МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

ОШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

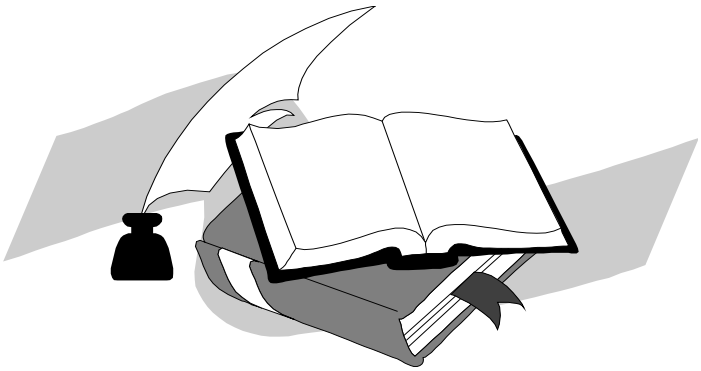
Кафедра «АВТОМАТИЗИРОВАННЫХ СИСТЕМ и ЦИФРОВЫХ ТЕХНОЛОГИЙ»

«Согласовано»
Председателем Методсовета
факультета МИТ
к.п.н., доцент: Зулпукарова
«_3_» _____ 2020 г.

«Утверждено»
на заседании кафедры АСЦТ
прот. № 1 от «3» сентября 2020 г.
Зав. каф. ИТАС, доцент:
Молдояров У.Д.

УЧЕБНО МЕТОДИЧЕСКИЙ КОМПЛЕКС

По дисциплине “Параллельные вычисления”



Составитель, доцент каф. АСЦТ: Атырова Р. С.

2020-2021 учебный год

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
ОШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ И ЦИФРОВЫЕ ТЕХНОЛОГИИ

«*Утверждено»*
на заседании кафедры
Прот.№_1_ от «_28 » _августа 2020г.
Зав. каф. АСЦТ, доцент Молдояров У.

«Утверждено»
Председатель УМС
факультета МИТ «3 » сентября 2020г.
доцент Зулпукарова З.

РАБОЧАЯ ПРОГРАММА

по дисциплине: «**Параллельные вычисления**»

для магистрантов очного отделения, обучающихся по направлению:

7100100 «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»

(шифр и наименование направления)

Сетка часов по учебному плану *(пример)*

Наим. дисцип-лины	Все-го	Ауд. зан.	Аудит.зан.		СРС	Отчетность	
			Лек-ции	Лаб. зан.		1-сем	1сем
Дисцип-лина	120ч (4кр)	60ч (2кр)	30ч	30ч	60ч	РК -2	Экз
1-сем	120	48	24	24	72	РК- 2	Экз.

Рабочая программа разработана на основании составлена на основе Республиканского ГОС ВПО (14.05.12., № гос. рег. 116) в соответствии с требованиями к структуре и результатам освоения основных образовательных программ магистранта по «профессиональному» циклу (КПВ) по направлению подготовки 710100 «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА» а также задачами, стоящими перед Ошским государственным университетом по реализации Программы развития ОшГУ.

(Для магистрантов)

Составитель: к.т.н., доцент _____Атырова Р.

2020-2021 – учебный год

1. Цели освоения дисциплины

Освоение системы понятий параллелизма, выработка навыков использования параллельных компьютерных систем, параллельного вычисления. Формирование у магитрантов представления о месте параллелизма в современном программировании, о природе трудностей в параллельном программировании и способах их преодоления. Изучение языков и сред параллельного вычисления, освоение методов разработки, оптимизации и организации выполнения на компьютерных системах (КС) параллельных программ.

По завершению освоения данной дисциплины магистрант:

- получает знания о состоянии, технических характеристиках и применении сильно связанных КС (кластеров) и распределенных КС (локальных и глобальных компьютерных ресурсов) для решения сложных задач;
- получает знания о языках и средах параллельного вычисления различной ориентации;
- получает знания о методах и средствах, предназначенных для организации выполнения параллельных программ на КС и управления параллельными процессами;

Код РОоп и его формулировка	**Код компетенции ООП и его формулировка**	**Код РО дисциплины (РОд) и его формулировка**
РО-2:_____ _____	ОК-1: способен глубоко понимать и критически оценивать новейшие теории, методы и способы, использовать междисциплинарный подход и интегрировать достижения различных наук для приобретения новых знаний;	Знает и понимает: аппаратные и программные аспекты реализации параллелизма; формальные модели параллельного программирования; принципы взаимодействия асинхронных программирования; (ОК1) - **Владеть:** теоретическими знаниями в области организации взаимодействующих процессов, средствами параллельного программирования MPI, OpenMP, Threads, способами проверки правильности параллельных программ
		-
РО-3: _____ _____	ПК-2: способен анализировать и систематизировать научно-техническую информацию по теме исследования, выбирать методики и средства решения задач;	**Уметь:** разрабатывать параллельные алгоритмы и программы для решения разного класса задач на компьютерах с распределенной памятью и общей памятью; совместно программировать на MPI+OpenMP, MPI+Threads; Использовать библиотеки для параллельных программ; использовать профилировщики и отладчики для параллельных программ; ставить и решать задачи, возникающие в процессе конструирования параллельных программ и эксплуатации системных программных средств. -
	ИК-4: способен делать выводы, четко и ясно объяснять	Уметь: реализовывать параллельные программы на локальном компьютере и в сети рабочих станций и уметь

	(транслировать) материал на основе приобретенных знаний (как специалисту, так и не специалисту), способен к дальнейшему самостоятельному обучению.	изменять при необходимости вид и характер своей профессиональной деятельности Владеть: технологиями создания многопоточных приложений для систем с общей памятью; владеть инструментальными средствами параллельных вычислительных и суперкомпьютерных систем

- приобретает способность разрабатывать сложные параллельные программы для реальных задач и организовывать их эффективное выполнение на КС;

приобретает способность выполнять проектирование программных средств для повышения эффективности процессов проектирования параллельных программ и организации их выполнения на КС.

2. Результаты обучения и компетенции, формируемые в процессе изучения дисциплины_____

В результате изучения дисциплины студент достигнет следующих **результатов обучения(РОд),**соответствующих ожидаемым ***результатам освоения образовательной программы*** **(РООп)** и заданным для дисциплины ***компетенциям:***

I модуль						
№	Лекции		Лабораторные работы		СРС	
	Часы	Балл	Часы	Балл	Часы	Балл
ТК-1						
		(№1 тема+№2 тема+...+№8 тема)/8		(№1 тема+№2 тема+...+№8 тема)/8		(№1 тема+№2 тема+...+№8 тема)/8
№1 тема	1	30	1	30	2	30
№2 тема	1	30	1	30	2	30
№3 тема	1	30	1	30	2	30
№4 тема	1	30	1	30	2	30
№5 тема	1	30	1	30	2	30
№6 тема	1	30	1	30	2	30
№7 тема	1	30	1	30	2	30
№8 тема	1	30	1	30	2	30
Всего	8	30	8	30	16	30
ТК-2						

		(№1 тема+№2 тема+...+№8 тема)/8		(№1 тема+№2 тема+...+№8 тема)/8		(№1 тема+№2 тема+...+№8тема)/8
№1 тема	1	30	1	30	2	30
№2 тема	1	30	1	30	2	30
№3 тема	1	30	1	30	2	30
№4 тема	1	30	1	30	2	30
№5 тема	1	30	1	30	2	30
№6 тема	1	30	1	30	2	30
№7 тема	1	30	1	30	2	30
№8 тема	1	30	1	30	2	30
Всего	8	30	8	30	16	30
Итого	16	(ТК-1+ ТК-2)/2 30	16	(ТК-1+ ТК-2)/2 30	32	(ТК-1+ ТК-2)/2 30

3. Место дисциплины в структуре ООП

Курс относится к дисциплинам вузовского компонента, утвержден Ученым советом факультета __МИТ___ Протокол №_8_ от 20 апреля_2019г

До этого утверждено на заседании кафедры, затем – на методсовете факультета.

4. Карта компетенций дисциплины в разрезе тем (разделов)

Разделы, № и название темы	**Кол-во час.**	**Компетенции**					
		ОК-1	**ОК-4**	**ИК-1**	**ПК-11**	**ПК-14**	**Σ общее кол-во комп-ций**
Раздел 1							
Тема 1.___		+		+			2
Тема 2.___			+		+	+	3
и т.д.							
Итого:							

5. Технологическая карта дисциплины

Модули	Всего	Ауд зан	Лекции		Лабор.		СРС		РК	ИК	Баллы
			часы	балл	часы	балл	часы	балл			
I	60	24	12	5	12	10	36	5	10б		30
II	60	24	12	5	12	10	36	5	10б		30

ИК										40б	40

6. Карта накопления баллов по дисциплине

В ОшГУ усвоение дисциплины оценивается в 100 баллах, из них 60 баллов дается для текущего и промежуточного (рубежного) контроля, 40 баллов – для итогового контроля.

I модуль				
Текущий контроль			РК1	М1=(Лек+Лаб+СРС+РК1)/4
Лек.	Лаб.	СРС		
30	30	30	30	30

II модуль						
№	Лекции		Лабораторные работы		СРС	
	Часы	Балл	Часы	Балл	Часы	Балл
ТК-1						
		(№1 тема+№2 тема+...+№9 тема)/9		(№1 тема+№2 тема+...+№9 тема)/9		(№1 тема+№2 тема+...+№9 тема)/9
№1 тема	1	30	1	30	2	30
№2 тема	1	30	1	30	2	30
№3 тема	1	30	1	30	2	30
№4 тема	1	30	1	30	2	30
№5 тема	1	30	1	30	2	30
№6 тема	1	30	1	30	2	30
№7 тема	1	30	1	30	2	30
Всего	7	30	7	30	14	30
ТК-2						
		(№1 тема+№2 тема+...+№9 тема)/10		(№1 тема+№2 тема+...+№9 тема)/10		(№1 тема+№2 тема+...+№9 тема)/10
№1 тема	1	30	1	30	2	30
№2 тема	1	30	1	30	2	30
№3 тема	1	30	1	30	2	30
№4 тема	1	30	1	30	2	30
№5 тема	1	30	1	30	2	30

№6 тема	1	30	1	30	2	30
№7 тема	1	30	1	30	2	30
Всего	7	30	7	30	14	30
Итого	14	(ТК-1+ ТК-2)/2 30	14	(ТК-1+ ТК-2)/2 30	28	(ТК-1+ ТК-2)/2 30

Основная задача Вашей работы на лекциях – понимание представляемого преподавателем материала с запоминанием его основных положений, ведение достаточно полного конспекта, позволяющего успешно подготовиться к лабораторным занятиям и последующему экзамену.
Максимальный балл (МБ) семестра – 19+38+19+20+8=100 баллов (100 %)
Недопустимо:

- Опоздание и уход с занятий;
- Пользование сотовыми телефонами во время занятий;
- Обман и плагиат;
- Несвоевременная сдача заданий.

В течение 1 учебного семестра будут проведены 2 рубежных контроля (РК1 и РК2) в форме тестов на сайте www.avn.osu,kg по различным темам, РК1 – 8 неделя, РК2 – 15 неделя. Дата и время Вам будут сообщены дополнительно.
Итоговый контроль после I семестра – экзамен в устной форме.
Внимание!
К экзамену не допускаются студенты, не набравшие рейтинг допуска (60% от МБ I семестра).

7. Тематический план распределения часов по видам занятий

№	Наименование разделов, тем дисциплины	Всего	Ауд. занят.			СРС	Обр. технологии	Оценоч. средства
			Лекции	№ декции по темам	Лабораторн. занятия			
	Семестр 1							
	Модуль 1							
1.	Введение. Понятие вычислимой функции	4	2	1		6	ПК интер доска	Презент тест
2.	Задача конструирования параллельной программы	14	2	2-3	4	10	ПК интер доска	През.

3.	Взаимодействующие процессы	20	6	4-6	6	12	ПК интер доска	През
	Модуль 2							
4.	Программирование взаимодействующих процессов	28	6	7-10	6	14	Презент. интер доска	Тест
5.	Организация параллельных вычислений в крупноблочных иерархических мультикомпьютерах	26	6	11-14	6	16	ПК, РМ	Прак работы
6.	Отображение алгоритмов на ресурсы мультикомпьютера Заключение.	16	2	15-16	2	14	ПК, презент.	раб на ПК
	Итого:	108	24		24	72		

8. Программа дисциплины

(Указываются разделы, темы и содержание тем дисциплины.)

9. Цели и результаты обучения по темам дисциплины

Тема 1._____			
Компе-тенции	**ОК10:_____** **СЛК1:_____**		
РОд	Знает_____ Умеет_____ Владеет_____		
Цели темы	*(сформулировать на основе компетенций и* РОд*)*		
РО темы (РОт)	**Лекц.**	2ч	Знает и понимает_____
	Сем.	2ч	Умеет_____ Владеет_____
	СРС	4ч	Умеет_____ Владеет_____
Тема 2. и т.д.			

10. Календарно-тематический план по видам занятий

Календарно-тематический план оформляется в виде таблицы.

Предлагаются ***2 варианта*** *оформления данного раздела рабочей программы на выбор преподавателя.*

Вариант 1.

Этот пункт может быть представлен в виде ***таблицы,*** *где для* ***каждой темы*** *планы лекций, семинарских (практических, лабораторных) занятий и задания для СРС пишутся сразу напротив темы - последовательно, по горизонтали.*

Такая форма представляется более компактной и удобной для использования, поскольку по конкретной теме *на одной странице по горизонтали располагаются планы лекций, семинаров (практических, лабораторных занятий) и задания СРС.*

10.1. Календарно-тематический план распределения часов дисциплины по видам занятий

№ и назв. темы	Формы компетенции	**Лекции**	**Практические (семинарские) занятия**	**СРС**			Лит-ра	Сроки сдачи
		(часы), вопросы ,методы	(часы), вопросы, методы	Задания	ч	Форм .контр		
		Модуль 1						
Тема1. _____ _____	ОК10 СЛК1	*План лекции*: (2ч) 1_____ 2._____ *Контрольные вопросы:* *1.*_____ *2.*____ ..... Методы: ЛБ, ЛВ, МШ	*План (*2ч) 1. Раскройте….. 2. Покажите особенности …. 3. Сравните ……. Методы: МГ, През, Д Формы контроля: През, Д	1. Изобразите схематически…… и обьясните ……	4	Схема 1	1,2 4,5 8, 13 16	2-я нед
Тема2 ит.*д*….					4	Т		3-я нед
Мод. 1:		**Лекций: __ ч**	**Семинаров: ___ ч**	**СРС:**	**_ _ ч**			**7 нед**
		Модуль 2						

Тема8 _____	ОК12 ПК 11	*План лекции:* (2ч) 1.____ 2.____ . *Контрольные вопросы* *1.*_____ *2.*_____ ….... Методы: ЛВ,През.	*План* 1. Представьте …. 2. Дайте оценку ……. 3. Проанализируйте ……. Методы: ДИ, През,МШ. Формы контроля: ДИ, През,МШ	1. Составьте … и дайте прогноз….	4	Анал. справка 1	3,5 8, 10 12 14	9-я нед
Тема9 и т.д.	ПК12				4	Контр раб.		
Мод.2:		**Лекций: ___ч**	**Семинаров: __ ч**	**СРС:**	**_ ч**			**8 нед**
ВСЕГО:		**Лекций: ___ час.**	**Семинаров: ___ час.**	**СРС:**	**_ _ ч**			**15 нед**

Вариант 2.

