n/s\right)/\beta\right)}, \tag{9.27}$$

$$E_s = \frac{n\cdot\left(2n-1\right)\cdot\tau}{n\left(2\cdot n-1\right)\cdot\tau + 2\cdot s\left(s-1\right)\left(\alpha + 2\cdot w\left(n/s\right)/\beta\right)}. \tag{9.28}$$

Для второго типа топологии сети имеем соответственно

$$R = \frac{n\cdot\left(2n-1\right)\cdot\tau}{\frac{n\left(2\cdot n-1\right)}{s}\cdot\tau + \left(s-1+\right]\log_2 s\left[\right)\left(\alpha + 2\cdot w\left(n/s\right)/\beta\right)}, \tag{9.29}$$

$$E_s = \frac{n\cdot\left(2n-1\right)\cdot\tau}{n\left(2\cdot n-1\right)\cdot\tau + s\left(s-1+\right]\log_2 s\left[\right)\left(\alpha + 2\cdot w\left(n/s\right)/\beta\right)}. \tag{9.30}$$

В данном случае также отсутствуют операции, которые могут выполняться только последовательно, поэтому без учета потерь на коммуникации (9.25), (9.26) максимально возможные теоретические значения ускорения и эффективности в соответствии с (9.29), (9.30) равны соответственно $R=s$, $E_s=1$.

6 Умножение матрицы на вектор при блочном разделении данных

Общее время умножения блоков матрицы A размером $(n/p)\times(n/q)$ на соответствующие блоки вектора b можно записать, используя полученные ранее формулы для ленточного разбиения по строкам и столбцам:

$$T_s = \left]\right]n/p\left[\cdot\left(2\cdot\right]n/q\left[-1\right)\right[\cdot\tau.$$

Здесь $\left(2\cdot\left[n/q\right]-1\right)$ - количество операций сложения и умножения в одной строке блока при разбиении на n/q столбцов, а n/p - количество строк в каждом блоке.

С учетом последующего суммирования промежуточных векторов вычислительные затраты увеличатся. Объединение промежуточных результатов может быть выполнено с использованием каскадной схемы за $log_2 q$ итераций. Приближенно эти затраты можно учесть, добавив слагаемое $\tau\ log_2 q$. При этом оценка общих потерь с учетом потерь на коммуникации составит

$$T_s = \left\lceil n/p\right\rceil\cdot\left(2\cdot\left\lceil n/q\right\rceil-1\right)\cdot\tau+\left\lceil\log_2 q\right\rceil\left(\tau+\alpha+w\left(n/p\right)/\beta\right).$$

Если количество строк матрицы кратно *p*, а количество столбцов – кратно *q*, т. е. $s=p\cdot q$, $m=k\cdot p$ и $n=k\cdot q$, выражения для ускорения и эффективности можно записать в виде

$$R=\frac{n\left(2n-1\right)\cdot\tau}{\left(n/p\right)\cdot\left(2\cdot\left(n/q\right)-1\right)\cdot\tau+\left\lceil\log_2 q\right\rceil\left(\tau+\alpha+w\left(n/q\right)/\beta\right)}, \qquad (9.31)$$

$$E_s=\frac{n\left(2n-1\right)\cdot\tau}{n\cdot\left(2n-q\right)\cdot\tau+s\left\lceil\log_2 q\right\rceil\left(\tau+\alpha+w\left(n/q\right)/\beta\right)}. \qquad (9.32)$$

Напомним, что здесь $s=p\cdot q$.

В заключение укажем на возможный способ выбора блочной структуры матрицы *A*. В данном случае увеличение числа блоков по горизонтали приводит к возрастанию числа итераций в операции суммирования промежуточных векторов, а увеличение числа блоков по вертикали позволяет снизить объем передаваемых данных между процессорами.

Если пренебречь затратами на телекоммуникации, то в соответствии с соотношениями (9.31), (9.32) ускорение равное *s* и эффективность 1 достигаются при

$$log_2 q=\frac{n\left(q-1\right)}{s}=\frac{n}{p}\frac{\left(q-1\right)}{q}.$$

Это соотношение может использоваться для подбора соотношения между $p\cdot$и q, обеспечивающего соблюдение указанного свойства.

Перемножение матриц

1 Последовательный алгоритм умножения матриц

Умножение матрицы A размера $m\times n$ на матрицу B размера $n\times l$ приводит к получению матрицы **C** размера $m\times l$, каждый элемент которой

$$c_{ij}-\sum_{k=0}^{n-1}a_{ik}\cdot b_{kj},\quad 0\le i<m,\quad 0\le j<l,$$

т.е. каждый элемент матрицы *C* вычисляется как скалярное произведение соответствующих строки матрицы *A* и столбца матрицы *B*:

$$c_{ij}=\left(a_i,b_j^T\right),$$

$$a_i=\left(a_{i,0},a_{i,1},\ldots,a_{i,n-1}\right),$$

$$b_j^T=\left(b_{i,0},b_{i,1},\ldots,b_{i,n-1}\right)^T$$

Легко подсчитать, что для реализации перемножения матриц потребуется $m\times n\times l$: операций умножения и $m\times(n\text{-}1)\times 1$ операций сложения. Существуют приемы, позволяющие перемножить матрицы за меньшее число операций. Однако мы ограничимся рассмотрением алгоритма, реализующего указанный простейший случай. Подсчитаем общее число операций последовательного алгоритма для перемножения квадратных $n\times n$-матриц.

Для вычисления одного элемента потребуется $2n-1$ операций (*n* операций умножения и $n-1$ операций сложения); для вычисления одной строки – $n\cdot\left(2n-1\right)$ операций, а общее

число операций сложения и умножения для получения результирующей матрицы C в последовательном алгоритме составит

$$N_{A*B} = n^2 \cdot (2n-1). \qquad (10.1)$$

Далее рассматриваются два возможных параллельных алгоритма перемножения матриц при ленточном разбиении и декомпозиции на блоки.

Параллельные алгоритмы при ленточном разбиении матрицы

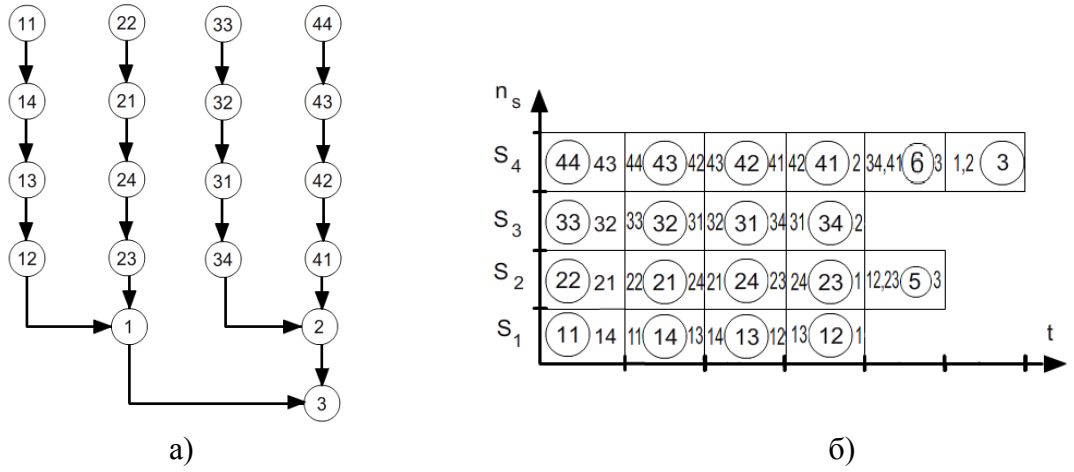
Рассмотрим два параллельных алгоритма умножения матриц с ленточным разбиением на строки или столбцы (полосы). Если каждая подзадача содержит по одной строке матрицы A и одному столбцу матрицы B, общее число подзадач (необходимых процессоров) равно n^2. При большой размерности матриц это может быть неприемлемо. Чаще декомпозицию осуществляют таким образом, что подзадача заключается в вычислении элементов одной строки матрицы C. При этом количество подзадач (потребных процессоров) равно n. Далее рассмотрим именно этот случай.