10.2.1. Лекции

№ и название темы	№ Лекции, комп.	Наименование изучаемых вопросов	К-во час	Литра	Исп. обр. зов-техн	Нед
1	2	3	4	6	7	8
Модуль 1						
1.1. Ведение. Основные этапы параллельного программирования. Общий обзор	ОК10 ПК-2	*План лекции*: 1._____ 2._____ *Контрольные вопросы:* *1.*_____ *2.*_____ *3.*_____	1	1,2 4,5, 8,13	ЛБ, ЛВ, МШ	1-я
1.2. Предварительные понятия.	ИК-4		1	1,2 4,5,	ЛБ, ЛВ, МШ	1-я
1.3. Понятие рекурсивной функции.	ОК10		1	1,2 4,5,	ЛБ, ЛВ, МШ	2-я

1.4. Детерминант вычислимой функции.	ПК-2		1	1,2 4,5,	ЛБ, ЛВ, МШ	2-я
2.1. Представление алгоритма.	ИК-4		1	1,2 4,5,	ЛБ, ЛВ, МШ	3-я
2.2. Требования к представлению параллельного алгоритма.	ОК10		1	1,2 4,5,	ЛБ, ЛВ, МШ	3-я
2.3. Простейшая программа, реализующая алгоритм.	ПК-2		1	1,2 4,5,	ЛБ, ЛВ, МШ	4-я
2.4. Сравнительная непроцедурность языков программирования.	ИК-4		1	1,2 4,5,	ЛБ, ЛВ, МШ	4-я
3.1. Последовательные процессы.	ОК10		1	1,2 4,5,	ЛБ, ЛВ, МШ	5-я
3.2. Выполнение системы процессов.	ПК-2		1	1,2 4,5,	ЛБ, ЛВ, МШ	5-я
3.3. Сети Петри.	ИК-4		1	1,2 4,5,	ЛБ, ЛВ, МШ	5-я
3.4. Задача взаимного исключения.	ОК10		1	1,2 4,5,	ЛБ, ЛВ, МШ	6-я
3.5. Дедлоки.	ПК-2		1	1,2 4,5,	ЛБ, ЛВ, МШ	6-я
3.6. Задача о пяти обедающих философах.	ИК-4		1	1,2 4,5,	ЛБ, ЛВ, МШ	6-я
3.7. Задача производитель/потребитель.	ОК10		1	1,2 4,5,	ЛБ, ЛВ, МШ	7-я
3.8. Реализация управления взаимодействующими процессами.	ПК-2		1	1,2 4,5,	ЛБ, ЛВ, МШ	7-я
Итого модуль 1	***7лек***		***14 ч***			***7 нед***
Модуль 2						

4.1. Асинхронное программирование.	ИК-4	*План лекции:* 1._____. 2_____.. *Контрольные вопросы:* 1._____ 2._____	2	6,7, 11, 15	ПЛ,. МШ Д	8-я
4.2. Message passing interface (MPI).	ОК10		2	6,7, 11, 15	ПЛ,. МШ Д	8-я
5.1. Иерархические мультикомпьютеры.	ПК-2		2	6,7, 11, 15	ПЛ,. МШ Д	9-я
5.2. Линейные алгоритмы.	ИК-4		2	6,7, 11, 15	ПЛ,. МШ Д	9-я
5.3. Децинтрализованное управление.	ОК10		2	6,7, 11, 15	ПЛ,. МШ Д	10-я
5.4. Централизованное управление.	ПК-2		2	6,7, 11, 15	ПЛ,. МШ Д	10-я
5.5. Топология структуры связей.	ИК-4		2	6,7, 11, 15	ПЛ,. МШ Д	11-я
6.1. Статическая постановка задачи.	ОК10		2	6,7, 11, 15	ПЛ,. МШ Д	11-я
6.2. Идеи параллельной реализации PIC.	ПК-2		2	6,7, 11, 15	ПЛ,. МШ Д	12-я
6.3. Распараллеливание метода частиц	ИК-4		2	6,7, 11, 15	ПЛ,. МШ Д	13-я
6.4. Централизованные алгоритмы балансировки загрузки	ОК10		2	6,7, 11, 15	ПЛ,. МШ Д	13-я
6.5. Децентрализованные алгоритмы динамической балансировки загрузки.	ПК-2		2	6,7, 11, 15	ПЛ,. МШ Д	14-я
6.6. Динамическое отображение алгоритма на ресурсы мультикомпьютера.	ИК-4		2	6,7, 11, 15	ПЛ,. МШ Д	14-я

6.7. Общие принципы сборочной технологии параллельного программирования.	ОК10		2	6,7, 11, 15	ПЛ,. МШ Д	15-я
6.8. Заключение. Тенденции и перспективы развития параллельного программирования.	ПК-2		2	6,7, 11, 15	ПЛ,. МШ Д	15-я
Итого модуль 2	**8 *лекц***		***16 ч***			***16 нед***
ВСЕГО	**15 лек.**		**30 ч**			**15 нед**

10.2.2. Семинарские занятия

№ и название темы	№ Сем., комп.	Изучаемые вопросы и задания	К-во час	Лит-ра	Испобр техн	Нед
1	2	3	4		7	8
Модуль 1						
1.1. Распределенные вычисления. Система параллельного программирования MPI - общее представление. Общая характеристика технологии MPI: парные взаимодействия процессов.	1 ОК10 ПК 1	*План* 1. Раскройте _____ 2. Покажите_____ 3. Сравните_____ *Форма контроля: През,Д,МШ*	2	1,2 4,5, 8,13	МГ През Д МШ	1-я
2.1. Распределенные вычисления. Коллективные взаимодействия процессов; глобальные операции редукции; выделенные группы процессов, объединяемые коммуникационными связями; оценка времяемкости операций передачи данных на кластерных системах. Параллельные алгоритмы для решения простых типовых задач на системах с распределенной памятью. Редуцированные MPI - операции.	ОК1		2	1,2 4,5, 8,13	МГ През Д МШ	2-я
2.2. Вычисления над общим полем памяти. Система параллельного программирования OpenMP - общее представление. Общая характеристика технологии OpenMP: потоки; области параллельных вычислений; задания работ.	ПК2		2	1,2 4,5, 8,13	МГ През Д МШ	3-я

Параллельные алгоритмы для решения простых типовых задач на системах с общей памятью. Глобальная и локальная память потоков. Критические секции. Синхронизация потоков. Редуцированные операции потоков.						
3.1. Параллельные алгоритмы матричных умножений на системах с распределенной памятью. Параллелизм по данным (геометрический параллелизм). Ускорение и эффективность параллельных алгоритмов. Масштабируемость алгоритмов. Задание виртуальных MPI - топологий. Локальные и глобальные схемы передачи данных, структурные способы взаимодействия, статические и динамические схемы передачи данных.	ОК1		2	1,2 4,5, 8,13	МГ През Д МШ	4-я
3.2. Параллельные алгоритмы матричных умножений на системах с общей памятью. Распределение вычислений между потоками.	ПК2		2	1,2 4,5, 8,13	МГ През Д МШ	5-я
		План..	2	1,2 4,5, 8,13	МГ През Д МШ	
Итого модуль 1	***7 сем***		***12 ч***			
		Модуль 2				
4.1. Параллельные методы решения систем линейных алгебраических уравнений итерационными методами на системах с распределенной памятью. Статическая балансировка загрузки процессоров.	ОК1		1	1,2 4,5, 8,13	МГ През Д МШ	6-я
4.2. Параллельные методы решения систем линейных алгебраических уравнений итерационными методами на системах с общей памятью. Разделяемые ресурсы.	ПК2		2	1,2 4,5, 8,13	МГ През Д МШ	7-я

4.3. Параллельные методы решения систем линейных алгебраических уравнений прямыми методами на системах с распределенной памятью.	ОК1		2	1,2 4,5, 8,13	МГ През Д МШ	8-я
4.4. Параллельные методы решения систем линейных алгебраических уравнений прямыми методами на системах с общей памятью.	ПК2		2	1,2 4,5, 8,13	МГ През Д МШ	9-я
5.1. Параллельные методы решения дифференциальных уравнений в частных производных на системах с распределенной памятью. Конвейерный параллелизм. Закон Амдала.	ОК1 ПК2	*План* 1.Дайте оценку_____ 2.Обоснуйте_____ 3.Предложите _____ *Форма контроля: През.КС*	2	2, 4, 7,10, 11,1 4	МГ, През КС	10-я
5.1. Параллельные методы решения дифференциальных уравнений в частных производных на системах с общей памятью. Проблемы синхронизации параллельных вычислений. Семафоры. Блокировки.	ОК1		1	2, 4, 7,10, 11,1 4	МГ, През КС	11-я
6.1. Параллельные методы сортировок на системах с распределенной памятью. Параллельное обобщение базовой операции сортировки.	ПК2		1	2, 4, 7,10, 11,1 4	МГ, През КС	12-я
6.2. Параллельные методы сортировок на системах с общей памятью.	ОК1		2	2, 4, 7,10, 11,1	МГ, През КС	13-я
6.3. Параллельные алгоритмы в MPI совместно с OpenMP.	ПК2		2	2, 4, 7,10,	МГ, Пре	14-я
6.4. Параллельные алгоритмы с использованием библиотеки MKL на системах с распределенной памятью.	ОК1		2	2, 4, 7,10, 11,1	МГ, През КС	15-я
Итого модуль 2	***8 сем***		***12 ч***			***16 нед***
ВСЕГО:	**15 сем.**		**24 ч**			***16 нед***

Практические занятия по темам Тема 1.1. (в MPI)

Практическое занятие 1. Две важные макрооперации, часто используемые при конструировании параллельных алгоритмов.

Цель - дополнить представление о конструировании простых параллельных алгоритмов на системах с распределенной памятью на более сложных примерах; дать представление о

параллельных программах, настраиваемых на размер вычислительной системы, как на параметр. Дать представление о важных макро операциях, используемых во многих параллельных алгоритмах.

Задача 1.1. Параллельные алгоритмы циклического сдвига на системе с распределенной памятью со структурой сети связей между процессами - “кольцо”, в которых каждый процесс инициирует рассылку своего сообщения одновременно в каком- либо выбранном направлении по кольцу.

Вариант 1.2. С асинхронной блокированной передачей, блокированным приемом. ***Вариант 1.3.*** С асинхронной неблокированной передачей, неблокированным приемом.

Вариант 1.4. С синхронной передачей, блокированным приемом.

Задача 1.2. Параллельные алгоритмы конвейерной передачи данных на системе с распределенной памятью со структурой сети связей между процессами - “линейка”.

Тема 2.1. (в MPI)

Практическое занятие 2. Операции над элементами векторов на системах с распределенной памятью.

Цель - дать общее представление о конструировании простых параллельных алгоритмов на системах с распределенной памятью. Практическое освоение основных функций MPI. Изучение редуцированных MPI-операций.

Задача 2.1. Параллельный алгоритм скалярного произведения векторов с использованием редуцированных операций (MPl_Reduce (...) , MPl_Allreduce (...)).

Задача 2.2. Параллельный алгоритм скалярного произведения векторов с использованием парных взаимодействий (MPl_Send (...) , MPl_Recv(...)).

Задача 2.3. Параллельный алгоритм суммирования элементов векторов с использованием редуцированных операций (MPl_Reduce (...) , MPl_Allreduce (...)).

1. Сравнить время выполнения алгоритмов задач 2.1 и 2.2.

Тема 2.2. (в OpenMP)

Практическое занятие 3. Операции над элементами векторов на системах с общей памятью.

Цель - дать общее представление о конструировании простых параллельных алгоритмов на системах с общей памятью.

Задача 3.1. Параллельные алгоритмы скалярного произведения векторов.

Вариант 3.1. Параллельный алгоритм скалярного произведения векторов с использованием редуцированной операции (reduction (...)).

Вариант 3.2. Параллельный алгоритм скалярного произведения векторов с использованием директив critical{...} или atomic.

Задача 3.2. Параллельные алгоритм суммирования элементов векторов.

Тема 3.1. (в MPI)

Практическое занятие 4. Умножение матрицы на вектор на системах с распределенной памятью на сети связей между процессами с топологией “кольцо”.

Цель - дать знания о конструировании параллельных алгоритмов умножения матрицы на вектор на системах с распределенной памятью; практическое освоение основных функций MPI.

Вариант 4.1. Количество строк матрицы ***M*** и элементов вектора ***V*** без остатка делятся на размер вычислительной системы ***P.*** Матрица ***M*** распределена по процессам горизонтальными ленточными полосами, вектор ***V*** распределен блоками.

Вариант 4.2. Количество строк матрицы ***M*** и элементов вектора ***V*** не обязательно на цело делятся на размер вычислительной системы. Матрица ***M*** распределена по процессам горизонтальными ленточными полосами, вектор ***V*** распределен блоками.

Практическое занятие 5. Умножение матрицы на матрицу на системах с распределенной памятью на сети связей между процессами с топологией “кольцо”.

Цель - дать знания о конструировании параллельных алгоритмов умножения матриц на системах с распределенной памятью.

Вариант 5.1. Матрица ***Mi*** распределена по процессам горизонтальными ленточными полосами, матрица M_2 распределена вертикальными ленточными полосами.

Вариант 5.2. Матрицы M_1 и M_2 обе распределены по процессам горизонтальными ленточными полосами.

1. Сделать выводы.

Практическое занятие 6. Умножение матрицы на матрицу на системах с распределенной памятью на сети связей между процессами “2D решетка” (алгоритм Фокса).

Цель - дать знания о конструировании параллельных алгоритмов умножения матриц на системах с распределенной памятью; дать знания о конструировании многомерных декартовых структур; дать общее представление о масштабируемости задач; практическое освоение основных функций MPI.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить с аналогичными показателями этих же алгоритмов, реализованных на топологии “кольцо” (лабораторная работа 5).
2. Сделать выводы.

Тема 3.2. (в OpenMP)

Практическое занятие 7. Умножение матрицы на вектор и матрицы на матрицу на системах с общей памятью.

Цель - дать знания о конструировании параллельных алгоритмов матричных умножений на системах с общей памятью; дать общее представление о масштабируемости задач; практическое освоение основных директив OpenMP; способах распределения вычислений между потоками; способах распределения вычислений итерационных циклов между потоками.

Задача 7.1. Параллельный алгоритм умножения матрицы на вектор.

Вариант 7.1.1. Параллельный алгоритм умножения матрицы ***M*** на вектор ***V*** с использованием директивы распараллеливания параметрических циклов #pragma omp for.

Вариант 7.1.2. Параллельный алгоритм умножения матрицы ***M*** на вектор ***V*** с использованием “ручного” задания работ (распараллеливания циклов).

Задача 7.2. Параллельный алгоритм умножения матрицы на матрицу с использованием директивы распараллеливания параметрических циклов #pragma omp for.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями этих же алгоритмов реализованных на системах с распределенной памятью.
2. Сделать выводы.

Тема 4.1. (в MPI)

Практическое занятие 8. Решение систем линейных алгебраических уравнений, используя явные двухслойные итерационные схемы.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений явными итерационными методами, используя двухслойные итерационные схемы, на вычислительных системах с распределенной памятью; практическое освоение основных функций MPI.

Данные равномерно распределены по процессам.

Задача 8.1. Решение системы линейных алгебраических уравнений $Ax = f$ ***методом простой итерации.***

Задача 8.2. Решение системы линейных алгебраических уравнений ***Ax = f методом оптимального чебышевского набора параметров.***

Практическое занятие 9. Решение систем линейных алгебраических уравнений, используя неявные попеременно треугольные методы.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений неявными итерационными методами, используя попеременно треугольные методы, на вычислительных системах с распределенной памятью; практическое освоение основных функций MPI.

Ленточное распределение данных по процессам.

Задача 9.1. Решение системы линейных алгебраических уравнений ***Ax = f методом Зейделя:***

Задача 9.2. Решение системы линейных алгебраических уравнений ***Ax = f методом верхней релаксации :***

Тема 4.2. (в OpenMP)

Практическое занятие 10. Решение систем линейных алгебраических уравнений, используя явные двухслойные итерационные схемы.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений явными итерационными методами, используя двухслойные итерационные схемы, на вычислительных системах с общей памятью.

Задача 10.1. Решение системы линейных алгебраических уравнений ***Ax = f методом простой итерации.***

Задача 10.2. Решение системы линейных алгебраических уравнений ***Ax = f методом оптимального чебышевского набора параметров.***

Практическое занятие 11. Решение систем линейных алгебраических уравнений, используя неявные попеременно треугольные методы.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений неявными итерационными методами, используя попеременно треугольные методы, на вычислительных системах с общей памятью; практическое освоение основных функций MPI.

Задача 11.1. Решение системы линейных алгебраических уравнений ***Ax = f методом Зейделя:***

Задача 11.2. Решение системы линейных алгебраических уравнений ***Ax = f методом верхней релаксации :***

Тема 4.3. (в MPI)

Практическое занятие 12. Решение систем линейных алгебраических уравнений прямыми методами на системах с распределенной памятью.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений прямыми методами на вычислительных системах с распределенной памятью. Ленточное распределение данных по процессам.

Задача 12.1. Решение системы линейных алгебраических уравнений ***Ax = f методом Гаусса-1.*** Алгоритм основан на ленточном разбиении матрицы коэффициентов ***A*** горизонтальными полосами с непрерывными последовательностями строк.

Задача 12.2. Решение системы линейных алгебраических уравнений ***Ax = f методом Гаусса-2.*** Алгоритм основан на ленточном разбиении матрицы коэффициентов ***A*** горизонтальными ***циклическими*** полосами.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями алгоритмов решения СЛАУ, реализованных ите-

рационными методами на системах с распределенной памятью и на системах с общей памятью.

2. Сделать выводы.

Тема 4.4. (в OpenMP)

Практическое занятие 13. Решение систем линейных алгебраических уравнений прямыми методами на системах с общей памятью.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений прямыми методами на вычислительных системах с общей памятью.

Задача 13.1. Решение системы линейных алгебраических уравнений ***Ax = f методом Гаусса.***

Задача 13.2. Решение системы линейных алгебраических уравнений ***Ax = f методом Гаусса-Зейделя.***

1. Сравнить показатели ускорения и эффективности приведенного алгоритма, сравнить его с аналогичными показателями алгоритмов решения СЛАУ на системах с распределенной памятью, а так же с реализованными итерационными методами на системах с распределенной памятью и на системах с общей памятью.
2. Сделать выводы.

Тема 5.1. (в MPI)

Практическое занятие 14. Численное решение задачи Дирихле для уравнения Пуассона на системах с распределенной памятью методом Якоби.

Цель - изучить организацию параллельных вычислений на системах с распределенной памятью решения задач явными методами, в которых при математическом моделировании используются дифференциальные уравнения в частных производных.

Задача 14. Численное решение задачи Дирихле для уравнения Пуассона методом Якоби, которая определяется как задача нахождения функции ***u = u(x,*** y), удовлетворяющей в области определения 2D уравнению:

$$\begin{cases} d^2u \; \partial^2 u \;,, \; . \; , \; . \\ = f(x, y), (x, y) \in 2D \\ < \partial x \; \partial y \; u(x, y) = g \; (x, y), (x, y) \in 2D^0 \end{cases}$$

и принимающей значения ***g*** (x, y) на границе $2D^0$ области 2D (f и g являются функциями, задаваемыми при постановке задачи).

Вариант 14.1. Решение задачи Дирихле в области 2D для уравнения Пуассона с использованием 1D декомпозиции вычислительного пространства (на сети связей процессов с топологией “кольцо”).

Вариант 14.2. Решение задачи Дирихле в области 2D для уравнения Пуассона с использованием 2D декомпозиции вычислительного пространства (на сети связей процессов с топологией “2D решетка”).

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Практическое занятие 15. Численное решение задачи Дирихле для уравнения Пуассона на системах с распределенной памятью методом Зейделя.

Цель - изучить организацию параллельных вычислений на системах с распределенной памятью решения задач неявными методами, в которых при математическом моделировании используются дифференциальные уравнения в частных производных.

Вариант 15.1. Решение задачи Дирихле в области 2D для уравнения Пуассона с использованием 1D декомпозиции вычислительного пространства (на сети связей процессов с топологией “кольцо”).

Вариант 15.2. Решение задачи Дирихле в области 2D для уравнения Пуассона с использованием 2D декомпозиции вычислительного пространства (на сети связей процессов с топологией “2D решетка”).

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями алгоритмов решения этих же задач методом Якоби.
2. Сделать выводы.

Тема 5.2. (в OpenMP)

Практическое занятие 16. Численное решение задачи Дирихле в области 2D для уравнения Пуассона на системах с общей памятью методом Якоби.

Цель - изучить организацию параллельных вычислений на системах с общей памятью решения задач явными методами, в которых при математическом моделировании используются дифференциальные уравнения в частных производных.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями алгоритмов решения этих же задач на системах с распределенной памятью.
2. Сделать выводы.

Практическое занятие 17. Численное решение задачи Дирихле в области 2D для уравнения Пуассона на системах с общей памятью методом Зейделя.

Цель - изучить организацию параллельных вычислений на системах с общей памятью решения задач неявными методами, в которых при математическом моделировании используются дифференциальные уравнения в частных производных.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями алгоритмов решения этих же задач на системах с распределенной памятью.
2. Сделать выводы.

Тема 6.1. (в MPI)

Практическое занятие 18. Сортировка на системах с распределенной памятью.

Цель - изучить организацию параллельных вычислений для не вычислительных задач на системах с распределенной памятью.

Задача 18.1. Сортировка методом пузырька.

Задача 18.2. Сортировка методом Шелла.

4. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
5. Сделать выводы.

Тема 6.2. (в OpenMP)

Практическое занятие 19. Сортировка на системах с общей памятью.

Цель - изучить организацию параллельных вычислений для не вычислительных задач на системах с общей памятью.

Задача 19.1. Сортировка методом пузырька.

Задача 19.2. Сортировка методом подстановок.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Тема 6.3. (в MPI и OpenMP)

Практическое занятие 20. Параллельные алгоритмы в MPI совместно с OpenMP.

Цель - изучить организацию параллельных вычислений одновременно в MPI и OpenMP.

Параллельные алгоритмы строятся одновременно в MPI и OpenMP. Параллельные вычисления внутри узла вычислительного кластера осуществляются в OpenMP, а взаимодействия процессов, расположенных в разных узлах, осуществляются в MPI.

Задача 20.1. Параллельные алгоритмы вычисления определенного интеграла

$$y = \int_a^b f(x)dx$$

с использованием метода прямоугольников (трапеций) для ***f*** (x) = sin(x), ***f*** (x) = x^2, ***f***(x)= .

Задача 20.2. Численное решение задачи Дирихле в области 2D для уравнения Пуассона методом Якоби с использованием 1D декомпозиции вычислительного пространства (на сети связей MPI-процессов с топологией “кольцо”).

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Тема 6.4. (в MPI)

Практическое занятие 21. Параллельные алгоритмы в MPI с использованием библиотеки MKL.

Цель - изучить организацию параллельных вычислений с использованием библиотеки параллельного программирования MKL; дать общее представление о библиотеке MKL; изучить вызовы функций MKL.

Задача 21.1. Параллельный алгоритм умножения матрицы на вектор.

Задача 21.2. Параллельный алгоритм умножения матрицы на матрицу.

Задача 21.3. Решение системы линейных алгебраических уравнений ***Ax*** = ***f.***

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов с алгоритмами построиными и реализованными в предыдущих лабораторных работах.
2. Сделать выводы.

10.2.3. Самостоятельная работа студентов (СРС)

№ и темы заданий	Компет.	Задания на СРС	К-во час	Форма контроля	Лит - ра	Сроки сдачи
		Модуль 1				
Тема1._____ _____	ОК10 ПК-1	1. Изобразите _____	4	Схема	1,2 4,5, 8,13	2-я нед
Тема 2 и т.д.			4			
Итого модуль 1			***28 ч***			***7 нед***
		Модуль 2				
Тема8. _____ _____	ОК10 ПК4	1. Составьте _____ 2.Разработайте свой _____	2 2	Анал справка План		9-я
Тема 9 и т.д.						
Итого модуль 2			***32 ч***			***8 нед***

ВСЕГО:			**60 ч**			**15 нед**

Дополнительные задания для самостоятельной работы по темам Тема 1.1. (в MPI)

Самостоятельная работа 1. Парные взаимодействия процессов в MPI.

Цель - дать общее представление об операциях передачи сообщений между двумя процессами на системах с распределенной памятью, закрепить практическое освоение функций парных взаимодействий между ветвями параллельной программы.

Вариант 1.1. Взаимодействия с асинхронной блокированной передачей, блокированным приемом (MPI_Send (...), MPI_Recv(...)).

Вариант 1.2. Взаимодействия с асинхронной неблокированной передачей, неблокированным приемом (MPI_Isend (...), MPI_Irecv(...)).

Вариант 1.3. Взаимодействия с синхронной передачей, блокированным приемом (MPI_Ssend (...), MPI_Recv (...)).

1. Оценить общее время ***T*** работы построенной параллельной программы при ***N*** кратной передаче сообщений размером ***M*** всем процессам выделенной группы; T_p - время однократной рассылки сообщений всем ***P*** процессам выделенной группы; T - время однократной передачи сообщения от одного процесса другому процессу.
2. Оценить зависимость время-ёмкости парных операций от длины сообщений и от размера распределенной вычислительной системы. Построить соответствующие графики.

Тема 2.1. (в MPI)

Самостоятельная работа 2. Параллельные алгоритмы с коллективными взаимодействиями процессов в MPI.

Цель - дать общее представление о коллективных операциях передачи сообщений между процессами на системах с распределенной памятью, закрепить практическое освоение функций коллективных взаимодействий между ветвями параллельной программы.

Вариант 2.1. Взаимодействия с передачей сообщения от одного процесса всем (MPI_Bcast (...)).

Вариант 2.2. Взаимодействия с передачей сообщения от одного процесса всем с разрезанием сообщения на части (MPi_Scatter (...)).

Вариант 2.3. Взаимодействия с передачей сообщения от всех процессов всем процессам (MPI_Allgather (...), MPI_Alltoall (...)).

1. Оценить общее время ***T*** работы построенной параллельной программы при ***N*** кратном выполнении коллективной операции с сообщениями размером M. T_1, - время однократного выполнения коллективной операции.

2. Сравнить общее время ***T*** работы построенных параллельных программ, при ***N*** кратной передаче сообщений размером ***M*** всем процессам группы размером ***P,*** реализованных парными взаимодействиями (сам. работа 1) и коллективными взаимодействиями (MPI_Bcast (...) , MPI_Allgather (...)).

3. Оценить зависимость время-ёмкости коллективных операций от длины сообщений и от размера распределенной вычислительной системы. Построить соответствующие графики.

4. Сделать выводы.

Самостоятельная работа 3. Умножение матрицы на вектор на системах с распределенной памятью на сети связей между процессами с топологией “кольцо”.

Цель - практическое освоение основных функций MPI. Изучение функций сбора данных от всех процессов.

Задача 3. Матрица ***M*** распределена по процессам горизонтальными ленточными полосами, вектор ***V*** дублирован в памяти всех процессов. После умножения, распределенный

блоками по процессам вектор результата ***C,*** восстанавливается в целый вектор (конкатенацией блоков) и перезаписывается в вектор ***B*** в памяти всех процессов.

Тема 2.2. (в OpenMP)

Самостоятельная работа 4. Основные директивы в OpenMP.

Цель - дать общее представление об основных директивах OpenMP - программного средства программирования на системах над общим полем памяти.

Задача 4.1. Параллельный алгоритм, в котором область программы, исполняемая ***P*** потоками параллельно, выполняется многократно в цикле, т.е. параллельная область программы, окаймлена (например) итерационным циклом с предусловием.

```
    n = N;
    tn = Tn;
    t1 = omp get wtime();
    while(n--) {
#pragma omp parallel {
} sleep(tn);
}
    t2 = omp get wtime();
    T1 = t2-t1;
```

Задача 4.2. Параллельный алгоритм, в котором область программы, исполняемая ***P*** потоками параллельно, включает в себя многократно выполняемый итерационный цикл с предусловием.

```
n = N; tn = Tn;
    t1 = omp get wtime();
#pragma omp parallel {
while(n--)
{
sleep(tn);
}
    }
    t2 = omp get wtime();
    T2 = t2-t1;
```

1. Оценить общее время T_i работы построенных параллельных программ (в задачах 3.1-3.2) при ***N*** кратном выполнении итерационного цикла и на ***P*** потоках. Вычисления имитировать временной задержкой (sleep (t_n)).
2. Сравнить времена работы параллельных программ в задачах 3.1-3.2 для ***P*** потоков, а так же сравнить их с временем работы программы задачи 3.1 (или задачи 3.2), в которой директива порождения нитей закомментирована, т.е. отсутствует параллельная область. И таким образом оценить время T_v выполнения главной директивы порождения и слияния нитей в OpenMP.
3. Оценить зависимость время-ёмкости программ для разного количества потоков. Построить соответствующие графики.

Тема 4.1. (в MPI)

Самостоятельная работа 5. Решение систем линейных алгебраических уравнений итерационными методами вариационного типа.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений вариационного типа, которые не используют в явном виде информацию о параметрах γ_1, γ_2, являющихся собственными значениями матрицы ***A*** на вычислительных системах с распределенной памятью.

Методы двухслойные, явные. Ленточное распределение данных по процессам.

Задача 5.1. Решение системы линейных алгебраических уравнений ***Ax = f методом минимальных невязок.***

Задача 5.2. Решение системы линейных алгебраических уравнений ***Ax = f методом скорешего спуска.***

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов и алгоритмов в лабораторных работах решения систем линейных алгебраических уравнений разными методами.
2. Сделать выводы.

Тема 4.2. (в OpenMP)

Самостоятельная работа 6. Решение систем линейных алгебраических уравнений итерационными методами вариационного типа.

Цель - практическое освоение методов решения систем линейных алгебраических уравнений вариационного типа, которые не используют в явном виде информацию о параметрах γ_1, γ_2, являющихся собственными значениями матрицы ***A*** на вычислительных системах с общей памятью.

Методы двухслойные, явные.

Задача 6.1. Решение системы линейных алгебраических уравнений ***Ax = f методом минимальных невязок.***

Задача 6.2. Решение системы линейных алгебраических уравнений ***Ax = f методом скорешего спуска.***

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов и алгоритмов в лабораторных работах решения систем линейных алгебраических уравнений разными методами, а тек же сравнить эти показатели с аналогичными показателями для тех же задач, но решаемых на системах с распределенной памятью.
2. Сделать выводы.

Тема 5.1. (в MPI)

Самостоятельная работа 7. Обмен данными в MPI-типах с использованием коллективных и парных операций.

Цель - практическое освоение методов задания MPI-типов, одним из важнейших инструментов манипуляции данными, и использования этих типов в операциях взаимодействий.

Задача 7. Задание MPI-типов граней 2D и 3D массивов и использование этих типов данных в коллективных и парных обмена между процессами.

Самостоятельная работа 8. Численное решение задачи Дирихле в области 3D для уравнения Пуассона с использованием 1D декомпозиции вычислительного пространства.

Самостоятельная работа 9. Численное решение задачи Дирихле в области 3D для уравнения Пуассона с использованием 1D декомпозиции вычислительного пространства.

1. Сравнить показатели ускорения и эффективности приведенного алгоритма, а так же сравнить его с аналогичными показателями алгоритмов решения этих же задач методом Якоби.
2. Сделать выводы.

Тема 5.2. (в OpenMP)

Самостоятельная работа 10. Численное решение задачи Дирихле в области 3D для уравнения Пуассона методом Якоби.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить их с аналогичными показателями алгоритмов решения этих же задач на системах с распределенной памятью.
2. Сделать выводы.

Самостоятельная работа 11. Численное решение задачи Дирихле в области 3D для уравнения Пуассона методом Зейделя.

Тема 6.1. (в MPI)

Самостоятельная работа 12. Сортировка методом Батчера.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Тема 6.2. (в OpenMP)

Самостоятельная работа 13. Сортировка на системах с общей памятью.

Задача 14.1. Сортировка методом Шелла.

Задача 14.2. Сортировка методом Батчера.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Тема 6.3. (в MPI и OpenMP)

Самостоятельная работа 14. Параллельные алгоритмы в MPI совместно с OpenMP.

Задача 14. Численное решение задачи Дирихле в области 2D для уравнения Пуассона методом Зейделя с использованием 2D декомпозиции вычислительного пространства.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов.
2. Сделать выводы.

Тема 6.4. (в MPI)

Самостоятельная работа 15. Параллельный алгоритм решения задачи Дирихле для уравнения Пуассона в MPI с использованием библиотеки MKL.

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов с алгоритмами построиными и реализованными в предыдущих лабораторных работах.
2. Сделать выводы.

Дополнительные задания повышенной сложности для самостоятельной работы по темам

Тема 3.1. (в MPI)

Самостоятельная работа 16. Умножение матрицы на матрицу на системах с распределенной памятью на сети связей “многомерная решетка”.

Задача 16. Параллельный алгоритм умножения матрицы ***Mi*** на матрицу M_2 на сети связей между процессами с топологией “3D решетка” (алгоритм Кеннона).

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же сравнить с аналогичными показателями этих же алгоритмов, реализованных на топологии “кольцо”.
2. Сделать выводы.

Тема 5.1. (в MPI)

Самостоятельная работа 17.

Задача 17.1. Решение задачи Дирихле методом Якоби в области 3D для уравнения Пуассона с использованием 2D декомпозиции вычислительного пространства.

Задача 17.2. Решение задачи Дирихле методом Якоби в области 3D для уравнения Пуассона с использованием 3D декомпозиции вычислительного пространства (на сети связей процессов с топологией “3D решетка”).

Задача 17.3. Решение задачи Дирихле в области 3D для уравнения Пуассона с использованием 2D декомпозиции вычислительного пространства.

Задача 17.4. Решение задачи Дирихле в области 3D для уравнения Пуассона с использованием 3D декомпозиции вычислительного пространства (на сети связей процессов с топологией “3D решетка”).

1. Сравнить показатели ускорения и эффективности приведенных алгоритмов, а так же

сравнить с аналогичными показателями этих же алгоритмов, реализованных на топологии “кольцо”.

2. Сделать выводы.

Для всех лабораторных и самостоятельных работ необходимо выполнить следующие задания:

1. Найти зависимость временны Пх характеристик алгоритмов от размеров вычислительной системы и размеров входных данных.
2. Построить графики ускорения и эффективности сконструированных алгоритмов.

11. Образовательные технологии

Курс «Основы параллельного программирования» включает в себя:

1. Лекционные занятия.
2. Практические занятия.
3. Самостоятельные работы.

Лекционные занятия

Образовательные технологии при подаче лекционного материала заключаются в следующем:

- Лекционный материал преподается в интерактивной форме. При этом используются средства отображения презентаций и других лекционных материалов на экран. Студенты конспектируют основные разделы лекционного материала по указанию преподавателя.
- В конце каждой лекции отводится дополнительное время (5-7 мин.) для ответов на вопросы, возникающие у студентов в процессе прослушивания лекции (активная форма).
- В конце каждого раздела дисциплины (2-3 лекции) в активной форме проводится повторение основных моментов предыдущих лекций по пройденному разделу и ответов на вопросы, возникшие в результате самостоятельной проработки лекционного материала. Для более глубокого понимания теории студентам предлагаются ссылки на статьи или прочие электронные ресурсы, дающие более детальное описание рассматриваемых проблем.
- В конце семестра проводится промежуточный контроль усвоения материала. Контроль проводится в письменной форме по билетам.

Данная стратегия ведения лекций позволяет устранить пробелы в понимании, возникающие на разных этапах восприятия лекционного материала.

Практические занятия

Образовательные технологии при проведении практических занятий широко используют в своем учебном процессе активные и интерактивные формы проведения занятий и состоят из двух типов проведения занятий:

1. Проведение аудиторных практических занятий.
2. Проведение практических занятий с использованием информационной сети Интернет.