Для выполнения всех вычислений в подзадаче процессору должны быть доступны одна из строк матрицы A и все столбцы матрицы B. Если дублирование матрицы B во всех подзадачах неприемлемо в силу большой размерности матрицы, необходимо строить итерационный процесс с обменом данными между процессорами. Возможны две схемы организации вычислений с обменом данными.

Первый алгоритм. На каждой итерации каждая подзадача содержит по одной строке матрицы A и одному столбцу матрицы B. При выполнении итерации проводится скалярное умножение содержащихся в подзадачах строк и столбцов, что приводит к получению соответствующих элементов результирующей матрицы C. В конце каждой итерации столбцы матрицы B передаются между подзадачами. Процесс передачи организуется так, чтобы после завершения итераций в каждой подзадаче последовательно «побывали» все столбцы матрицы B.

Для указанной схемы передачи данных наиболее подходящей является топология связей подзадач в виде кольцевой структуры. При этом на каждой итерации подзадача i, $0 \le i < n$ передает свой столбец матрицы B подзадаче с номером $i+1$, а подзадача $n-1$ передает свои данные подзадаче с номером 0. На рис. 10.1 приведены граф-схема и временная диаграмма алгоритма перемножения матриц для случая, когда матрицы состоят из четырех строк и четырех столбцов ($n=4$).

Для наглядности здесь для операций вычисления произведений строк на столбцы принята нумерация в виде двух цифр. Первая цифра обозначает номер строки матрицы A, а вторая – номер столбца матрицы B в произведении. Цифрами 1, 2 и 3 обозначены операции «сборки» результатов. В частности, оператором 1 осуществляется сборка части искомой матрицы, включающей первую и вторую строку, а оператором 2 – объединение третьей и четвертой строк. Операция 3 – реализует завершающий этап «сборки» всей результирующей матрицы.



а) б)

Рис. 10.1 Граф-схема (а) и временная диаграмма (б)
первого алгоритма перемножения матриц (блоки по строкам)

Второй алгоритм. Отличие в том, что в подзадачах располагаются не столбцы, а строки матрицы *B*. А перемножение матриц сводится к умножению строк матрицы *B* на соответствующие элементы строк матрицы *A* и последующему сложению получающихся строк.

Например, в случае, когда за каждым процессором закреплена одна строка матрицы *A*, для получения первой строки матрицы ***C*** каждая, например, *i*-я строка матрицы *B* умножается на соответствующий элемент, в данном случае *i*-й, первой строки матрицы *A*, а затем все полученные промежуточные векторы (*i*=1,2,..,*n*) складываются. Для получения второй строки матрицы *C* все строки матрицы *B* умножаются на соответствующие элементы второй строки матрицы *A* и т.д.

В этой схеме вычислений после каждого шага умножения очередного элемента матрицы *A* на соответствующую строку матрицы *B* в каждой подзадаче (на каждом процессоре) получается строка частичных результатов, которые поэлементно суммируются со своей строкой матрицы *C*. Для выполнения следующего шага умножения строк матрицы *B* на элементы матрицы *A* необходимо передать во все подзадачи соответствующие строки матрицы *B*. Передача строк матрицы *B* между подзадачами в данном случае может быть выполнена с использованием кольцевой структуры информационных связей, т.е. наиболее подходящей является топология «кольцо».

До начала вычислений в каждой подзадаче *i*, $0 \le i < n$ располагаются *i*-е строки матрицы *A* и матрицы *B*. В результате их перемножения подзадача определяет *i*-ю строку частичных результатов искомой матрицы *C*. Далее подзадачи осуществляют обмен строками, в ходе которого каждая подзадача передает свою строку матрицы *B* следующей подзадаче до завершения всех итераций параллельного алгоритма.

Для случая, когда матрицы состоят из четырех строк и четырех столбцов (*n*=4), граф-схема алгоритма и временная диаграмма могут быть представлены в точно таком же виде, как это показано на рис. 10.1. Для этого достаточно придать другое содержание обозначениям операций. В частности, теперь первая цифра в обозначении операции перемножения должна служить для обозначения столбца матрицы A, а вторая – строки матрицы B.

Цифрой 1 теперь обозначена операция объединения первого и второго столбца, цифрой 2 – операция объединения третьего и четвертого столбца, а цифрой 3 – операция «сборки» всех столбцов искомой матрицы.

Если размерность матрицы *n* оказывается больше, чем число процессоров *s*, подзадачи можно укрупнить, объединив несколько соседних строк и столбцов перемножаемых матриц. При этом исходная матрица *A* разбивается на ряд горизонтальных полос, а матрица *B* представляется в виде набора вертикальных (для первого алгоритма) или горизонтальных (для второго алгоритма) полос. Если *n* кратно *s*, размер полос *k*=*n*/*s*. Распределение подзадач между процессорами следует осуществлять таким образом, чтобы подзадачи, являющиеся соседними в кольцевой топологии, располагались на процессорах, между которыми имеются прямые линии передачи данных.

Контрольные вопросы

1. Какую функцию нужно использовать, чтобы получить PID текущего процесса?
2. Какую работу выполняет функция fork()?
3. Что общего у родительского процесса и дочернего, порожденного вызовом fork()?
4. Чем отличаются родительский и дочерний процессы?
5. Можно ли гарантировать, что родительский и дочерний процессы будут выполнять свои операции в определённом порядке? Ответ обоснуйте.
6. Какие функции используются для загрузки нового кода в текущий процесс?
7. Может ли программа, вызванная exec*() обращаться к переменным, объявленным ранее в родительском процессе?
8. В чем различие между функциями wait() и waitpid()?

Порядок выполнения работы

1. Изучите теоретические сведения.
2. Изучите примеры программ, приведенных в теоретической части.
3. Напишите программу, которая выводит сообщение, содержащее свой PID и PID своего родителя. Сколько сообщений было выведено на экран? Объясните полученные результаты.

Модифицируйте предыдущую программу так, чтобы было выведено ровно 3 сообщения.

4. Оформите отчет.

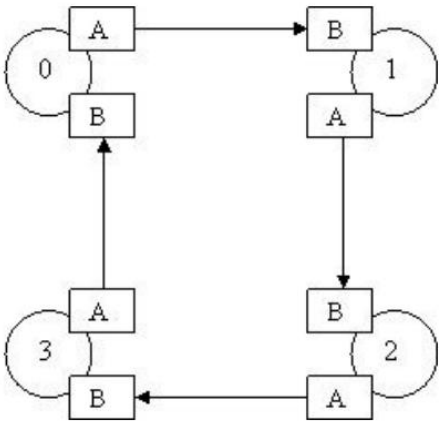
Варианты заданий

Вариант 1

1. Процессы с четным рангом генерируют массивы из 5 целых чисел и отсылают их процессам с рангом на 1 больше. Процессы с нечетными рангами сортируют массивы по возрастанию . .

2. Организовать передачу массива вдоль кольца компьютеров. Процесс с наибольшим рангом пересылает данные 0 процессу. В этом примере программу алгоритма нужно написать настраиваемой автоматически на размер вычислительной системы, программа должна запускаться на любом допустимом количестве процессов.

3. Все ветви одновременно создают и пересылают некоторые свои данные ветвям с номерами на единицу большими, чем передающие ветви. Процесс с наибольшим рангом пересылает данные 0 процессу . В этом примере программу алгоритма нужно написать настраиваемой автоматически на размер вычислительной системы, т.е. программа должна запускаться на любом допустимом количестве процессов.



.

Форма отчетности:

Отчет по работе содержит:

1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе

При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .

2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

Лабораторная работа 5.

Синхронизация обмена данными

Цель работы: : Изучить разновидности операций передачи данных в MPI

Теоретический материал

Режимы передачи данных

Рассмотренная ранее функция *MPI_Send* обеспечивает так называемый *стандартный Standard*) режим отправки сообщений, при котором операция передачи данных считается завершенной, как только сообщение отправлено, независимо от того, получено оно, или еще нет.

Кроме стандартного режима в MPI предусматриваются следующие дополнительные режимы передачи сообщений:

• *Синхронный режим* состоит в том, что завершение функции отправки сообщения происходит только при получении от процесса-получателя подтверждения о начале приема отправленного сообщения, отправленное сообщение или полностью принято процессом-получателем или находится в

состоянии приема,

• *Буферизованный* (*Buffered*) *режим* предполагает использование дополнительных системных буферов для копирования в них отправляемых сообщений; как результат, функция отправки сообщения завершается сразу же после копирования сообщения в системный буфер, и неважно, дошло сообщение до адресата или нет.

• *Режим передачи по готовности* (*Ready*) может быть использован только, если операция приема сообщения уже инициирована. Сразу после этого передача считается завершенной. Буфер сообщения после завершения функции отправки сообщения может быть повторно использован.

Для именования функций отправки сообщения для разных режимов выполнения в MPI используется название функции *MPI_Send*, к которому как префикс добавляется начальный символ названия соответствующего режима работы, т.е.

- MPI_Ssend – функция отправки сообщения в синхронном режиме,
- MPI_Bsend – функция отправки сообщения в буферизованном режиме,
- MPI_Rsend – функция отправки сообщения в режиме по готовности.

Список параметров всех перечисленных функций совпадает с составом параметров функции *MPI_Send*.

Буферизованная передача

Для использования буферизованного режима передачи сначала должен быть создан и передан MPI буфер памяти– используемая для этого функция имеет вид:

int MPI_Buffer_attach(void *buf, int size),
где
- buf - буфер памяти для буферизации сообщений,
- size – размер буфера.

Размер памяти, требуемый для хранения одного сообщения, вычисляется с помощью функции MPI_Pack_size (count, type, comm, *size)
где
-count
-type
-comm
size – место, куда помещается итоговый размер

В буфер вместе с сообщением помещается служебная информация, размер которой MPI_BSEND_OVERHEAD

После завершения работы с буфером он должен быть отключен от MPI при помощи функции:
int MPI_Buffer_detach(void *buf, int *size).

По практическому использованию режимов можно привести следующие рекомендации:
1. Режим передачи по готовности формально является наиболее быстрым, но используется достаточно
редко, т.к. обычно сложно гарантировать готовность операции приема,
2. Стандартный и буферизованный режимы также выполняются достаточно быстро, но могут приводить к большим расходам ресурсов (памяти) – в целом может быть рекомендован для передачи коротких сообщений,
3. Синхронный режим является наиболее медленным, т.к. требует подтверждения приема. В тоже время, этот режим наиболее надежен – можно рекомендовать его для передачи длинных сообщений.
В заключение отметим, что для функции приема *MPI_Recv* не существует различных режимов работы.
Примеры
1. Буферизованный режим

```
#include<stdio.h>
#include <malloc.h>
#include<mpi.h>
int main(int argc, char **argv){
  int proc_num, rank;
  int i,  tag=0,msize, bsize, m,x ; //m сообщений по одному x
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &proc_num);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  m=proc_num;
  if (rank==0){
     MPI_Pack_size(1,MPI_INT,MPI_COMM_WORLD,&msize);
     bsize=m*(msize+MPI_BSEND_OVERHEAD);
     int*buf=(int*)malloc(bsize);
     MPI_Buffer_attach(buf,bsize);
     for(i=1;i<m;i++){//каждому процессу по числу
           x=2*i;
           MPI_Bsend(&x,1,MPI_INT,i, tag,MPI_COMM_WORLD);
     }
      for(i=1;i<m;i++){//ещё раз
           x=2*i+1;
           MPI_Bsend(&x,1,MPI_INT,i, tag,MPI_COMM_WORLD);
     }
     MPI_Buffer_detach (buf,&bsize);
     printf ("All get on \n");
  }
  if (rank!=0){
            MPI_Recv(&x,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
            printf("Proc %d, get  x=%d  \n", rank, x);
            MPI_Recv(&x,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
            printf("Proc %d, get  x=%d  \n", rank, x);
  }
  MPI_Finalize();
 return 0;
}
```

. Синхронный режим

```
#include<stdio.h>
#include<mpi.h>
#include <unistd.h>
#include <stdlib.h>

void do_busy(int r){
    printf ("Process %d is busy \n",r);
    usleep(r*100000);
    printf ("Process %d is free \n",r);
}

int main(int argc, char **argv)
{
  int proc_num, rank;
  int i,  tag=0,x;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &proc_num);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  if (rank==0){
      x=60;
      for(i=1; i<proc_num; i++){
            MPI_Ssend(&x,1,MPI_INT, i, tag,MPI_COMM_WORLD);
             x++;
      }
  }
  if (rank!=0){
             do_busy(rank);
             MPI_Recv(&x,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
             printf("Process %d, got  x=%d  \n", rank, x);
  }
  MPI_Finalize();
 return 0;
}
```

Задание 1

Написать программу использующую синхронную передачу данных между двумя процессами:

Процесс 0:

1) создает массив из 100 целых чисел и посылает их процессу 1,
2) после чего вычисляет наименьший элемент и сразу посылает его процессу 1.
3) Затем принимает от процесса 2 переработанный массив и пишет сообщение

Процесс1:

1) Принимает у процесса 1 массив
2) печатает его
3) возводит его элементы в квадрат
4) отправляет его обратно
5) Принимает от процесса 0 наименьший элемент и печатает его

Измерить время работы программы.