Образовательные технологии при проведении аудиторных практических занятий состоят в следующем:

- В начале каждого аудиторного занятия проводится разбор и объяснение в интерактивной форме дополнительного теоретического материала, необходимого для выполнения практического задания.
- Далее осуществляется подробное обсуждение и разбор практических заданий. Разбор и обсуждение осуществляются в активной форме вопросов - ответов.
- В течении следующих 45 минут студенты самостоятельно выполняют задание. При этом осуществляется разбор конкретных ситуаций.

- Затем проводится разбор и объяснение дополнительных самостоятельных заданий, которые студенты должны выполнить вне аудиторных занятий.
- По желанию студента он может выполнять нестандартное более сложное задание, включающее, например, вопросы исследовательского характера, связанные, например, эффективностью разрабатываемых параллельных алгоритмов. Это обсуждается с преподавателем индивидуально и, при успешном выполнении, учитывается в общей оценке работ студента.
- В конце аудиторного занятия в соответствии с календарным планом осуществляется контроль по выполнению самостоятельных работ, заданных на предыдущих занятиях. Производится оценка работоспособности и функциональности предлагаемого студентом программного продукта, связанного с конкретным заданием, и производится защита программного кода.

В течение семестра проводится в соответствии с календарным планом два промежуточных аудиторных занятия по контролю своевременных сдач и оценке выполненных практических работ за предыдущий интервал времени, а в конце семестра выделяется аудиторное занятие (одно или два) для сдачи и досдачи всех самостоятельных практических работ.

- На контрольном занятии каждый студент после сдачи программной части защищает свой отчет (в виде пояснительной записки) по каждому заданию и отвечает на теоретические вопросы. Результирующая оценка работы находится как среднее арифметическое от оценок за программную часть и отчет с соответствующими весовыми коэффициентами каждой части.

Самостоятельные работы

Образовательные технологии при проведении самостоятельных работ широко используют в своем учебном процессе активные и интерактивные формы проведения занятий и состоят в следующем:

- Обсуждение и разбор самостоятельных работ осуществляется в активной форме вопросов - ответов при проведении аудиторных занятий или с использованием сети Internet.
- Освоение разделов тем для написания рефератов и обсуждения на семинарах. Обсуждение и разбор рефератов осуществляется в активной форме при проведении аудиторных занятий или с использованием сети Internet. При этом проводятся консультации по выбору темы реферата, подбору литературы, детализации планов рефератов и ответов на возникающие вопросы.
- Доклад на семинаре.

Образовательные технологии при проведении занятий с использование сети Интернет состоят в следующем.

По сети Интернет индивидуально с каждым студентом осуществляется взаимодействие в виде:

- Вопросов-ответов, связанных с дополнительными самостоятельными заданиями или индивидуальными нестандартными более сложными заданиями.
- Выдачи дополнительных заданий или нестандартных более сложных заданий.
- Контроля выполнения заданий и усвоения материала.

Задания и контрольные вопросы для самостоятельных работ дополнительно выставлены на сайте ssdonline.ssscc.ru/korneev.

Удельный вес занятий, проводимый в интерактивных формах, является главным в программе и в целом учебном процессе он составляет не менее 60% аудиторных занятий. Занятия лекционного типа составляют не более 40% аудиторных занятий.

12. Учебно-методическое и информационное обеспечение дисциплины

а)Основная литература:

1. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. Изд-во НГТУ, 2006 г.
2. В.Д. Корнеев. Параллельное программирование кластеров: учеб. пособие. - Новосибирск: Изд-во НГТУ, 2008. - 312 с.
3. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. - СПб.: БХВ- Петербург, 2002. - 609с.
4. Г.Р. Эндрюс. Основы многопоточного параллельного и распределенного программирования. - М.: Изд. Дом Вильямс, 2003. - 330 с.
5. С. Немнюгин, О. Стесик. Параллельное программирование для многопроцессорных вычислительных систем. -СПб.: БХВ - Петербург, 2002. - 400 с.
6. Д. Кнут. Искусство программирования для ЭВМ. - М.: Изд. Мир, 1978. - Т.3. – 843 с.
7. Богачев К.Ю. Основы параллельного программирования. - М.: БИНОМ. Лаборатория знаний, 2003.

б) Дополнительная литература:

1. Малышкин В.Э. Основы параллельных вычислений. - Новосибирск, 1998. - (Методическое пособие, Изд-во НГТУ;) - 55 с.
2. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. - Новосибирск: Наука. Сибирское отделение, 1988. - 128 с.
3. В.Д. Корнеев. Параллельные программирование в MPI. - Новосибирск, 2002. 215 с.
4. ***Snir M., Otto S. W., Huss-Lederman S., Walker D., andDongarra J.*.** MPI: The Complete Reference. MIT Press. Boston, 1996.
5. Питерсон Дж. Теория сетей Петри и моделирование систем. - М.: Мир, 1984.
6. Хоар Ч. Взаимодействующие последовательные процессы. - М.: Мир, 1989. -264 с.
7. Берзин Ю.А., Вшивков В.А. Метод частиц в разреженной плазме. - Новосибирск: Наука, 1980.
8. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. - Н.Новгород, ННГУ, 2001.
9. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. - М.: МЦНТО, 1999.
10. Тихонов А.Н., Самарский А.А. Уравнения математической физики. - М.: Наука, 1977.
11. Fox G.C. et.al. Solving Problems on Concurrent Processors. - Prentice Hall, Englewood Cliffs, NJ, 1988.

в) учебно-методические пособия:

12. Стрченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах. - Томск: ТГУ, 2002.
13. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI: Пособие - Мн. БГУ, 2002.
14. Якобовский М.В. Распределенные системы и сети. - М.: МГТУ «Станкин», 2000.

в) Интернет-ресурсы

1. Информационно-аналитические материалы по параллельным вычислениям (http://parallel.ru).
2. Информационные материалы Центра компьютерного моделирования Нижегородского университета (http://www.software.unn.as.ru/ccam).
3. Информационные материалы рабочей группы IEEE по кластерным вычислениям (http://www.ieeetfcc.org).
4. Материалы по параллельным вычислениям (www.openmp.opg).

д) программное обеспечение: OS Linux, MPI-2, OpenMP, C, Fortran.

Материально-техническое обеспечение дисциплины «Введение в параллельное программирование»

Лекции читаются в специальной лекционной аудитории, обеспеченной средствами отображения презентаций и других лекционных материалов на экран.

Практические занятия проводятся в компьютерных аудиториях с отдельными рабочими местами для каждого студента.

13. Политика выставления баллов

В соответствии с картой накопления баллов, студент может набирать баллы по всем видам занятий. На лекциях и семинарах (*указать за что)* за_____, на лабораторных занятиях за_____; СРС за _____;
за рубежный контроль - максимум 10б за_____; итоговый контроль – максимум 40б за _____.

Критерий оценивания

Форма контроля	**Критерии оценивания**				**Этап**
	Отлично	**Хорошо**	**Удовл.**	**Неуд.**	
Семестр 1					
Текущий конт	**роль**				
Лабораторные работы	Оборудование и методы использованы правильно. Проявлена превосходная теоретическая подготовка. Необходимые навыки и умения полностью освоены. Результат лабораторной работы полностью соответствует её целям.	Оборудование и методы использованы в основном правильно. Проявлена хорошая теоретическая подготовка. Необходимые навыки и умения в основном освоены. Результат лабораторной работы в основном соответствует её целям.	Оборудование и методы частично использованы правильно. Проявлена удовлетворительная теоретическая подготовка. Необходимые навыки и умения частично освоены. Результат лабораторной работы частично соответствует её целям.	Оборудование и методы использованы неправильно. Проявлена неудовлетворительная теоретическая подготовка. Необходимые навыки и умения не освоены. Результат лабораторной работы не соответствует её целям.	13
Самостоятельная работа	Правильно выполнены все задания. Продемонстрирован высокий уровень владения материалом. Проявлены превосходные способности применять знания и умения к выполнению конкретных заданий.	Правильно выполнена большая часть заданий. Присутствуют незначительные ошибки. Продемонстрирован хороший уровень владения материалом. Проявлены средние способности применять знания и	Задания выполнены более чем наполовину. Присутствуют серьёзные ошибки. Продемонстрирован удовлетворительный уровень владения материалом. Проявлены низкие способности применять знания и умения к выполнению конкретных заданий.	Задания выполнены менее чем наполовину. Продемонстрирован неудовлетворительный уровень владения материалом. Проявлены недостаточные способности применять знания и умения к выполнению конкретных заданий.	2

		умения к выполнению конкретных заданий.			
	Отлично	**Хорошо**	**Удовлетворительно**	**Неудовлетворительно**	

5. Оценочные средства для текущего контроля успеваемости, промежуточной аттестации по итогам освоения дисциплины и учебно-методическое обеспечение самостоятельной работы студентов

Правила текущего контроля:

1. В течение семестра необходимо представить и защитить 21 практическую работу и 15 самостоятельных работ по этим же темам, в сроки, установленные учебным графиком (см. таблицу).
2. К защите допускаются студенты, выполнившие практические работы в полном объеме (все задания согласно варианту) и оформившие отчет по работе в соответствии с требованиями.
3. На защите каждой практической работы предлагается 5-7 вопросов (по ходу выполнения работы).
4. Максимальное количество баллов выставляется за практическую работу, если студент полностью ответил на все вопросы, без серьезных замечаний и недочетов.
5. Выставляется 80% - 90% от максимального значения баллов за работу (в зависимости от вида работы), если студент полностью ответил на 80% - 90% вопросов.
6. Минимальное количество баллов (75% от максимального значения) выставляется, если студент ответил на 75% вопросов, с серьезными замечаниями, недочетами.
7. Пересдача практической работы назначается, если студент не ориентируется в учебном материале, не может объяснить ход и результаты выполнения работы. В случае пересдачи работы происходит потеря баллов (до 20% в зависимости от вида работы).
8. В случае представления и защиты работ с опозданием от учебного графика происходит потеря баллов (опоздание на 1 неделю - потеря 5% -9% в зависимости от вида работы, опоздание на 2 недели - потеря 10% - 15%, 3 недели и более - потеря 20% баллов от максимально возможного).

Правила рубежного контроля:

1. К промежуточной аттестации (зачету) допускаются студенты, сдавшие практические работы и набравшие не менее 75% баллов (60 баллов) по результатам текущего контроля.
2. Выставляется 20 баллов, если успешно выполнены все три задания без серьезных замечаний.
3. Выставляется 15 баллов, если успешно выполнены два задания из трех без серьезных замечаний и одно с замечаниями.
4. Для успешной сдачи модуля необходимо за теоретическую часть набрать не менее 15

баллов.

5. Возможно получить зачет «автомат» по дисциплине (без сдачи теоретической части), если студент в течение семестра выполняет дополнительные задания повышенной сложности и набирает свыше 90 баллов по текущему рейтингу.

К дополнительным заданиям повышенной сложности относятся:

1. Самостоятельное углубленное освоение тем;
2. Выполнение самостоятельных работ № 16 -17.

За дополнительные задания максимально выставляется 10 баллов.

Таблица

№п/п	**Вид учебной работы (учебной деятельности) (практические + самостоятельные по этим же темам)**	**Максимальное количество баллов**	**Срок представлени я и защиты (неделя семестра)**
1.	Практическое занятие № 1	1	2
2.	Практическое занятие №2	1	2
3.	Практическое занятие №3	1	3
4.	Практическое занятие №4	1	3
5.	Практическое занятие №5	2	4
6.	Практическое занятие №6	8	4
7.	Практическое занятие №7	2	5
8.	Практическое занятие №8	3	5
9	Практическое занятие №9	3	6
10	Практическое занятие №10	4	6
11	Практическое занятие №11	4	7
12	Практическое занятие №12	4	7
13	Практическое занятие №13	4	8
14	Практическое занятие №14	8	9
15	Практическое занятие №15	8	10
16	Практическое занятие №16	8	11
17	Практическое занятие №17	8	12
18	Практическое занятие №18	3	13
19	Практическое занятие №19	3	14
20	Практическое занятие №20	2	15
21	Практическое занятие №21	2	15
Итого по текущему рейтингу:		**80**	
22.	Зачет	20	
Итого за семестр:		**100**	

Для успешной сдачи модуля необходимо за практические работы набрать не менее 60 баллов и за теоретическую часть не менее 15 баллов, т.е. в сумме не менее 75 баллов.

Вопросы для текущего контроля успеваемости

Оценочные средства для текущего контроля успеваемости представляют собой контрольные вопросы к работам практических занятий и к самостоятельным работам по теме каждого из разделов.

Тема 1.1.

1. Модели программирования, поддерживаемые MPI.

2. Важные свойства MPI, обеспечивающие переносимость параллельных программ.
3. Семантика парных взаимодействий: синхронных, асинхронных, блокированных не блокированных.
4. В чем преимущества и недостатки синхронных и асинхронных взаимодействий.
5. Команды компиляции и запуска параллельных программ.

Тема 2.1.

6. Семантика коллективных операций:

a. MPI Cart create(...);
b. MPI Cart sub (...) ;
c. MPI Type create subarray(.) ;
d. MPI Scatter(...), MPI Scatterv(...);
e. MPI Gather(...), MPI Gatherv(...);
f. MPI_Alltoall (...) ;
g. MPI_Bcast (...) .

7. В чем преимущество использования коллективных операций перед парными.
8. Что понимается под операцией редукции.
9. В чем состоят возможные алгоритмы выполнения операции редукции? Какой алгоритм является наилучшим по времени выполнения: с MPI_Allreduce (...) или с парными взаимодействиями.

Тема 2.2.

10. Семантика директив:

a. #pragma omp parallel;
b. #pragma omp for;
c. #pragma omp sections.

11. В чем разница между потоком в OpenMP и процессом в MPI?
12. Сколько типов задания работ в OpenMP?
13. В чем различие глобальных и локальных переменных?
14. Каково время и область существования локальных переменных?

Тема 3.1.

15. Способы распределения подматриц между процессорами вычислительной системы с связи 1D («кольцо»), 2D, 3D.
16. Для чего задаются подрешетки Декартовых структур?
17. Что понимается в MPI под виртуальной топологией?
18. В чем различие между виртуальной и физической топологией процессов?
19. Как определить «соседей» по топологии?
20. Почему полезно использование виртуальных топологий?
21. В чем необходимость использования виртуальных топологий межкомпьютерных связей в вычислительной системе при решении задач?
22. Способы конструирования типов в MPI.
23. Для чего конструируются и используются MPI-типы данных?
24. Примеры задач, в которых используется операция умножения матрицы на вектор.
25. Общая характеристика программной реализации алгоритма Фокса.
26. Что такое ускорение и эффективность параллельного алгоритма? Как они определяются?
27. Что такое закон Амдала? Что он определяет?

Тема 3.2.

28. Методы распараллеливания и модели программ, поддерживаемые OpenMP.
29. Как распределяются элементы матриц и векторов по потокам?

30. Почему полученное ускорение построенных параллельных алгоритмов не линейно для разного размера данных?
31. Какие алгоритмы умножения матриц (одинаковых размеров) более эффективны в MPI или OpenMP на одном количестве процессов (потоков)?

Тема 4.1.

32. В чем состоит идея распараллеливания алгоритмов решения СЛАУ итерационными методами на системах с распределенной памятью?
33. Как распределяются элементы матриц и векторов по процессам?
34. Как организовать правильное завершение работы процессоров, если в качестве завершения вычислений взять условие:

$| xf^{+1} - xk | < £ i = 0,..., N -1$?

35. В чем различие между методами простой итерации и Г аусса-Зейделя для решения СЛАУ?
36. Какие показатели эффективности методов простой итерации и Гаусса-Зейделя?

Тема 4.2.

37. В чем состоит идея распараллеливания алгоритмов решения СЛАУ итерационными методами на системах с общей памятью?
38. Как распределяется обработка элементов матриц и векторов по потокам?
39. Как организовать правильное завершение работы процессоров, если в качестве завершения вычислений взять условие:

$| xk^{+1} - xk | < £ i = 0,..., N -1$?

40. Какие показатели эффективности методов простой итерации и Гаусса-Зейделя?
41. Почему метод Гаусса-Зейделя эффективнее метода простой итерации?
42. Какой алгоритм более эффективен: решения СЛАУ реализованных в MPI или OpenMP на одинаковом количестве процессоров (потоков) и на одинаковом размере исходных данных? Почему?

Тема 4.3.

43. В чем различие между прямыми и итерационными методами решения СЛАУ на системах с распределенной памятью?
44. Как распределяются элементы матрицы и вектора по процессам?
45. Какие показатели эффективности для параллельного варианта двух методов Г аусса?
46. Какой из двух методов Гаусса (с распределением матрицы горизонтальными ленточными полосами и горизонтальными циклическими полосами) является более быстродействующим для больших и маленьких матриц и почему?
47. Какие методы решения СЛАУ эффективнее: итерационные или прямые на системах с распределенной памятью?

Тема 4.4.

48. В чем различие между алгоритмами решения СЛАУ прямыми методами на системах с распределенной и общей памятью?
49. Как распределяется обработка элементов матрицы и вектора по потокам на системах с общей памятью?
50. Какие показатели эффективности для параллельных вариантов методов Гаусса и Гаусса-Зейделя?
51. Какие методы решения СЛАУ эффективнее: итерационные или прямые на системах с общей памятью?

Тема 5.1.

52. Как определяется задача Дирихле для уравнения Пуассона?