Заменить все синхронные функции на стандартные и снова измерить время.

Задание 2

Написать программу, осуществляющую буферизованную передачу данных между двумя процессами

Процесс 0:

1) создает буфер для 30 букв
2) отправляет их процессу 1
3) принимает результат
4) печатает его

Процесс1:

1) Принимает у процесса массив букв
2) печатает его
3) изменяет массив
4) отправляет обратно

Контрольные вопросы

1. Каким образом родительский процесс отправляет сигнал дочернему?
2. Среди перечисленных сигналов, укажите управляющие заданиями: SIGINT, SIGFPE, SIGSTOP, SIGKILL, SIGCHILD.
3. Какие варианты реакции на сигнал предусмотрены системой?
4. Какие два сигнала невозможно перехватить или проигнорировать?
5. Для чего нужна структура sigaction?
6. Может ли функция-обработчик сигнала возвращать целое значение?
7. Как Вы думаете, почему не рекомендуется совместное использование функций alarm() и sleep()?
8. Какие виды таймеров используются в системе?

Порядок выполнения работы

1. Изучите теоретические сведения.
2. Изучите примеры программ, приведенных в теоретической части.
3. Выполните своё задания согласно варианту.
4. Получите распечатки примеров работы программы.
5. Оформите отчет.

Форма отчетности:

Отчет по работе содержит:

1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе

При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .

2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

Лабораторная работа 6.

Сортировка последовательности

Цель работы:

Изучить особенности функционирования неименованных каналов. Научиться организовывать передачу данных от процесса к процессу. Освоить применение программных каналов и программ-фильтров.

Теоретические сведения

Параллельная пузырьковая сортировка

Постановка задачи

Сортировка является одной из типовых проблем обработки данных и обычно понимается как задача размещения элементов неупорядоченного набора значений

S={a1, a2, …, an}

в порядке монотонного возрастания или убывания

S ~ S’ = { (a′1, a′2, …., a′n): a′1≤ a′2≤ ….≤ a′n }.

Вычислительная трудоемкость процедуры упорядочивания является достаточно высокой. Так, для ряда известных простых методов (пузырьковая сортировка, сортировка включением и др.) количество необходимых операций определяется квадратичной зависимостью от числа упорядочиваемых данных

T ~ n2 .

Для более эффективных алгоритмов (сортировка слиянием, сортировка Шелла, быстрая сортировка) трудоемкость определяется величиной

T1 ~ nlog2 n.

Данное выражение дает также нижнюю оценку необходимого количества операций для упорядочивания набора из n значений; алгоритмы с меньшей трудоемкостью могут быть получены только для частных вариантов задачи.

Ускорение сортировки может быть обеспечено при использовании нескольких (p, p>1) процессоров. Исходный упорядочиваемый набор разделяется между процессорами; в ходе сортировки данные пересылаются между процессорами и сравниваются между собой. Результирующий набор, как правило, также разделен между процессорами; при этом для систематизации такого разделения для процессоров вводится та или иная система последовательной нумерации и обычно требуется, чтобы при завершении сортировки значения, располагающиеся на процессорах с меньшими номерами, не превышали значений процессоров с большими номерами.

2. Описание алгоритма решения задачи

Алгоритм последовательной пузырьковой сортировки

Дадим краткое описание алгоритма.

Последовательный алгоритм "пузырьковой" сортировки сравнивает и обменивает соседние элементы в последовательности, которую нужно отсортировать. Для последовательности

(a1, a2, ..., an)

алгоритм сначала выполняет n-1 операций "сравнения-обмена" для всех последовательных пар элементов (a1, a2), (a2, a3), ..., (an-1,an). На этом шаге самый большой элемент перемещается в конец последовательности. Последний элемент в преобразованной последовательности затем исключается из сортировки, и последовательность "сравнений-обменов" применяется к возникающей в результате сокращенной последовательности (a'1, a'2, ..., a' n-1).

Последовательность будет отсортирована после n-1 итерации. Мы можем улучшить эффективность "пузырьковой" сортировки, переходя к завершению процесса уже тогда, когда не происходит никаких обменов в течение итерации.

Алгоритм 1. Последовательный алгоритм "пузырьковой" сортировки

```
BUBBLE_SORT(n){
for (i=n-1; i<1; i--)
for (j=1; j<i; j++)
compare_exchange(aj, aj+1);
```

}

Итерация внутреннего цикла "пузырьковой" сортировки выполняется за время O(n), всего мы выполняем O(n) итераций; таким образом, сложность "пузырьковой" сортировки - O(n2).

3. Алгоритм четной-нечетной перестановки

Алгоритм пузырьковой сортировки в прямом виде достаточно сложен для распараллеливания – сравнение пар значений упорядочиваемого набора данных происходит строго последовательно. В связи с этим для параллельного применения используется не сам этот алгоритм, а его модификация, известная в литературе как метод чет-нечетной перестановки

Алгоритм "чет-нечетных перестановок" также сортирует n элементов за n итераций (n - четно), однако правила итераций различаются в зависимости от четности или нечетности номера итерации и, кроме того, каждая из итераций требует n/2 операций сравнения-обмена.

Пусть (a1,a2, ...,an) - последовательность, которую нужно отсортировать. На итерациях с нечетными итерациями элементы с нечетными индексами сравниваются с их правыми соседями, и если они не находятся в нужном порядке, они меняются местами (т.е. сравниваются пары (a1, a2), (a3, a4), ..., (an-1,an) и, при необходимости, обмениваются (принимая, что n четно). Точно так же в течение четной фазы элементы с четными индексами сравниваются с их правыми соседями (т.е. сравниваются пары (a2, a3), (a4, a5), ..., (an-2,an-1)), и если они находятся не в порядке сортировки, их меняют местами. После n фазы нечетно-четных перестановок последовательность отсортирована. Каждая фаза алгоритма (и нечетная, и четная) требует O(n) сравнений, а всего n фаз; таким образом, последовательная сложность алгоритма - O(n2).

Алгоритм2.Последовательный алгоритм "чет-нечетных перестановок"

```
ODD_EVEN(n) {
for (i=1; i<n; i++){
if (i%2==1) // нечетная итерация
for (j=0; j<n/2-1; j++)
compare_exchange(a2j+1,a2j+2);
if (i%2==0) // четная итерация
for (j=1; j<n/2-1; j++)
compare_exchange(a2j,a2j+1);
} }
```

4. Параллельная реализация алгоритма

Получение параллельного варианта алгоритма чет-нечетных перестановок не представляет каких-либо затрудений. Для каждой итерации алгоритма операции сравнения-обмена для всех пар элементов являются независимыми и выполняются одновременно. Рассмотрим случай "один элемент на процессор". Пусть p=n - число процессоров (а также число элементов, которые нужно сортировать). Для простоты изложения материала будем предполагать, что вычислительная система имеет топологию кольца. Пусть далее элементы ai i = 1, 2, ..., n, первоначально располагаются на процессорах pi, i = 1, 2, ..., n. В течение нечетной итерации каждый процессор, который имеет нечетный номер, производит сравнение-обмен своего элемента с элементом, находящимся на процессоре-соседе справа. Аналогично, в течение четной итерации каждый процессор с четным номером производит сравнение-обмен своего элемента с элементом правого соседа.