53. Какие способы определяют организацию параллельных вычислений для сеточных методов на системах с распределенной памятью?
54. Методы 1D, 2D и 3D декомпозиции 2D и 3D областей вычислений;
55. Обмен данными между процессорами на системах с распределенной памятью;
56. Процесс вычислений для явной и неявной схем;
57. Завершение вычислений для явной и неявной схем на системах с распределенной памятью;
58. Какой объем вычислительного пространства необходим для явных и неявных схем?
59. Какие методы решения задачи Дирихле более эффективны: явные или не явные?

Тема 5.2.

60. Какие способы определяют организацию параллельных вычислений для сеточных методов на системах с общей памятью?
61. В чем состоит проблема синхронизации параллельных вычислений?
62. Процесс вычислений для явной и неявной схем;
63. В чем состоит проблема взаимоблокировки?
64. Завершение вычислений для явной и неявной схем на системах с общей памятью;
65. Какой объем вычислительного пространства необходим для явных и неявных схем?
66. Какие методы решения задачи Дирихле более эффективны: явные или не явные на системах с общей памятью?
67. Какие методы решения задачи Дирихле более эффективны: на системах с общей памятью или на системах с распределенной памятью?

Тема 6.1.

68. В чем состоит постановка задачи сортировки данных на системах с распределенной памятью?
69. Какова организация параллельных вычислений при упорядочивании элементов множества методами пузырька, Шелла, Батчера на системах с распределенной памятью?
70. Какова вычислительная сложность приведенных алгоритмов?
71. Какой из приведенных алгоритмов является более эффективным?
72. Для каких топологий могут применяться рассмотренные алгоритмы сортировки?

Тема 6.2.

73. В чем состоит постановка задачи сортировки данных на системах с общей памятью?
74. Какова организация параллельных вычислений при упорядочивании элементов множества методами пузырька, Шелла, Батчера на системах с общей памятью?
75. Какова вычислительная сложность приведенных алгоритмов?
76. Какой из приведенных алгоритмов является более эффективным?

Тема 6.3.

77. В чем состоит организация вычислений в MPI совместно с OpenMP?
78. Какова организация параллельных вычислений определенного интеграла?
79. Какова организация параллельных вычислений задачи Дирихле в области 2D для уравнения Пуассона методом Якоби?
80. Какова вычислительная сложность приведенных алгоритмов?
81. Какой из приведенных алгоритмов является более эффективным?
82. Какой из алгоритмов решения задачи Дирихле является более эффективным: в MPI совместно с OpenMP или в MPI или OpenMP?

Тема 6.4.

83. Какие преимущества получает пользователь, использующий параллельные библиотеки?
84. Сравнить время умножения матрицы на матрицу вашего алгоритма и умножения с помощью библиотечной программы (библиотека MKL); Сделать выводы;

85. Какой алгоритм решения СЛАУ является более эффективным: ваш алгоритм или решение с помощью библиотеки MKL.
86. Каков вызов библиотечных программ из MKL для умножения матриц и для решения СЛАУ?

Контрольные вопросы для рубежного контроля.

В этой части пункта приводятся контрольные вопросы для лекционных занятий.

1.1. Основная идея формализации понятия вычислимой функции.

1.2. Операторы суперпозиции и примитивной рекурсии, порождаемые множества термов и реализующие программы.

1.3. Доказательство существования не примитивно рекурсивных функций. Оператор минимизации, порождаемые множества термов и реализующие программы.

2.1. Сформулировать задачу конструирования параллельной программы.

2.2. Понятие представления и реализации алгоритма. Непроцедурность представления алгоритма. Требования к представлению алгоритма.

2.3. Сравнительная непроцедурность языков программирования.

3.1. Основные понятия сети Петри. Сеть Петри как средство задания прямого управления в программах. Формулировка задачи взаимного исключения.

3.2. Понятие дедлока. Необходимые условия возникновения дедлока. Стратегии борьбы с дедлоком.

3.3. Задача об обедающих философах: формулировка и способы ее решения.

3.4. Задачи производитель-потребитель, конвейер.

4.1. Определение семафора. Программы задач производитель-потребитель и конвейер.

4.2. Задача читатели-писатели и ее программирование с использованием семафоров.

4.3. Понятие асинхронной программы. Проблемы асинхронного программирования. Определение MPI.

4.4. Управление параллельными программами. Типы асинхронного управления: событийное, потоковое, динамическое.

4.5. Параллельная программа разделения множеств и ее верификация.

4.6. Ускорение и эффективность вычислений. Закон Амдала.

4.7. Метод частиц-в-ячейках: описание схемы вычислений и особенности распараллеливания его алгоритмов. Лагранжева и Эйлерова декомпозиция.

4.8. Организация вычислений в реализации метода частиц-в-ячейках с Эйлеровой декомпозицией

4.9. Динамическая балансировка загрузки мультикомпьютера. Диффузионные и централизованные алгоритмы.

4.10. Модель параллельного программирования передачи сообщений. Краткая характеристика модели. Пример языка, поддерживающего данную модель.

4.11. Модель параллельного программирования с общей памятью. Краткая характеристика модели. Пример языка, поддерживающего данную модель.

4.12. Модель параллельного программирования параллелизма по данным. Краткая характеристика. Пример языка, поддерживающего данную модель.

5.1. Организация параллельных вычислений в крупноблочных иерархических мультикомпьютерах.

5.2. Организация параллельных вычислений в крупноблочных иерархических мультикомпьютерах. Линейные алгоритмы.

5.3. Организация параллельных вычислений в крупноблочных иерархических мультикомпьютерах. Фиксированный М-алгоритм.

6.1. Статическая постановка задачи отображения алгоритма на ресурсы вычислителя.

6.2. Эвристические алгоритмы конструирования отображения.
Организация параллельных вычислений в крупноблочных иерархических мультикомпьютерах. Отображение алгоритмов на ресурсы мультикомпьютера.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

ОШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Автоматизированные системы и цифровые технологии»

«Утверждено»
на заседании кафедры АСЦТ
от «_28 » _августа 2020г.

Зав. каф. АСЦТ, доцент Молдояров У.

_____

Силлабус

по дисциплине: «**Параллельные вычисления**»

для магитрантов обучающихся по направлению ИВТ
(шифр и наименование специальности)

Сетка часов по учебному плану

Наименование дисциплин	Количество часов					Рейтинг / консультация	Отчетность
	Всего ауд.	Аудит. занятия					
		Ауд. зан.	Лекция	Практ. (семин.)	Лабор.		1 сем.
Парал.прогр	*56*	*48*	*24*	-	*24*	*6 / 2*	*Экзамен*
Всего	***56***	***48***	***24***	***-***	***24***	***8***	

Рабочая программа разработана на основании составлена на основе Республиканского ГОС ВПО (14.05.12., № гос. рег. 116) в соответствии с требованиями к структуре и результатам освоения основных образовательных программ бакалавриата по «профессиональному» циклу (базовая часть) по направлению подготовки 710100 «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА» а также задачами, стоящими перед Ошским государственным университетом по реализации Программы развития ОшГУ.
(Для магистрантов)

Составитель: к.т.н., доцент _____Атырова Р.

2020-2021 – учебный год

– к.т .н., доц нт каф дры И МИ Ф шГУ, общий стаж работы - 15 л т, образовани – высш , закончила фак льт т Киб рн - тики и информационны т нологий ш У, 2002 г.; и ИПК по направл нию “Б галт рский ч т и а дит”, шГУ 2017 г.

Рабочий телефон: 03222-2-11-85,
Рабочее место: 723500. Главный корпус ОшГУ, ул. Ленина, 331, каб. 205.
Мобильный телефон: 0778-26-33-63
-mail: rahat.ars@gmail.com

Для I мод ля проводится 2 раза т к щ го контроля пров рки
Для II мод ля проводится 2 раза т к щ го контроля пров рки

Л кционны занятия проводится в л кционном класс (а д. 331,203), лабораторны занятия в компьют рны класса 305, 302, гд осваиваются навыки работы с различными пак тами программ.

Д ж рство пр подават ля проводится в а дитории 224 по пятницам с 8^{00} до 14^{00}.

Объяснительная записка

Курс входит в число дисциплин специализации, включенных в учебный план. Переход основных производителей процессоров на многоядерные архитектуры приводит к тому, что однопроцессорные системы стремительно исчезают. Программирование многопроцессорной системы связано с дополнительными как техническими, так и психологическими трудностями.

Для подготовки высокопрофессиональных программистов, архитекторов информационных систем и исследователей в различных областях информатики и прикладной математики необходима дополнительная подготовка по параллельному программированию. Данный курс ориентирован на обучение эффективному использованию и программированию современных высокопроизводительных компьютерных систем.

Цели курса. Освоение системы понятий параллелизма, выработка навыков использования параллельных компьютерных систем, параллельного вычисления. Формирование у магитрантов представления о месте параллелизма в современном программировании, о природе трудностей в параллельном программировании и способах их преодоления. Изучение языков и сред параллельного вычисления, освоение методов разработки, оптимизации и организации выполнения на компьютерных системах (КС) параллельных программ.

По завершению освоения данной дисциплины магитрант:

- получает знания о состоянии, технических характеристиках и применении сильно связанных КС (кластеров) и распределенных КС (локальных и глобальных компьютерных ресурсов) для решения сложных задач;
- получает знания о языках и средах параллельного вычисления различной ориентации;
- получает знания о методах и средствах, предназначенных для организации выполнения параллельных программ на КС и управления параллельными процессами;
- приобретает способность разрабатывать сложные параллельные программы для реальных задач и организовывать их эффективное выполнение на КС;

приобретает способность выполнять проектирование программных средств для повышения эффективности процессов проектирования параллельных программ и организации их выполнения на КС.

Задачи курса:

- Знание средств параллельного вычисления.
- Освоение создание параллельных программ для распространенных численных и получисленных задач.
- Освоение параллельных алгоритмов для задач теории графов, параллельная обработка дискретных структур.
- получение знаний об архитектурных особенностях современных КС, их технических характеристиках и программном обеспечении;
- освоение методов разработки параллельных алгоритмов и программ для ускорения процессов решения сложных задач (вычислительных задач, задач моделирования и распределенного управления) на КС.

Актуальность указанных знаний и навыков связана с развитием новых подходов в современном программировании к повышению производительности при решении сложных вычислительных задач.

Для успешного изучения курса от магитрантов требуются знания и навыки, получаемые из следующих курсов:

- Основы вычисления;
- Основы дискретной математики;
- Алгебра и геометрия;
- Архитектура вычислительных систем;
- Операционные системы.

Также рекомендуются знания и навыки, получаемые из следующих курсов:

- Математический анализ;
- Математическая логика и теория алгоритмов;
- Теория конечных графов и ее приложения;
- Алгоритмы и анализ сложности;
- Компьютерные сети.

1 семестр. Лекционный курс – 24 ч

№	Разделы	Количество часов
1.	Понятие параллельного вычисления. Многопроцессорные вычислительные комплексы. Распределенная и общая память.	1
2.	Основные парадигмы параллельного вычисления. Процессы и треды. Основные инструменты и методы для вычисления систем с общей и распределенной памятью.	1
3.	Библиотека MPI. Модель SPMD.Инициализация и завершение MPI-приложения.	1
4.	Точечные обмены данными между процессами MPI-программы.	1
5.	Режимы буферизации. Прием по шаблону и «условие гонок».	1
6.	Коллективные взаимодействия процессов в MPI.	1
7.	Управление группами и коммуникаторами в MPI	1
8.	Библиотека PVM. Управление виртуальной машиной. Создание и завершение процессов.	2
9.	Взаимодействие процессов в PVM. Функции запаковки и распаковки данных. Точечные обмены.	2
10.	Библиотека pthreads. Создание и завершение потоков.	2
11.	Проблема синхронизации доступа к общим данным. Семафоры. Критические секции.	2
12.	Основные компоненты среды OpenMP. Модель выполнения OpenMP-приложения. Директива parallel.	2
13.	Директивы для распределения работы в OpenMP.	2
14.	Директивы синхронизации в OpenMP.	2
15.	Языки параллельного вычисления. Преимущества и недостатки языков по сравнению с библиотеками для параллельного вычисления.	4
16.	Параллелизм на уровне заданий. Язык mpC.	

17.	Параллелизм по данным. Язык HPF	
18.	Вычисления, управляемы потоком данных. Язык Charm ++.	
	Итого:	**24 ч**

Лабораторные занятия

№	Темы и содержание лабораторных работ	Количество часов
1 семестр		**24**
1	Введение в параллельное программирование. Основные классы параллельных программ	2
2	Параллелизм в современных ОС. Виды параллелизма	4
3	Языки параллельного вычисления	4
4	Коллективные операции и их исполнение	6
5	Разработка сложных параллельных программ с использованием MPI.	8
	Итого:	**24 ч**

Самостоятельная работа

№	Темы и содержание	Количество часов
1.	Многопроцессорные архитектуры	2
2.	Сильно связанные КС	2
3.	Слабо связанные КС	2
4.	Средства параллельного вычисления	2
5.	Задачная и процессная форма задания параллелизма	2
6.	Модели параллельных процессов Р.Милнера	2
7.	Модели параллельных процессов Ч.Хоара	2
8.	Модели сетей Петри	2
9.	Критерии и параметры сложности параллелизма	2
10.	Распараллеливание последовательных программ	2
11.	Примитивы и средства для описания параллелизма	2
12.	Реализация на многопроцессорных КС	2
13.	Библиотека MPI	2
14.	Программирование на суперкомпьютерах с общей памятью	2
15.	Нитевое программирование	2
16.	Взаимодействие нитей и синхронизация	2
	Итого:	**32**

Контрольные вопросы

Архитектура и технические характеристики современных компьютеров и компьютерных систем (КС). Введение в параллельное программирование. Основные классы параллельных

программ. Параллелизм в современных ОС. Виды параллелизма. Семафоры. Свойства семафоров. Создание семафоров в Windows. Модель параллельных вычислений. Механизмы распределенного вычисления. Основные парадигмы. параллельного вычисления. Основы вычисления на MPI. Коллективные взаимодействия процессов в MPI. Параллельное программирование в системах OpenMP. Языки параллельного вычисления

№	Вопросы
1	Аппаратное обеспечение
2	Многопроцессорные архитектуры
3	Методы повышения быстродействия процессоров и памяти
4	Многопроцессорные КС
5	Кластеры
6	Параллельная программа
7	Виды взаимодействия процессов
8	Использование общей памяти
9	Обмен сообщениями
10	Многопоточные системы
11	Распределенные системы
12	Синхронные параллельные вычисления
13	Итеративный параллелизм
14	Рекурсивный параллелизм
15	Кластерные системы
16	Состояние параллельной программы
17	Свойства параллельных программ
18	Виды синхронизации
19	Взаимное исключение
20	Условная синхронизация
21	Метод активной блокировки
22	Алгоритм разрыва (Питерсона)
23	Семафоры
24	Свойства семафоров
25	Барьерная синхронизация
26	Функциональный параллелизм
27	Алгоритмы на графах
28	Распределенные вычисления
29	Модель SPMD
30	Механизмы распределенного вычисления
31	Различия механизмов распределенного вычисления
32	Обмен сообщениями
33	Канал
34	Синхронная передача
35	Удаленный вызов процедур (RPC)
36	Язык Occam
37	Треды
38	Процессы

39	Различие тредов и процессов
40	Интерфейс MPI
41	Цель разработки MPI
42	Функции MPI
43	Методы передачи данных в MPI
44	Коллективные операции в MPI
45	Управление процессами в MPI
46	Отладка MPI-приложений
47	Архитектура OpenMP
48	Директива parallel
49	Язык вычисления Esterel
50	Язык вычисления Lustre

Содержание лекционных занятий

Лекция № 1.
Введение в параллельное программирование. Основные классы параллельных программ.

Литература: Основная: []. Дополнительная: [].
Контрольные вопросы:
1. Аппаратное обеспечение;
2. Многопроцессорные архитектуры;
3. Организация компьютера;
4. Многопроцессорные КС.
5. Кластеры;
6. Понятие параллельное программирование;
7. Виды используемой памяти;
8. Виды взаимодействия процессов;
9. Основные классы параллельных программ;
10. Многопоточные системы;
11. Распределенные системы;
12. Синхронные параллельные вычисления.

Формы проверки знаний:
1. Опрос;
2. Проверка изученных терминологий;
3. Проверка выполнения лабораторных работ

Лекция № 2.
Параллелизм в современных ОС. Виды параллелизма
Литература: Основная: [2,3,6]. Дополнительная: [3].
Контрольные вопросы:
1. Процесс;
2. Задача (поток, нить);
3. Состояние параллельной программы;

4. Свойства параллельных программ;
5. Виды синхронизации;
1. Метод активной блокировки.
2. Общая память;
3. Распределенная память;

Формы проверки знаний:
1. Опрос;
2. Проверка выполнения лабораторных работ**.**

Лекция № 3.
Языки параллельного вычисления.
Литература: Основная: [1,5,6]. Дополнительная: [1,2].
Контрольные вопросы:
1.Язык вычисления Esterel;
2.Язык вычисления Lustre;
3.Язык вычисления Argos.
Формы проверки знаний:
1. Опрос;
2. Проверка выполнения лабораторных работ**.**

Основная литература:

1. Кутепов В.П. Параллельные системы и параллельное программирование. Электронная версия монографии, 2009, с. 362.
2. Воеводин Вал.В., Воеводин В.В. Параллельные системы и параллельные вычисления, Изд. «БХВ-Петербург», Санкт-Петербург, 2004, с. 599
3. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 2002.
4. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб: БХВ-Петербугр, 2002.
5. Богачев К.Ю. Основы параллельного вычисления. – М.:Бином, 2003.
6. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного вычисления. – М.:Вильямс, 2003.

Дополнительная литература:

1. Эндрюс Г.Р. Основы многопоточного и распределенного вычисления. Изд. Дом «Вильямс», 2003, с. 506.

2. Камерон Хьюз, Трейси Хьюз. Параллельное и распределенное программирование с использованием C++. Изд. дом «Вильямс», 2004, с. 668.

Электронные образовательные ресурсы:

1. http://www.parallel.ru
2. Электронная версия монографии: Кутепов В.П., «Параллельные системы и параллельные вычисления»
3. Диск «Электронная библиотека «Параллельное программирование». Абдугулова Г.С. Кафедра Вычисления»

Критерии оценки знаний магитрантов на экзамене

Оценка «**отлично**» ставится, если магитрант строит ответ правильно и логично в соответствии с планом, обнаруживает глубокое знание информационных терминов, понятий, концепций и теорий. Устанавливает содержательные междисциплинарные связи. Обоснованно аргументирует ответы на поставленные вопросы и задачи, приводит убедительные примеры. Обнаруживает аналитический подход в освещении различных теорий и концепций. Делает содержательные выводы. Пользуется современной литературой в предметной области.

Оценка «**хорошо**» ставится, если магитрант строит свой ответ в соответствии с планом. В ответе представлены различные подходы к поставленной задаче, их обоснование недостаточно полно. При необходимости ответ подтверждается примерами. Однако наблюдается некоторая непоследовательность анализа и обоснования. Выводы и решения вопросов правильны.

Оценка «**удовлетворительно**» ставится, если ответ недостаточно логически выстроен, план ответа отсутствует или соблюдается непоследовательность. Магитрант обнаруживает слабость в развернутом раскрытии классических понятий, хотя общее знание в области информатики присутствует, но недостаточно аргументируются. Магитрант испытывает трудности в установлении междисциплинарных связей в рамках естественных наук. Ответ носит преимущественно описательный, формальный характер.

Оценка «**неудовлетворительно**» ставится при условии неспособности магитранта правильно раскрыть базовые понятия и суть вопроса теории. Магитрант проявляет стремление подменить научное обоснование проблем рассуждениями обыденно-повседневного бытового характера. Ответ содержит ряд серьезных неточностей, нет решения примеров и задач.

Отметка *"Зачет" ставится если магитрант освоил объем знаний достаточный для использования его в дальнейшей своей деятельности по своей специализации и может в доступной форме изложить свои знания, как теоретические, так и практические*. Магитрантом выполнены все лабораторные задания и он может объяснить последовательность инструкций хода работы. Магитрантом получен тот объем знаний, который необходим ему для дальнейшего освоения курса других дисциплин по специальности. Также магитрант освоил и сдал в виде теста теоретический материал, как аудиторный, так и в виде самостоятельной работы.

Отметка *"Незачет"* ставится магитрантам, не выполнившим лабораторные задания и не освоившие минимум материала курса дисциплины.

Содержание

Интерактивные методы обучения

При **компетентностном подходе** в образовании главным фактором учебной деятельности является не столько компонент получения знаний, сколько компонент ***приобретения обучающимися различных способов деятельности***для решения поставленных образовательных задач. Поэтому для ***достижения ожидаемыхрезультатов обучения*** дисциплины необходимо использовать различные ***новые технологии и интерактивные методы.***

Интерактивное обучение – это, в первую очередь, ***диалоговое обучение,*** в процессе которого происходит как взаимодействие между студентом и преподавателем, так и между самими студентами. Интерактивные методы способствуют формированию **компетенций** и **достижению определенных результатов обучения** - получению знаний, формированию умений и навыков.

К методам интерактивного обучения относятся те, которые способствуют вовлечению в активный процесс получения и переработки знаний, например: лекция-визуализация (ЛВ), проблемная лекция (ПЛ), мини-лекция (МЛ), лекция – пресс-конференция (ЛПК), занятие – конференция (ЗК), тренинг (Тр), дебаты (Д), мозговой штурм (МШ), мастер-класс (МК), «круглый стол» (КС), активизация творческой деятельности (АТД), развитие критического мышления через чтение и письмо (КМ), регламентированная дискуссия (РД), дискуссия типа форум (Ф), деловая и ролевая учебная игра (ДИ, РИ), метод малых групп (МГ), занятия с использованием тренажёров, имитаторов (ТИ), компьютерная симуляция (КСим), использование компьютерных обучающих программ (КОП), интерактивных атласов (ИА), разбор клинических случаев (РКС), подготовка и защита истории болезни (ИБ), посещение врачебных конференции, консилиумов (ВК), участие в научно-практических конференциях (НПК), съездах, симпозиумах (Сим), учебно-исследовательская работа студента (УИРС), проведение предметных олимпиад (О), подготовка письменных аналитических работ (АР), подготовка и защита рефератов (Р), проектная технология (ПТ), экскурсии (Э), подготовка и защита курсовых работ (КР), дистанционные образовательные технологии (ДОТ), Тесты (Т), приглашение специалиста (ПС), выступление в роли обучающего (РО), разработка проекта (П), решение ситуационных задач (СЗ), презентации с использованием различных вспомогательных средств (През): интерактивная доска (ИД), раздаточные материалы (РМ), видеофильмы (В), слайды (С), мультимедийная презентация (МПрез), задания на самостоятельной работы, IT-метод (IT), работа в команде (РК), Case-study (метод конкретных ситуаций)-(КСт), поисковый метод (ПМ), исследовательский метод (ИМ) и др.

Интерактивные методы способствуют достижениюконкретных ***результатов обучения*** и формированию компетенций:

- способствуют эффективному усвоению учебного материала;
- формируют умения формировать и отстаивать свою позицию и точку зрения;
- способствуют формированию организационных и деловых навыков;
- формируют навыки анализа, сравнения, критического мышления;
- формируют умения выдвигать новые идеи, т.е создавать;
- способствуют формированию навыков лидерства и работы в команде, коммуникации, сотрудничества:
- формируют умения прогнозировать события;
- формируют навыки творчества, поисковой и исследовательской работы;
- формируют навыки оценки и самооценки, применения знаний на практике;
- формируют умения определять проблему и предлагать пути ее решения;
- формируют умения проектирования, планирования и др.

Лекционные материалы

Основы параллельных вычислений
Учебное пособие

Электронная версия учебного пособия

Содержание

1. ВВЕДЕНИЕ

Учебное пособие содержит основной понятийный материал области параллельных вычислений и, в частности, параллельного программирования суперЭВМ. Параллельные вычисления сейчас быстро развиваются, что, прежде всего, обусловлено появлением разнообразных коммерческих суперкомпьютеров и ростом числа их приложений.

Суперкомпьютеры позволили начать решать новые, ранее недоступные задачи большого размера, приступить к реализации математических моделей сложных физических явлений и технических устройств. Здесь суперкомпьютеры являются незаменимым и универсальным технологическим инструментом как в изучении природы, так и в практической деятельности. Понятна привлекательность изучения ядерного взрыва на модели, а не в натуре. Без всякого преувеличения можно сказать, что новые технологии ближайшего будущего станут разрабатываться на основе изучения математических моделей реальных процессов. Именно по этой причине американское правительство наложило строгие запреты на продажу суперкомпьютеров не только в Россию, но и в любые другие страны!

Использование суперкомпьютеров наталкивается на большие трудности. Очень сложно создавать и отлаживать параллельные программы, гораздо сложнее последовательных. В самых простых с виду параллельных программах обнаруживаются иной раз фатальные ошибки.

Известны программы, состоящие буквально из десятка операторов, которые долгое время считались правильными, их правильность даже умудрялись доказать формально. И тем не менее со временем в них обнаруживались тяжелые ошибки! А ведь среди приложений суперкомпьютеров есть много и таких, которые очень чувствительны к ошибкам (по своим последствиям). Отсюда значительно больший, чем в последовательном программировании, интерес к автоматизации конструирования параллельных программ, к строгим математическим методам конструирования и анализа правильности программ. Сейчас нет уже сомнений, что без основательного знания математики невозможно стать хорошим параллельным программистом, а значит, и программистом вообще, поскольку и сейчас уже, по существу, нет непараллельных ЭВМ. Даже персональные ЭВМ в значительной степени используют эффект параллелизма. А в ближайшем времени ожидается массовый переход на использование многопроцессорных ЭВМ.

В 1996 г рынок продаж мультикомпьютеров вырос на 43 процента. Незнание математики существенно ограничивает возможности понимания поведения системы параллельно протекающих вычислительных процессов. Инженеры, не имеющие подготовки в математике, будут попросту не в состоянии понять устройство параллельного программного обеспечения, и в таком случае их труд может свестись только лишь к "нажиманию кнопок".

Теория параллельных вычислений характеризуется большим разнообразием идей, подходов к решению задач, и к разработке параллельных программ. Очень многие задачи требуют учета их особенностей для создания эффективных программ, их решения. Это, в свою очередь, приводит к разработке разнообразных и многочисленных средств параллельного программирования, что еще более затрудняет использование суперкомпьютеров.

Ключ к успеху в приложениях суперкомпьютеров, в решении задач большого размера - эффективные параллельные вычислительные технологии. Они призваны значительно сократить путь от теоретических и экспериментальных результатов к их приложениям. Фактически речь идет о создании технологий реализации математических моделей как физических явлений, так и интеллектуальной деятельности человека, что позволит затем использовать эти модели в необходимых конкретных приложениях. Сами вычислительные технологии, таким образом, являются метатехнологиями, т.е. средством разработки новых прикладных технологий. Это далеко не тривиальная задача для теоретиков и программистов.

Перечисленные обстоятельства определили потребность в параллельных вычислениях в различных дисциплинах. Нынешние "последовательные" математические модели часто искусственно были ограничены в силу необходимости их именно последовательной реализации. С появлением параллельных суперЭВМ стала актуальной и задача создания параллельных алгоритмов и моделей в самых разнообразных предметных областях и научных дисциплинах.

Курс сопровождается семинарскими занятиями и лабораторными работами, цель которых - привить первые навыки программирования параллельных высокопроизводительных суперЭВМ. На семинарских занятиях рассматриваются приемы программирования вычислительных систем с использованием 1-2 языков параллельного программирования, на лабораторных - разрабатываются и отлаживаются программы, решаются конкретные задачи на мультикомпьютерах.

2. ПОНЯТИЕ ВЫЧИСЛИМОЙ ФУНКЦИИ

2.1. Введение

Мы начнем с обсуждения понятия вычислимой функции, которое необходимо далее для формирования правильной исходной позиции при анализе, конструировании и размышлениях о программах. Существует много разных формализаций понятия вычислимой функции (в [6] рассматривается 12 различных формализаций понятия "алгоритм"). Мы будем обсуждать формализацию Клини и следовать изложению [9, 13], как наиболее соответствующих нашим целям. За дополнительной информацией можно обратиться к [15].

Понятие вычислимой функции - ключевое понятие из тех, которыми должен владеть программист и вообще каждый, работающий в области вычислений. Задача этой главы - дать начальные понятия теории алгоритмов.

Говоря о программе, мы всегда имеем в виду всевозможные способы ее исполнения, включая частный случай - последовательное исполнение. Термины "параллельная/последовательная программа" используются, чтобы подчеркнуть предпочтительный/единственно допустимый способ исполнения программы.

Многочисленные примеры алгоритмов известны каждому программисту. Интуитивно ясно, какие процессы могут быть отнесены к алгоритмам. Однако на интуитивном уровне понятие алгоритма остается неуловимо понятным и одновременно совершенно непонятным. В [15] авторы на стр 13 утверждают, что "... в теории алгоритмов ... открытия состоят не столько в получении новых результатов, сколько в обнаружении новых понятий и в уточнении старых." Каждый легко может определить, является ли некоторый предъявленный процесс алгоритмом или нет. Все, по-видимому, согласятся, что вычисление корней квадратного полинома по формуле Виета есть алгоритм. Однако без формализации понятия алгоритма невозможно показать, что не существует алгоритмического решения проблемы.

Для построения формализации понятия алгоритма можно сначала просмотреть ряд примеров алгоритмов и попробовать выделить неформальные свойства любого процесса, претендующего быть алгоритмом. Затем можно построить формализацию, удовлетворяющую неформальным свойствам. Так, собственно, строится любая математическая теория.

Исходя из подобных примеров, Мальцев [9] перечисляет следующие неформальные свойства алгоритмов:

a. Алгоритм - это процесс последовательного построения величин, идущий в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент новая конечная система величин получается по определенному закону (программе) из системы величин, имеющихся в предыдущий момент времени (*дискретность алгоритма*).

В последовательной программе операторы исполняются, вычисляя шаг за шагом новые значения из предшествующих.

b. Система величин, полученная в какой-то (не начальный) момент времени, однозначно определяется системой величин, полученных в предшествующие моменты времени (*детерминированность алгоритма*).

Программа, исполняясь несколько раз с одними и теми же входными данными, должна всегда выработать один и тот же результат.

c. Закон получения последующей системы величин из предшествующей должен быть простым и локальным (*элементарность шагов алгоритма*).

d. Если способ получения последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указано, что надо считать результатом алгоритма (*направленность алгоритма*).

Если программа не может вычислить нужный результат, она должна сообщить об ошибке, описать ее и подсказать, как ее исправить. Это и есть результат работы программы в таком случае. Но программа не должна зацикливаться без выдачи какого-либо результата!

e. Начальная система величин может выбираться из некоторого потенциально бесконечного множества (*массовость алгоритма*).

Другое не строгое и более "машинное" описание понятия алгоритма приводит Роджерс [13].

1. Алгоритм задается как конечный набор инструкций конечных размеров. Любой алгоритм может быть описан словами, например, задан как программа на языке С.

2. Имеется вычислитель...., который умеет обращаться с инструкциями и производить вычисления.

3. Имеются возможности для выделения, запоминания и повторения шагов вычисления.

4. Пусть Р - набор инструкций из I, L - вычислитель из 2. Тогда L взаимодействует с Р так, что для любого данного входа вычисление происходит дискретным образом по шагам, без использования аналоговых устройств и соответствующих методов.

5. L взаимодействует с Р так, что вычисления продвигаются вперед детерминированно, без обращения к случайным методам или устройствам. В частности, следующий исполняемый оператор программы не может выбираться жребием.

Это неформальное определение алгоритма уточняется ответами на следующие вопросы.

6. Следует ли фиксировать конечную границу для размера входов (начальная система величин в определении Мальцева)?

Ответ нам очевиден из программирования - нет. И в самом деле, если входной величиной является файл (стандартная ситуация), его длина программе неизвестна и она работает с потенциально бесконечным входом, т.е. имеется в виду, что каждый

раз обрабатывается конечная строка (мы говорим о файле, как о строке, потому как строкой является и любая последовательность битов, составленная, например, из записей файла), однако для любого наперед заданного натурального N найдется строка длиной больше, чем N.

7. Следует ли фиксировать конечную границу для размера памяти?

Ответ тоже очевиден - нет. Если мы станем вычислять значение полинома второй и более степени при неограниченной длине входа (числа символов, необходимых для представления входных значений), то ясно, что для промежуточных вычислений необходимо иметь неограниченную память (ее потребный объем зависит от длины входа). И реально программисты часто сталкиваются с ситуацией, когда программа, скажем, умножения матриц размера n_0 не может исполняться из-за недостатка памяти и матрицы размера $n_0 - 1$ умножаются благополучно.

Можно задавать и другие вопросы, но для нас этого достаточно. На часть не заданных вопросов ответы будут получены в ходе построения формального понятия алгоритма.

Этих не строгих определений на практике достаточно и для того, чтобы понять, является ли процесс алгоритмом, и для построения нужного алгоритма. Однако его совершенно недостаточно для ответа на вопрос, существует ли алгоритм решения некоторой проблемы? Для доказательства существования алгоритма решения проблемы достаточно построить такой алгоритм. Здесь годится и неформальное определение. Но чтобы доказать отсутствие такого алгоритма, надо для начала точно знать, что такое алгоритм.

Каждый алгоритм А вычисляет значение функции F_A при некоторых заданных значениях входных величин. Функции, вычисляемые некоторым алгоритмом, называются *вычислимыми (алгоритмически вычислимыми)* функциями. Понятие вычислимой функции, так же как и понятие алгоритма, тоже здесь интуитивно. Существует много разных формализаций понятия алгоритм, удовлетворяющих неформальным требованиям. Однако класс вычислимых функций, определенный во всех таких формализациях, - класс рекурсивных функций - оказался одним и тем же при самых разнообразных попытках расширить понятие алгоритма. Черч высказал гипотезу - *тезис Черча,* - что класс всех вычислимых функций совпадает с классом рекурсивных функций. Так как понятие вычислимой функции определено неформально, то и тезис Черча строго недоказуем. Принимая тезис Черча, доказательство вычислимости функции сводим к доказательству ее рекурсивности. Мы принимаем тезис Черча и используем термины вычислимой и рекурсивной функции наравне.