Алгоритм 3. Параллельный алгоритм "чет-нечетных перестановок" на n-процессорном кольце

```
ODD_EVEN_PAR(n){
id = GetProcId
for (i=1; i<n; i++){
if (i%2 == 1) // нечетная итерация
```

if (id%2 == 1) // нечетный номер процессора
compare_exchange_min(id+1); // сравнение-обмен с элементом на процессоре-соседе справа
else
compare_exchange_max(id-1); // сравнение-обмен с элементом на процессоре-соседе слева
if (i%2 == 0) // четная итерация
if(id%2 == 0) // четный номер процессора
compare_exchange_min(id+1); // сравнение-обмен с элементом на процессоре-соседе справа
else
compare_exchange_max(id-1); // сравнение-обмен с элементом на процессоре-соседе слева
} }

В течение каждой итерации алгоритма операции "сравнения-обмена" выполняются только между соседними процессорами. Это требует Θ(1) времени. Общее количество таких итераций - n; таким образом, параллельное время выполнения алгоритма - Θ (n).

Рассмотрим теперь случай, когда число процессоров меньше, чем длина сортируемой последовательности. Пусть p - число процессоров, n - длина сортируемой последовательности, где p<n. Первоначально каждый процессор получает блок элементов, который может быть упорядочен за время pn/log()))//((pnpnΘ каким-либо быстрым алгоритмом сортировки. После этого процессоры выполняют p (нечетных и четных) итераций, выполняя операцию "сравнить и разделить" (compare-split). Суть данной операции состоит в следующей последовательности действий:

− соседние по кольцу процессоры передают копии своих данных друг другу; в результате этих действий на каждом из пар соседних процессоров будет располагаться одинаковый набор двух блоков упорядочиваемых значений;

− далее на каждом процессоре имеющиеся блоки объединяются при помощи операции слияния;

− затем на каждом процессоре блок удвоенного размера разделяется на две части; после этого левый сосед процессорной пары оставляет первую половину элементов (с меньшими по значению элементами), а правый процессор пары - вторую (с большими значениями данных).

По окончании p итераций алгоритма исходная последовательность данных оказывается отсортированной.

5. Анализ эффективности

Оценим трудоемкость рассмотренного параллельного метода. Длительность операций передачи данных между процессорами полностью определяется топологией вычислительной сети. Если логически соседние процессоры, участвующие в выполнении операции "сравнить и разделить", являются близкими (например, для кольцевой топологии сети), общая коммуникационная сложность алгоритма.

пропорциональна количеству упорядоченных данных, т. е. n (объем пересылаемых на итерации данных определяется размером блока , количество итераций равно p). Тем самым, вычислительная трудоемкость алгоритма определяется выражением: n/p

Tp=Θ ((n/p)/log(n/p))+Θ(n)+Θ(n),

где первая часть соотношения учитывает сложность первоначальной сортировки блоков, а вторая величина задает общее количество операций для слияния блоков в ходе исполнения операций "сравнить и разделить" (слияние двух блоков требует 2(n/p) операций, всего выполняется p итераций сортировки). С учетом данной оценки показатели параллельного алгоритма имеют вид:

• для ускорения– $S_p = \frac{\Theta(n \log n)}{\Theta((n/) \log(n/p) + \Theta(n)}$

• для эффективности $E_p = \frac{1}{1 - \Theta(\log p / \log n) + \Theta(p / \log n)}$.

*Напомним, что под ускорением понимается отношение времени выполнения последовательной программы к времени выполнения параллельной программы. Под эффективностью понимают отношение ускорения к количеству используемых процессоров

Анализ выражений показывает, что если количество процессоров совпадает с числом сортируемых данных, эффективность использования процессоров падает с ростом n; получение асимптотически ненулевого значения показателя Ер может быть обеспечено при количестве процессоров, пропорциональном величине log n.

6. Описание программы

Файл bubble.h

```
#ifndef __BUBBLE_H
#define __BUBBLE_H
#include <mpi.h>
#include <stdlib.h>
#include <stdio.h>
struct PROCESS {  // данные процесса
int ProcRank; // ранг процесса
int CommSize; // общее число процессов
int *pProcData; // указатель на данные
int BlockSize; // количество значений
};
// генерация исходных данных
void InitData ( int **pData, int *pDataSize, PROCESS *pProcs, int *argc, char **argv[] );
// параллельная сортировка
void ParallelBubble ( int *pData, int DataSize, PROCESS *pProcs );
// завершение работы
void FreeData ( int *pData, int DataSize, PROCESS *pProcs ); //рассылка и прием
исходных данных
void BcastData ( int *pData, int DataSize, PROCESS *pProcs );
// сортировка данных
void SortLocalData ( PROCESS *pProcs );
// сбор отсортированных данных
void GatherData ( int *pData, PROCESS *pProcs ) ;
// локальная пузырьковая сортировка
void SortLocalData ( PROCESS *pProcs );
// выполнение операции "сравнить и разделить"
void SortLinkedData ( int **pTemp1, int **pTemp2, int BlockSize, int LinkedRank, int
**pMergeData, int Invert);
#endif
```