Пост и Тьюринг для уточнения понятия алгоритма построили точно описанные в математических терминах "машины". Несмотря на предельную тривиальность этих машин на них оказалось возможным определить (выполнить) все алгоритмические процессы, когда-либо рассматриваемые в математике.

2.2. ОСНОВНЫЕ ПРЕДВАРИТЕЛЬНЫЕ ПОНЯТИЯ

Задача этого раздела - обсудить ряд необходимых понятий и свести рассмотрение к классу только числовых функций.

2.2.1. Алфавит

Алфавитом называется конечное множество символов A = (a,b,...,c). Конечная последовательность символов алфавита А называется *словом* в алфавите А. Количество символов в слове называется *длиной* слова. Например, если A = {a,b}, то ab, aa, abbaa, bbbbba - слова в алфавите А. Множество всех слов алфавита А обозначается A^*.

Если s_1 и s_2 - слова, то слово s_1s_2 называется *конкатенацией (произведением)* слов s_1 и s_2. Пустое слово не содержит символов, его конкатенация с любым словом s равна слову s. Слово s_2 называется *подсловом* слова s, если s представимо в виде $s = s_1s_2s_3$, s_1 и s_3 - тоже слова в алфавите A, возможно пустые. Слово $s_1s_2s_3$ называется разложением слова s, s_1 - *префикс* разложения $s_1s_2s_3$.

Возможны несколько различных разложений слова s, содержащих подслово s_2. Слово s_2 называется первым вхождением s_2 в s, если префикс s_1 обладает наименьшей длиной среди всех таких разложений слова s. Например, слово abaabaaba = a · ba · abaaba = abaa · ba · aba=abaabaa · ba имеет несколько разных разложений с подсловом ba, первое из них содержит первое вхождение ba к слово abaabaaba.

2.2.2. Кодирование

В теории всегда рассматриваются вычислимые функции, отображающие одни множества слов в другие. Множества слов в некотором алфавите выбраны в силу универсальности этих множеств для представления величин. Все прочие виды величин кодируются и представляются словами.

Рассмотрим алфавит A = {1}. Тогда натуральное число n может быть представлено словом 111...1, содержащим n единиц, а само слово 111 ... 1 называется *кодом* числа n . Если использовать алфавит A = {0,1}, то всем программистам известно, как кодируются целые и вещественные числа, а также и символы различных алфавитов. Слово 11 есть код числа 3, а слово 011, вообще говоря, не является кодом никакого числа, если специально не оговорить случай незначащих предшествующих нулей (что и сделано во всех компьютерах). Таким образом, при кодировании возможны несколько различных кодов одной и той же величины и возможны коды, не представляющие никакой величины. Однако всегда по заданному коду закодированный объект должен восстанавливаться однозначно.

Понятно, что при изучении вычислимых функций можно ограничиться рассмотрением только *числовых* функций (функций, определенных на множестве натуральных чисел N, со значениями в N), предполагая наличие подходящей кодировки. В этой главе мы ограничимся рассмотрением только числовых функций.

Пусть A - алфавит. A^* - множество всех слов в алфавите A, $B \subseteq A$, M - множество некоторых объектов. Закодировать объекты из M в алфавите A означает задать однозначное соответствие $f: B \to M$. Взаимно однозначного соответствия не требуется. Слово $b \in B$, $b = f(m)$, $m \in M$, называется *кодом* или именем объекта m в кодировании f.

2.2.3. Бесконечный алфавит

Иногда удобно использовать бесконечные алфавиты вида $A = \{a, b, ..., c, d_i, t_i^j\}$. Такой алфавит формально бесконечен, так как содержит бесконечное число символов d_i, t_i^j. Однако он легко кодируется в конечном алфавите $A_o = \{a, b, ... , c, a, t, n, m\}$. При этом символы d_i кодируются словами dnn...n, а символы t_i^j - словами tnn...nmm...m, в которых символ n повторяется i раз, а символ m - j раз. Таким образом, всегда можно оставаться в конечных алфавитах даже при использовании такого сорта бесконечных алфавитов.

2.2.4. Наборы (кортежи)

Пусть задан алфавит, $B = A \cup \{,\}$, где "," - специальный выделенный символ. Кроме слов в A придется далее рассматривать наборы слов вида $<s_1,s_2><s_1,s_2,s_3>$ и т.д. Такие наборы слов алфавита A кодируются словами в алфавите B, при этом слово s_1s_2 кодируется словом $<s_1,s_2>$, а слово $s_1s_2s_3$ - словом $<s_1,s_2,s_3>$.

2.2.5. Термы

Термы (функциональные термы) - это слова особого вида, записанные в функциональном алфавите. Функциональный алфавит состоит из трех групп символов, используемых для записи функций. Первую группу составляют *предметные* символы (*предметные переменные*), изображающие переменные. Для их обозначения будут использованы строчные буквы x, y, z, d или эти же символы с индексами. Вторую группу составляют *функциональные* символы, изображающие функции. В качестве функциональных символов будем использовать буквы f,g,h или эти же символы с верхними и нижними индексами. Верхний индекс обычно указывает арность (местность) функционального символа. Третью группу составляют специальные символы " (",")", ",".

Понятие терма определяется шагами по индукции. Сначала определяются термы длины l, т.е. слова длины l в функциональном алфавите, которые есть термы по определению. Термами длины l называются односимвольные слова из предметных символов. Далее пусть для некоторого k > l термы длины меньше k уже определены.

Тогда слово t длины k называется термом, если оно имеет вид $f(t_1, t_2, \ldots, t_m)$, где $t_1, t_2, \ldots, t_m$ - термы длины меньшей, чем k, f - функциональный символ. Пусть, например, дан функциональный алфавит $A = \{x, y, a\} \cup \{f, g^2\} \cup \{(,), ,\}$. Тогда слова $f(x)$, $g^2(y, a)$, $g^2(f(x), g^2(y, a))$ есть термы. На рис 2.1. показано графическое представление последнего терма.

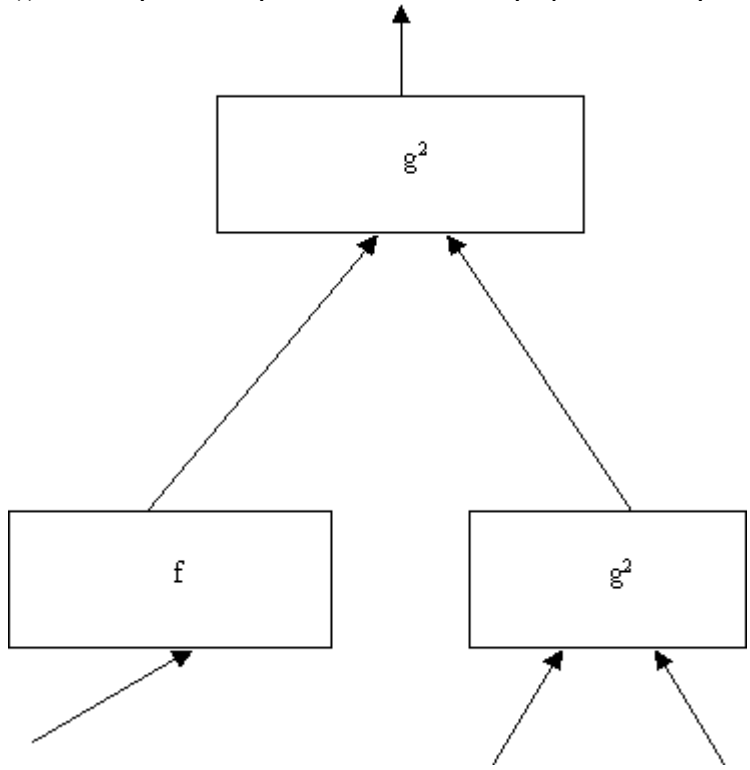


Рис. 2.1.

Заметим, что понятие терма введено чисто синтаксически безотносительно к понятиям функции, переменные и т.п. Просто есть алфавит, символы которого разбиты на три подмножества, и слова особого вида названы термами. Сейчас будет введено понятие интерпретации, которое только и придаст смысл терминам "предметные и функциональные символы".

Интерпретацией I называется отображение, которое ставит в соответствие:

- каждой предметной переменной x - элемент $I(x) = d_x \in N$, элемент d_x множества N называется *значением переменной x,* N - множество значений переменной x, N называется *основным* множеством или *областью интерпретации*;
- каждому k-местному функциональному символу f^k - частичную функцию $I(f^k) = F^k : N \times N \times ... \times N \rightarrow N$ Функция F^k называется *значением функционального символа* f^k;
- каждому терму $t = f^k(x1, x2, ... , xk)$ - значение терма $I(t) = d_e \in N$, $d_e = val(f^k(x1, x2, ... , xk))$. Функция val (от английского value - значение) задана на множестве термов, она определяется по индукции:
  - $val(x)=I(x)=d_x \in D_x$;
  - пусть $t = f^k (t1, t2, ... , tk)$ - терм длины n, термы t1, t2, ... , tk длины меньшей, чем n и функция val определена для термов длины меньшей, чем n. Тогда $val(t) = val(f^k (t1, t2, ... , tk)) = F^k(val(t1), (val(tk),..., (val(tk))$, $F^k=I(f^k)$ - значение функционального символа f^k.

Таким образом, в качестве значения терму $t = f^k (t1, t2,..., tk)$ ставится в соответствие то значение функции F^k, которое она имеет при значениях ее аргументов, равных val(t1), (val(tk),..., (val(tk). Например, если I(f)= F и $I(g) = G^2$?

Тогда

$$val(x)=d_x,\ val(y)=d_y,\ val(f(x))=F(val(x))=F(d_x),$$
$$val(g(x,y)) = G^2(val(x), val(y))= G^2(d_x, d_y),$$
$$val(g(f(x), g(x,y)) = G^2(val(x), val(g(x,y)))=$$
$$G^2(F(val(x)), G^2(val(x),val(y))) = G^2(F(d_x), G^2(d_x, d_y)).$$

Понятно, что функциональный терм задает новую функцию, определенную как комбинацию заданных функций. Если эти функции частичны, то и терм определяет частичную функцию.

Такого определения понятия *терм* достаточно в этой главе. Однако оно мало пригодно для целей программирования и будет далее уточняться.

2.3. ПОНЯТИЕ РЕКУРСИВНОЙ ФУНКЦИИ

Теперь можно приступить к определению понятия частичной рекурсивной функции. Идея состоит в том, чтобы сначала выбрать множество простейших рекурсивных функций, вычислимых "по определению". Они выбираются таким образом, чтобы в их вычислимости не было сомнений.

Затем определяются операторы - схемы конструирования новых функций. Каждый оператор будет определен таким образом, что в его вычислимости тоже не возникнет сомнений. Применение этих операторов к вычислимым функциям вновь должно дать вычислимую функцию. Класс построенных таким образом функций называется классом частично рекурсивных функций.

2.3.1. Простейшие вычислимые функции

Итак, на первом шаге выбирается счетный базис простейших рекурсивных функций, вычислимых по определению. Это следующие функции:

1. функция следования $s(x)= x+1$;
2. нулевая функция $o(x)= 0$;
3. функции выбора $Imn(x1,x2,...,xn)= xm$, $m \leq n$.

2 3.2 Суперпозиции частичных функций

Первым рассмотрим оператор суперпозиции частичных рекурсивных функций. Пусть заданы n частичных функций m переменных fim: D1 × D2 × ... × Dm → Do, i = 1, 2, ... , n, и пусть определена функция fn : Do × Do × ... × Do → D. Напомним, что в качестве областей определения и значения функций в главе всегда берутся либо множество натуральных чисел N, либо его подмножества. Следовательно, D, Do, D1, D2, ... , Dm ⊆N. Определим функцию m переменных gm, заданную на множестве D1 × D2 ×... × Dm со значениями в D, gm(x1, x2, ... ,xm) = fn (f1m(x1, x2, ... ,xm),..., fnm (x1, x2, ... ,xm)). Функция gm получается суперпозицией (подстановкой) из функций fn, f1m, fnm. Оператор суперпозиции будем обозначать Sn+1. Например, функция S3 (I12(x1, x2), I34(x1, x2, x3, x4), I24(x1, x2, x3, x4)) = I14(x1, x2, x3, x4).

Понятно, что функция gm (x1, x2, ... ,xm) и есть та функция, которую определяет функциональный терм fn (f1m(x1, x2, ... ,xm),..., fnm (x1, x2, ... ,xm)) при соответствующей интерпретации (рис. 2.2).

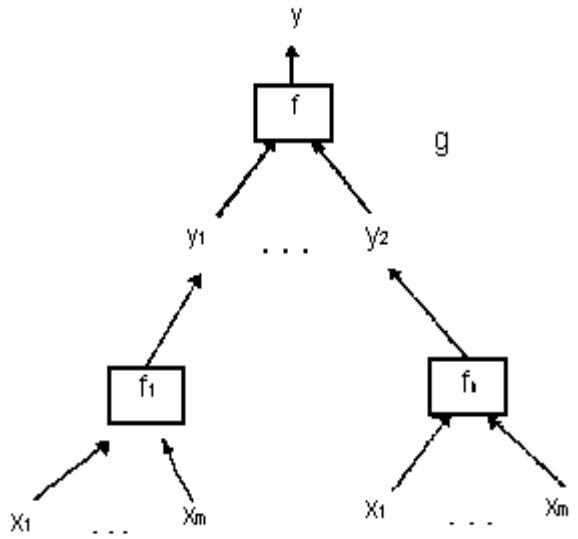


Рис. 2.2

Таким образом, функция g = Sn+1(f, f1,..,fn) определятся/вычисляется функциональным термом (рис. 2.2). И наоборот, функциональному терму соответствует применение некоторого оператора суперпозиции. Переменные y1, y2, .. ,ym представляют промежуточные величины.

Пусть φn - множество всех частичных вычислимых функций от n переменных. Тогда оператор суперпозиции Sn+1 может рассматриваться как всюду определенная функция Sn+1 : φnxφmxφmx...xφm → φm. Теперь в этих терминах можно перефразировать понятие функции, полученной применением оператора суперпозиции, а именно:

функция g(x1, x2, ... ,xm) получается суперпозицией из функций f(x1, x2, ... ,xn), f1(x1, x2, ... ,xm),..., fnm (x1, x2, ... ,xm) , если g есть значение операторного терма Sn+1 (h, h1, ..., hn), где h, h1, ..., hn - переменные и интерпретация ставят в соответствие:

- переменной h - значениеf, элемент из φn,
- переменным h,h1, ..., hn - значения fi из φm соответственно, i = 1,2,..., n,
- функциональному символу Sn+1 - функцию f(f1(x1, x2, ... ,xm),..., fn (x1, x2, ... ,xm))= g(x1, x2, ... ,xm).

Таким образом, можно использовать либо операторную, либо термальную запись представления функции g. В первой из них g есть значение операторного терма, во второй g определяется как функциональный терм. Этот функциональный терм и определяет алгоритм вычисления функции g.

Например, значением операторного терма S3 (I12 (x1, x2), I34(x1, x2, x3, x4), I24(x1, x2, x3, x4)) является функция I12(x1, x2, x3, x4) (здесь в операторный терм уже подставлены значенияI12 (x1, x2), I34(x1, x2, x3, x4), I24(x1, x2, x3, x4) переменных.

Возьмем функцию x1*(x2+x3) в термальном представлении, в качестве функций использованы обычные функции сложения + и умножения *. Ее операторное представление есть S3 (x, I13, S3(+, I23, I33).

Программисты легко могут представить себе наличие процедур, вычисляющих простейшие вычислимые функции. Оператор суперпозиции определяет с позиций программирования, простое "сцепление" процедур, при котором одни процедуры вырабатывают значения своих выходных переменных, а другие их используют в качестве входных величин. Понятен и "механический" характер определения оператора суперпозиции, его очевидная вычислимость, а именно: если известно, как вычислить значения функций f, f1,...,fn, то понятен и алгоритм вычисления функции g(x, x1,...,xm)=f(f1(x1, x2,...,xm),..., fn(x1, x2,...,xm)).

Каждый программист легко узнает в нижеследующей программе "реализацию" оператора суперпозиции. Слово "реализация" взято в кавычки, чтобы подчеркнуть его неформальность. Позднее этот термин будет точно определен.

Пусть процедуры f, f1,...,fn вычисляют одноименные функции. Тогда программа P вычисляет функцию g.

P : y1:= f1(x1, x2,...,xm);
.....
yn:= fn(x1, x2,...,xm);
y:= f(y1, y2,...,ym).

Если применить оператор суперпозиции конечное число раз, то будет построено конечное число функциональных термов. Таким образом, конечным числом применения оператора суперпозиции к простейшим функциям могут быть построены только вычислимые функции.

2.3.3. Оператор примитивной рекурсии

Оператор примитивной рекурсии устроен более сложно, чем оператор суперпозиции. Он позволяет определить циклические вычисления специального вида.

Пусть заданы n-местная вычислимая функция g и (n+2)-местная вычислимая функция h. Тогда (n+1)-местная функция f строится примитивной рекурсией из функций g и h, если для всех натуральных значений x1, x2, ..., xm, y имеем:

f(x1, x2, ..., xn,0) = g(x1, x2, ..., xn),

(1)

f(x1, x2, ..., xn,y+1) = h (x1, x2, ..., xn,y, f(x1, x2, ..., xn, y)).