Файл bubble.cpp

```
#include "Bubble.h"
int main(int argc, char *argv[]) {
PROCESS Procs;
int *pData;
int DataSize;
// генерация исходных данных
InitData ( &pData, &DataSize, &Procs, &argc, &argv );
// параллельная сортировка
```

```
ParallelBubble ( pData, DataSize, &Procs );
// завершение работы
FreeData ( pData, DataSize, &Procs );
return 0;
}
//генерация исходных данных
void InitData ( int **pData, int *pDataSize, PROCESS *pProcs, int *argc, char **argv[] ) {
MPI_Init ( argc, argv );
MPI_Comm_rank ( MPI_COMM_WORLD, &pProcs->ProcRank );
MPI_Comm_size ( MPI_COMM_WORLD, &pProcs->CommSize );
if ( pProcs->ProcRank == 0 ) {
sscanf((*argv)[1], "%d", pDataSize);
if ( *pDataSize%pProcs->CommSize != 0 )
*pDataSize += pProcs->CommSize - ( *pDataSize % pProcs->CommSize );
*pData = new int[*pDataSize];
srand(100);
for(int i=0; i<(*pDataSize); i++) (*pData)[i] = rand();
}
}
// параллельная сортировка
void ParallelBubble ( int *pData, int DataSize, PROCESS *pProcs ) {
//рассылка и прием исходных данных
BcastData ( pData, DataSize, pProcs );
// сортировка данных
SortData ( pProcs );
// сбор отсортированных данных
GatherData ( pData, pProcs );
}
// завершение работы
void FreeData ( int *pData, int DataSize, PROCESS *pProcs ) {
if ( pProcs->ProcRank == 0) delete [] pData;
MPI_Finalize();
}
// рассылка и прием исходных данных
void BcastData ( int *pData, int DataSize, PROCESS *pProcs ) {
if ( pProcs->ProcRank == 0 ) { // рассылка блоков данных
pProcs->pProcData = pData;
pProcs->BlockSize = DataSize / pProcs->CommSize;
MPI_Bcast ( &pProcs->BlockSize, 1, MPI_INT, 0, MPI_COMM_WORLD );
pProcs->BlockSize, MPI_INT, pProcs->pProcData, pProcs->BlockSize, MPI_INT, 0,
MPI_COMM_WORLD );
}
else { // прием блоков данных
MPI_Bcast ( &pProcs->BlockSize, 1, MPI_INT, 0, MPI_COMM_WORLD );
pProcs->pProcData = new int[pProcs->BlockSize * 2];
MPI_Scatter ( &pData[(pProcs->ProcRank)*(pProcs->BlockSize)], pProcs->BlockSize,
MPI_INT, pProcs->pProcData, pProcs->BlockSize, MPI_INT, 0, MPI_COMM_WORLD );
}
}
// сортировка данных
void SortData ( PROCESS *pProcs ) {
// данные с соседнего процессора
int *pLinkedData = &(pProcs->pProcData[pProcs->BlockSize]);
int *pMergeData = new int[ 2*pProcs->BlockSize ]; // данные после слияния
```

```
int Offset; // сдвиг номера процесса
int Invert; // перестановка указателей
SortLocalData ( pProcs ); // локальная сортировка на процессоре
for ( int i=1; i<=pProcs->CommSize; i++ ) { // чет-нечетная перестановка
if ( (i%2) == 1) { // нечетная итерация
if ( (pProcs->ProcRank) % 2 == 1) { // нечетный процесс
if ( (pProcs->CommSize)-1 == pProcs->ProcRank ) continue;
Offset=1; Invert=0;
}
else { // четный процесс
if ( pProcs->ProcRank == 0 ) continue;
Offset=-1; Invert=1;
}
}
else { // четная итерация
if ( (pProcs->ProcRank) % 2 == 1) { // нечетный процесс
Offset=-1; Invert=1;
}
else { // четный процесс
if ( (pProcs->ProcRank) == (pProcs->CommSize)-1 ) continue;
Offset=1; Invert=0;
}
}
SortLinkedData ( &(pProcs->pProcData), &pLinkedData, pProcs->BlockSize,
pProcs->ProcRank + Offset, & pMergeData , Invert );
}
delete [] pMergeData;
MPI_Barrier ( MPI_COMM_WORLD );
}
// сбор отсортированных данных
void GatherData ( int * pData, PROCESS * pProcs ) {
MPI_Gather ( pProcs->pProcData, pProcs->BlockSize, MPI_INT, pData, pProcs->BlockSize,
MPI_INT, 0, MPI_COMM_WORLD );
}
// локальная пузырьковая сортировка
void SortLocalData PROCESS * pProcs ) {
int Tmp;
for ( int i=0; i<(pProcs->BlockSize)-1; i++ )
for ( int j=0; j<(pProcs->BlockSize)-1; j++) {
if ( pProcs->pProcData[j] > pProcs->pProcData[j+1] ) {
Tmp = pProcs->pProcData[j];
pProcs->pProcData[j] = pProcs->pProcData[j+1];
pProcs->pProcData[j+1] = Tmp;
}
}
}
// выполнение операции "сравнить и разделить"
void SortLinkedData ( int ** pTemp1, int ** pTemp2, int BlockSize, int LinkedRank, int
** pMergeData, int Invert) {
MPI_Status status;
// обмен данными между соседними процессорами
MPI_Sendrecv ( * pTemp1, BlockSize, MPI_INT, LinkedRank, 0, * pTemp2, BlockSize,
MPI_INT, LinkedRank, 0, MPI_COMM_WORLD, &status );
int i=0, j=0;
```

```
while ( (i<BlockSize) && (j<BlockSize) ) { // сортировка слиянием
if ( (*pTemp1)[i] < (*pTemp2)[j] ) { (*pMergeData)[i+j] = (*pTemp1)[i]; i++; }
else { (*pMergeData)[i+j] = (*pTemp2)[j]; j++; }
}
if ( i<BlockSize )
for ( int k=i; k<BlockSize; k++ ) (*pMergeData)[k+j] = (*pTemp1)[k];
if ( j<BlockSize )
for ( int k=j; k<BlockSize; k++ ) (*pMergeData)[k+i] = (*pTemp2)[k];
int *pTmp;
if (Invert) { // перестановка указателей и разделение последовательности
pTmp = *pTemp1; *pTemp2 = * pMergeData;
*pTemp1 = &( (*pMergeData)[BlockSize] );
}
else {
pTmp = *pTemp2; *pTemp1 = * pMergeData;
*pTemp2 = &( (*pMergeData)[BlockSize] );
}
*pMergeData = pTmp;
```

Контрольные вопросы

1. Для чего служат дескрипторы концов канала?
2. Какая функция создаёт новый канал?
3. Какие функции используются для чтения/записи в канал?
4. Почему каналом могут пользоваться только родственные процессы?
5. Что такое команда-фильтр?
6. Какой максимальный размер буфера памяти отводится под канал в той системе, в которой Вы работаете ?
7. Что произойдет, если функции popen() передать имя несуществующей команды?

Порядок выполнения работы

1. Изучите теоретические сведения.
2. Изучите примеры программ, приведенных в теоретической части.
3. На основе примеров напишите программу
4. Протестируйте разработанные Вами программы.
5. Получите распечатки примеров работы программ.
6. Оформите отчет.

Задание

1. Напишите последовательную программу чет-нечетеной сортировки.
2. Напишите программу чет-нечетной сортировки для 4 параллельных процессов:

Управляющий процессор (процессор 0) выполняет:

- ввод длины сортируемой последовательности и ее генерацию случайным образом;
- разбиение последовательности на части и пересылку на соответствующие процессоры блоков последовательности;
- выполнение итераций параллельной сортировки (эта часть действий является общей для всех процессоров и детально описана в алгоритме работы функциональных процессоров);
- сбор отсортированных частей от всех процессоров;
- освобождение занимаемой памяти.

Функциональные процессоры выполняют:

− прием от процессора 0 соответствующих элементов;

− внутреннюю сортировку при помощи алгоритма пузырьковой сортировки на каждом процессоре;

− реализацию параллельной чет-нечетной перестановки;

− обмен данными между соседними процессорами;

− объединение слиянием упорядоченных блоков;

− разделение объединенного блока (процессор с меньшим номером сохраняет блок с меньшими элементами, процессор с большим номером – блок с большими элементами);

− передача результата (отсортированного блока данных после выполнения всех итераций алгоритма) на управляющий процессор.

Форма отчетности:

Отчет по работе содержит:

1. Наименование лабораторной работы;
2. Разработанную программу;
3. Результаты её тестирования;
4. Выводы по работе.

Рекомендации по выполнению заданий и подготовке к лабораторной работе

При подготовка к лабораторной работе необходимо изучить литературу, указанную ниже:

Основная литература

1. Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2015. - 640 с. - (Учебник для вузов. Стандарт третьего поколения) .

2. Лупин, С. А. Технологии параллельного программирования : учебное пособие / С. А. Лупин, М. А. Посыпкин. - Москва : ИНФРА-М, 2011. - 208 с.

9.2. Методические указания по выполнению контрольной работы

Контрольная работа представляет собой способ проверки знаний студента, его умений и предполагают письменные ответы на поставленные вопросы, либо самостоятельное выполнение практических заданий. Подготовка к контрольной работе состоит в ответственном выполнении всех домашних заданий по дисциплине и самостоятельной проработке основной и дополнительной литературы.

Целью контрольной работы является приобретение навыков самостоятельной работы с литературой, закрепление умений работы со средой программирования, формирование навыков оценки результатов собственной деятельности.

Выполнения контрольной работы предполагает:

- анализ поставленных задач и выбор методов их решения;
- реализацию решения поставленных задач;
- проверку и анализ полученных результатов;
- оформление отчета.

Отчет по контрольной работе оформляется в печатном виде и содержит:

- формулировку заданий;
- описание их решений;
- полученные результаты;
-выводы.

10. ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, ИСПОЛЬЗУЕМЫХ ПРИ ОСУЩЕСТВЛЕНИИ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

1. Microsoft Imagine Premium: Microsoft Windows Professional 7;
2. Microsoft Office 2007 Russian Academic OPEN No Level;
3. Антивирусное программное обеспечение Kaspersky Security;
4 ОС Linux;
5.GNU gcc

11. ОПИСАНИЕ МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЙ БАЗЫ, НЕОБХОДИМОЙ ДЛЯ ОСУЩЕСТВЛЕНИЯ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

Вид занятия	***Наименование аудитории***	***Перечень основного оборудования***	***№ ЛР***
1	**2**	**3**	**4**
Лк	Лекционная аудитория	-	-
ЛР	Лаборатория параллельных вычислений	Персональные компьютеры i5-2500/H67/4Gb/500Gb (монитор TFT19 Samsung E1920NR); интерактивная доска Smart Board X885ix со встроенным проектором UX60	№ 1-6
кр	Ч31	Оборудование 10 ПК i5-2500/H67/4Gb(монитор TFT19 Samsung); принтер HP LaserJet P2055D	-
СР	Ч31	Оборудование 10 ПК i5-2500/H67/4Gb(монитор TFT19 Samsung); принтер HP LaserJet P2055D	-

Приложение 1

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

1. Описание фонда оценочных средств (паспорт)

№ компетенции	Элемент компетенции	Раздел	Тема	ФОС
ОПК-3	Способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей	**1.** Основные компоненты параллельной вычислительной системы	1.1. Архитектура оп вычислительных систем	Вопрос к зачету
			1.2. Модели вычислительных процессов и систем	Вопрос к зачету
			1.3. Оценки производительности	Вопрос к зачету, задание контрольной работы
ОПК-4	Способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности		1.4. Построение и анализ информационного графа	Вопрос к зачету, задание контрольной работы
		2. Параллельные алгоритмы распространенных задач	2.1 Обработка линейной последовательности	Вопрос к зачету
ПК-9	Способность составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы		2.2 Умножение матрицы на вектор	Вопрос к зачету
			2.3. Интегрирование	Вопрос к зачету, задание контрольной работы
			2.4. Сортировка	Вопрос к зачету

2. Вопросы к зачету

№	Компетенции		ВОПРОСЫ К ЗАЧЕТУ	№ и наименование раздела
п/п	Код	Определение		
1	2	3	4	5
1.	ОПК-3	Способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей	1. Основные понятия параллельного программирования. Классификация вычислительных систем.	**1**. Основные компоненты параллельной вычислительной системы
			2. Обзор технологий параллельных вычислений.	**1**. Основные компоненты параллельной вычислительной системы
2.	ОПК-4	Способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности	3. Граф алгоритма «операции -операнды»..	**1**. Основные компоненты параллельной вычислительной системы
			4. Законы Амдала.	**1**. Основные компоненты параллельной вычислительной системы
			5. Классификация алгоритмов по типу параллелизма..	**1**. Основные компоненты параллельной вычислительной системы
			6. Декомпозиция в задачах с параллелизмом по данным.	**1**. Основные компоненты параллельной вычислительной системы
			7. Вычисление частных сумм последовательности числовых значений..	**2.** Параллельные алгоритмы распространенных задач
			8. Каскадная схема суммирования. Модифицированная каскадная схема.	**2.** Параллельные алгоритмы распространенных задач
4.	ПК-9	Способность составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	9. Умножение матрицы на вектор при разделении данных по строкам.	**2.** Параллельные алгоритмы распространенных задач
			10. Умножение матрицы на вектор при блочном разделении данных.	**2.** Параллельные алгоритмы распространенных задач

3. Описание показателей и критериев оценивания компетенций

Показатели	Оценка	Критерии
знать: ОПК-3 - основные параллельные алгоритмы ОПК-4 - основы информационно-коммуникационных технологий и информационной безопасности; ПК-9 - принципы планирования работы по осуществлению разработки параллельных программ ; **уметь:** ОПК-3 - разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; ОПК-4 - использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности ПК-9 - планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; **владеть:** ОПК-3 - приемами построения алгоритмических и программных решений в области параллельного программирования	**зачтено**	Демонстрирует владение не менее, чем 5 показателями компетенций, а именно: -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет планировать ресурсы, необходимые для построения

ОПК-4 – навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования; ПК-9 – навыками планирования, проектирования и оценки параллельных вычислительных систем.		параллельных вычислительных систем, оценивать результаты работы параллельной программы; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет приемами построения алгоритмических и программных решений в области параллельного программирования ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с

		применением информационно-коммуникационных технологий параллельного программирования ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности; -умеет планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; -владеет приемами построения алгоритмических и программных решений в области параллельного программирования ; -владеет навыками планирования, проектирования и оценки параллельных вычислительных систем. ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению

		разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности; -умеет планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; -владеет навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования ; -владеет навыками планирования, проектирования и оценки параллельных вычислительных систем. ; или -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ;

		-знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования
	незачтено	Владеет менее чем половиной параметров компетенций.

4. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и опыта деятельности

Дисциплина Параллельное программирование направлена на ознакомление обучающихся с основами разработки системных программ, способами использования и управления системными ресурсами; на получение теоретических знаний и практических навыков разработки программ, формулирования и решения проблем из различных областей наук, а также осуществления поиска, хранения, обработки и анализа информации из различных источников и представления ее в соответствующем виде и для их дальнейшего использования в практической деятельности.

Изучение дисциплины Параллельное программирование предусматривает:

- лекции,
- лабораторные работы;
- контрольную работу;
- зачет;
- самостоятельную работу.

В ходе освоения раздела 1 «Основные компоненты параллельной вычислительной системы» обучающиеся должны изучить принципы устройства и функционирования операционных систем, их структуру, способы обращения к компонентам системы, научиться получать информацию о процессах в системе, атрибутах файлов, использовать системные

ресурсы в прикладных и системных программах.

В ходе освоения раздела 2 «Параллельные алгоритмы распространенных задач» обучающиеся осваивают особенности управления процессами в системе, использования механизмов передачи сообщений, отправку и перехват сигналов.

Обучающимся небходимо овладеть навыками и умениями применения изученных методов для разработки и реализации профессионально ориентированных проектов в последующей профессиональной деятельности.

Овладение ключевыми понятиями является основой усвоения учебного материала по дисциплине.

При подготовке к зачету особое внимание необходимо уделить рекомендациям и замечаниям преподавателей, ведущих аудиторные занятия по дисциплине

В процессе проведения лабораторных занятий происходит закрепление знаний, формирование умений и навыков применения различных методов решения стандартных ситуаций.

Самостоятельную работу необходимо начинать с чтения лекций и учебников.

В процессе консультации с преподавателем обучающийся выясняет наличие пробелов в знаниях и способах решения разных ситуаций.

Работа с литературой является важнейшим элементом в получении знаний по дисциплине. Прежде всего, необходимо воспользоваться списком рекомендуемой по данной дисциплине литературой. Дополнительные сведения по изучаемым темам можно найти в периодической печати и Интернете.

Предусмотрено проведение аудиторных занятий в виде разнообразных тренингов и ситуаций общения в сочетании с внеаудиторной работой.

Приложение 2

АННОТАЦИЯ
рабочей программы дисциплины
Параллельное программирование

1. Цель и задачи дисциплины

Целью изучения дисциплины является: ознакомление обучающихся с различными методами, приемами разработки параллельных алгоритмов, проектированием параллельных вычислительных систем.

Задачами дисциплины являются

- обучение методам компьютерного формализованного представления знаний и реализации выводов для последующей выработки и принятия человеком вариантов принимаемого решения;
- формирование умения и навыков самостоятельного исследования и решения различного рода задач путем применения средств функционального программирования совместно с другими видами программного обеспечения;
- формирование и развитие умений и навыков, позволяющих применять современные математические методы и программное обеспечение для решения задач науки и техники.
- .

2. Структура дисциплины

2.1 Распределение трудоемкости по отдельным видам учебных занятий, включая самостоятельную работу: Лк.-24 час., ЛР- 24 час.; СР - 60 час.

Общая трудоемкость дисциплины составляет 108 часов, 3 зачетных единиц.

2.2 Основные разделы дисциплины:

1 – Основные компоненты параллельной вычислительной системы;
2 – Параллельные алгоритмы распространенных задач

3. Планируемые результаты обучения (перечень компетенций)

Процесс изучения дисциплины направлен на формирование следующих компетенций:

ОПК-3 Способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям.

ПК-1 Способность собирать, обрабатывать и интерпретировать данные современных научных исследований, необходимые для формирования выводов по соответствующим научным исследованиям .

ПК-7 Способность к разработке и применению алгоритмических и программных решений в области системного и прикладного программного обеспечения .

4. Вид промежуточной аттестации: зачет.

Приложение 3

Протокол о дополнениях и изменениях в рабочей программе
на 20___-20___ учебный год

1. В рабочую программу по дисциплине вносятся следующие дополнения:
_____
_____

2. В рабочую программу по дисциплине вносятся следующие изменения:

_____
_____
_____

Протокол заседания кафедры №____ от «___» _____ 20 ____ г.,
(разработчик)

Заведующий кафедрой _____ _____
(подпись) *(Ф.И.О.)*

Приложение 4

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ ПО ДИСЦИПЛИНЕ

1. Описание фонда оценочных средств (паспорт)

№ компетенции	Элемент компетенции	Раздел	Тема	ФОС
ОПК-3	Способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей	**1.** Основные компоненты параллельной вычислительной системы	1.2. Модели вычислительных процессов и систем	ЛР №1
			1.3. Оценки производительности	ЛР№ 2, кр, задание 1
ОПК-4	Способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности		1.4. Построение и анализ информационного графа	ЛР№ 3, кр, задание 2
			2.1 Обработка линейной последовательности	ЛР№4 кр, задание 3
		2. Параллельные алгоритмы распространенных задач	2.2 Умножение матрицы на вектор	ЛР№5 кр, задание 4
ПК-9	Способность составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы		2.3. Интегрирование	ЛР№5
			2.4. Сортировка	ЛР№6

3. Описание показателей и критериев оценивания компетенций

Показатели	Оценка	Критерии
знать: ОПК-3 - основные параллельные алгоритмы ОПК-4 - основы информационно-	**зачтено**	Демонстрирует владение не менее, чем 5 показателями компетенций, а именно: -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности;

коммуникационных технологий и информационной безопасности; ПК-9 - принципы планирования работы по осуществлению разработки параллельных программ ; **уметь:** ОПК-3 - разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; ОПК-4 - использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности ПК-9 - планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; **владеть:** ОПК-3 - приемами построения алгоритмических и программных решений в области параллельного программирования ОПК-4 – навыками решения стандартных задач		-знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -умеет планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет приемами построения алгоритмических и программных решений в области параллельного программирования ; или

профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования; ПК-9 – навыками планирования, проектирования и оценки параллельных вычислительных систем.		-знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет навыками решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий параллельного программирования ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет навыками планирования, проектирования и оценки параллельных вычислительных систем. ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет использовать информационно-коммуникационные технологи для решения задач

		профессиональной деятельности; -умеет планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы; или - или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет разрабатывать параллельные алгоритмы решения задач из области системного и прикладного программирования, математических, информационных и имитационных моделей ; -владеет навыками планирования, проектирования и оценки параллельных вычислительных систем. ; или -знает основные параллельные алгоритмы; -знает основы информационно-коммуникационных технологий и информационной безопасности; -знает принципы планирования работы по осуществлению разработки параллельных программ ; -умеет использовать информационно-коммуникационные технологи для решения задач профессиональной деятельности; -умеет планировать ресурсы, необходимые для построения параллельных вычислительных систем, оценивать результаты работы параллельной программы.
	незачтено	Владеет менее чем половиной параметров компетенций.

Программа составлена в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению подготовки 01.03.02 Прикладная математика и информатика от «12» марта 2015 г. № 228

для набора 2015 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «13 » июля 2015 г. № 475

для набора 2016 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «06» июня 2016г. № 429

для набора 2017 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «6» марта 2017г. № 125

Программу составил:

Ратинская Е.В., ст. препод. каф. МиФ _____

Рабочая программа рассмотрена и утверждена на заседании кафедры МиФ

от «21» ноября 2018 г., протокол № 3

И.о. зав.выпускающей кафедрой _____ О.И.Медведева

СОГЛАСОВАНО:

И.о. зав.выпускающей кафедрой _____ О.И Медведева.

Директор библиотеки _____ Т.Ф.Сотник

Рабочая программа одобрена методической комиссией ЕН факультета

от «20 » декабря 2018 г., протокол № 4

Председатель методической комиссии факультета _____ М.А. Варданян

СОГЛАСОВАНО:

Начальник
учебно-методического управления _____ Г.П. Нежевец

Регистрационный №_____