Если n = 0, то одноместная функция f строится примитивной рекурсией из функции-константы Cnst (это просто некоторое натуральное число) и двухместной функции h

f(0)= Cnst, f(x+l)= h(x,f(x1)).

Соотношения (1) называются схемой примитивной рекурсии. Они определяют оператор примитивной рекурсии R. Эта схема, собственно, и есть алгоритм вычисления функции f. Оператор определен на множестве φ частичных вычислимых функций, f = R(g,h).

Понятно, что функция f существует и единственна. Это следует из того, что определение функции, построенной примитивной рекурсией, фактически содержит схему (алгоритм) ее вычисления. Распишем детально эту схему, используя термальные представления.

Пусть необходимо вычислить значение функции f при y = k. Тогда из определения (1) схемы примитивной рекурсии имеем последовательность функциональных термов, вычисляющих значение функции f(x1, x2, ... ,xn, k):

t_o: $f_o=f(x_1, x_2, ... ,x_n, 0) =g(x_1, x_2, ... ,x_n)$;

t_1: $f_1=f(x_1, x_2, ... ,x_n, 1) =h(x_1, x_2, ... ,x_n, 0, g(x_1, x_2, ... , x_n))$;

t_2: $f_2=f(x_1, x_2, ... ,x_n, 2) =h(x_1, x_2, ... ,x_n, 1, f(x_1, x_2, ... , x_n, 1))$;

.....

t_k: $f_k=f(x_1, x_2, ... ,x_n, k) =h(x_1, x_2, ... ,x_n, k-1, f(x_1, x_2, ... , x_n, k-1))$;

Здесь представлен способ вычисления f, следовательно, функция f определена однозначно. Если одна из функций f(x1, x2, ... ,xn, m), m1, x2, ... ,xn,y) для всех y>m тоже не определено. Эту последовательность k+1 функциональных термов to, t1,... удобно для наглядности изобразить графически (рис. 2.3).

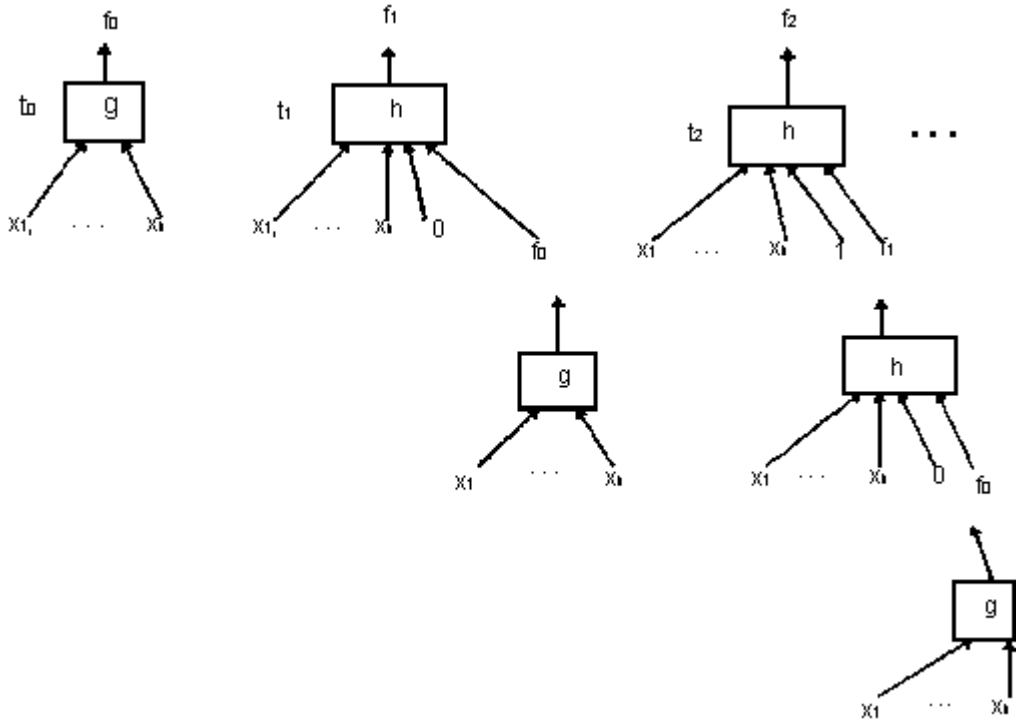


Рис. 2.3

Это множество функциональных термов и определяет, собственно, алгоритм вычисления функции f. Множество термов (представление алгоритмов в виде множества термов) позволяет легко построить программу P, реализующую оператор суперпозиции и вычисляющую функцию f.

P : s:=k-1;

```
f[0]:= g(x1, x2, ... ,xn);
for i=1 to s do
   f[i]:= h(x1, x2, ... , xn, i-1, f[i-1]);
   f:=f[k-1].
```

Программисты записывают обычно P более экономно, но не "функционально":

```
P1 : s:=k-1;
   f:= g(x1, x2, ... ,xn);
   for i=1 to s do
      f:= h(x1, x2, ... , xn, i-1, f);
```

Отметим следующие важные обстоятельства этого определения:

1. Если мы умеем вычислять функции g и h, то значение функции f(x1, x2,..., xn, k) вычисляется опять же "механически", т.е. алгоритмом, таким образом, следует признать функцию f вычислимой.
2. Для любого натурального числа k значение функции f(x1, x2,..., xn, k) задается/вычисляется k функциональными термами (рис. 2.3).
3. Так как число k - любое натуральное число, то функция f, полученная применением оператора примитивной рекурсии к функциям g и h, может быть определена потенциально бесконечным множеством функциональных термов (рис. 2.3) особого вида. Перефразируя, можно сказать, что функция f может быть построена потенциально бесконечным применением оператора суперпозиции.
4. В программе оператор примитивной рекурсии реализуется циклом типа for. Границы изменения параметра цикла, понятно, должны быть определены до начала выполнения цикла. Это характерное свойство примитивно рекурсивных функций: значения входных величин заранее не ограничены, однако они должны принять конкретное конечное значение до начала исполнения цикла.

Функция f называется примитивно рекурсивной относительно множества простейших функций, если она строится из простейших вычислимых функций конечным числом применения операторов суперпозиции и примитивной рекурсии.

Если расширить множество простейших функций добавлением в него примитивно рекурсивных функций, то, применяя конечное число раз операторы примитивной рекурсии и суперпозиции к функциям расширенного множества, можно будет построить только примитивно рекурсивные функции.

Если расширить множество простейших функций множеством всюду определенных примитивно рекурсивных функций, то, применяя конечное число раз операторы примитивной рекурсии и суперпозиции к функциям расширенного множества, можно будет построить только примитивно рекурсивные функции. В частности, все примитивно рекурсивные функции всюду определены.

П1. По определению функции $s(x)=x+1$, $o1(x)=o(x)=0$ $Imn(x1, x2,..., xn) = xm$, $m < n$ являются примитивно рекурсивными функциями. Функция $on(x1, x2,..., xn)=0$ представляется операторным термом $on=S2(o1, I1n)$. Следовательно, функция $o''(x1, x2,..., xn)= 0$ примитивно рекурсивна.

П2. Рассмотрим примитивно рекурсивную схему

$x + 0 = x = I11(x)$,

$x + (y+1) = (x+y) + 1 = s(x+y)$.

Следовательно, функция $(x+y)$ строится применением оператора примитивной рекурсии R к примитивно рекурсивным функциям $g(x) = I11$ и $h(x,y,z) = z + 1$. Следовательно, функция $x+y$ примитивно рекурсивна.

ПЗ. Аналогично, функция x · y удовлетворяет следующей схеме примитивной рекурсии

x · 0 = o(x),

x ·(y+1) =xy+1

с o(x) в качестве функции g и h(x, y, z) = z + x. Следовательно, функция x · y примитивно рекурсивна.

Следует различать понятия алгоритмически вычислимой функции и алгоритма, а именно: пусть существует некоторая нерешенная математическая проблема. Определим функцию

$$h(x) = \begin{cases} 1, \text{ если проблема решается положительно,} \\ 0, \text{ если проблема решается отрицательно.} \end{cases}$$

Ясно, что функция h(x) - константа (либо 0, либо 1) и, следовательно, примитивно рекурсивна. Однако алгоритм (примитивно рекурсивная схема) ее вычисления неизвестен, поскольку функции h(x) = 0 и h(x) = 1 вычисляются разными алгоритмами и мы не можем сделать правильный выбор.

Как оказалось, класс примитивно рекурсивных функций не исчерпывает всех вычислимых функций. Нашлись с очевидностью алгоритмически вычислимые функции, для которых показано не существование примитивно рекурсивных схем. Одна из таких функций - функция Аккермана:

$$f(0,x,y) = y + x,$$

$$f(1,x,y) = y*x,$$

$$f(2,x,y) = y^x,$$

.....

Таким образом, необходимо расширить класс примитивно рекурсивных функций в попытке определить класс всех вычислимых функций.

2.3.4. ДИАГОНАЛИЗАЦИЯ

Насколько, однако, далеко можно строить расширения?

Прежде всего заметим, что для определения класса примитивно рекурсивных функций использовалось конечное число символов: символы простейших функций, вспомогательные символы типа скобок "(" и ")", запятой ",", равенства =, символ конца строки.

Далее всякая примитивно рекурсивная функция строится конечным числом применения операторов суперпозиции и примитивной рекурсии. Следовательно, определение каждой примитивно рекурсивной функции задается в выбранном формализме конечной строкой символов.

Если задана строка символов, то можно построить алгоритмическую процедуру для распознавания, определяет ли строка схему (1) примитивной рекурсии или нет.

Рассмотрим теперь метод диагонализации Кантора в применении к процессу конструирования примитивно рекурсивных функций.

Мы можем последовательно перечислять все примитивно рекурсивные схемы: сначала анализируются все строки длины 1 и отбираются те, что определяют примитивно рекурсивные схемы. Затем так же исследуются все строки длины 2 и т.д. Каждое множество

строк длины k конечно (оно может быть упорядочено произвольным образом, например лексикографически), и для каждого k такой анализ закончится за конечное время (за конечное число шагов). При анализе распознанные примитивно рекурсивные схемы последовательно нумеруются.

Пусть теперь Gx есть схема с номером x, а gx - примитивно рекурсивная функция, определяемая этой схемой. По аналогии с методом диагонализации Кантора определим функцию h:

h(x)=gx(x)+1.

Ясно, что функция h(x) вычислимая. Для любого натурального x в качестве значения функции h(x) берется значение функции gx(x), вычисляющейся схемой Gx. Ясно также, что функция h(x) не примитивно рекурсивна. Если предположить ее примитивную рекурсивность, то тогда существует такое x0, что h(x0)= gx0(x0), т.е. при перечислении будет выбрана схема Gx0, определяющая функцию h(x). При этом из определения функции h(x) сразу возникает противоречие:

h(x0)=gx0(x0)=gx0(x0)+1. (2)

Попытки расширить формально определенный класс вычислимых функций выглядят теперь бесперспективными. К каждому вновь определенному классу вычислимых функций можно будет применить процедуру диагонализации и показать, что вне его есть другие вычислимые функции, а следовательно, требуется новое расширение класса.

Противоречия (2) можно, однако, избежать, если определить новый оператор таким образом, чтобы он задавал частичные функции. В этом случае функция gx0(x0), а с нею и функция h(x0), не обязаны быть вычислимыми! Возможно, что вычисление их значения в x0 никогда не завершается и это значение останется неопределенным, а значит, и нет противоречия в (2).

2.3.5. Оператор (неограниченной) минимизации

Пусть f- некоторая n-местная вычислимая функция, $n \geq 1$, алгоритм вычисления f известен. Вычисление значения функции f для некоторого значения аргумента невозможно лишь тогда, когда алгоритм не может завершиться.

Зафиксируем некоторые значения первых n - 1 переменных x1, x2, ..., xn-1. Рассмотрим уравнение:

f(x1, x2,..., xn-1,y)=xn (3)

Для его решения начнем вычислять последовательно значения функции f(x1, x2,..., xn-1,y) для y=0,1,2,... . Наименьшее число k такое, что f(x1, x2,..., xn-1,k)=xn есть решение этого уравнения. Это решение обозначается μy(f(x1, x2,..., xn-1,y)=xn). Решения может и не оказаться, например, если:

- значения f(x1, x2,..., xn-1,k), k = 0, 1, 2, ... определены и отличны от xn, а значение f(x1, x2,..., xn-1,k+1) не определено,
- значения f(x1, x2,..., xn-1,k), k = 0, 1, 2, ... все определены и отличны от xn.

В таких случаях значение μy(f(x1, x2,..., xn-1,y) = xn) считается неопределенным. Величина μy(f(x1, x2,..., xn-1,y) = xn) зависит от значений переменных x1, x2,..., xn-1, xn и потому она может рассматриваться как значение функции от аргументов x1, x2,..., xn-1, xn. Эта функция обозначается Mf, M называется оператором минимизации. Если f - одноместная функция, то очевидно, что f1(x) = μy(f(y)=x).

Оператор минимизации очевидно вычислим, а значит, и функция Mf частично вычислима. Функция f называется частично рекурсивной относительно простейших функций, если она строится конечным числом применений операторов суперпозиции, примитивной рекурсии и минимизации.

Перефразируя, можно сказать, что функция называется частично рекурсивной, если она представляется конечным операторным термом. В соответствии с тезисом Черча класс частично рекурсивных функций совпадает с классом всех вычислимых функций.

Как и для оператора примитивной рекурсии, для оператора минимизации можно построить представление в виде счетного множества функциональных термов. Введем предикат P(yi):f(x1, x2,..., xn-1, yi) = xn.

Тогда оператор минимизации представляется счетным множеством функциональных термов (рис. 2.4). Предикат P(yi) выделяет функциональный терм, который вырабатывает значение функции μy(f(x1, x2,..., xn-1,y) = xn).

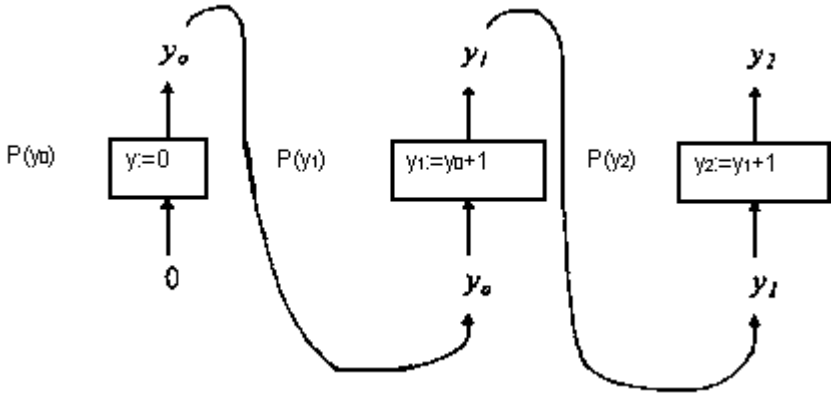


Рис. 2.4

Из всех этих термов лишь один может выработать значение функции Mf, а именно тот, который выработает значение zk = xn для минимального k. Таким образом, и здесь, как и в случае оператора примитивной рекурсии, операторный терм Mf представляется счетным множеством функциональных термов. Одно из существенных отличий состоит в том, что конечное множество функциональных термов, представляющих оператор примитивной рекурсии, фиксируется для конкретных значений до начала вычислений. В случае оператора минимизации терм, вычисляющий значение функции, определяется динамически в ходе вычисления.

Следующая программа реализует оператор минимизации:

```
P : z :=f(x1, x2,..., xn-1,0);
i :=0;
while z ≠ xn do
  {
  i := i + 1;
  z :=f(x1, x2,..., xn-1,i);
  };
Mf:=i;
```

Понятно, что эта программа не всегда останавливается и, следовательно, вычисляемая ею функция может оказаться не всюду определенной.

2.4. ДЕТЕРМИНАНТ ВЫЧИСЛЯЕМОЙ ФУНКЦИИ

Следуя [6], определим понятие вычислимой функции, которое более удобно для работы с параллельными алгоритмами и программами.

Характеристической функцией ХА множества А называется одноместная функция, равная 0 на элементах множества А и 1 за пределами А. Характеристическая функция называется частичной, если она не определена за пределами А. Множество А называется примитивно рекурсивным, если его характеристическая функция примитивно рекурсивна. Множество А называется частично рекурсивным, если его характеристическая функция частично рекурсивна.

Множество А называется рекурсивно перечислимым, если существует двухместная частично рекурсивная функция $f(a,x)$ такая, что уравнение $f(a,x) = 0$ имеет решение тогда и только тогда, когда $a \in A$.

Было показано, что каждая частично рекурсивная функция представляется конечным операторным термом. Этому представлению соответствует представление частично рекурсивной функции потенциально бесконечным множеством функциональных термов. Это множество, очевидно, частично рекурсивно перечислимое, оно устроено специальным образом (для каждого оператора свое множество функциональных термов), и существует алгоритм, перечисляющий термы множества. Для каждого операторного терма мы рассмотрели примеры программ, которые реализуют представляющее его множество функциональных термов.

И наоборот, каждой программе может быть поставлено в соответствие множество функциональных термов, определяющее тот же алгоритм, что реализован программой. Например, программа покомпонентного сложения двух векторов

```
n: = ... ;
for i = 1 to n do
z[i]: = x[i] + y[i] ;
```

может быть представлена множеством термов

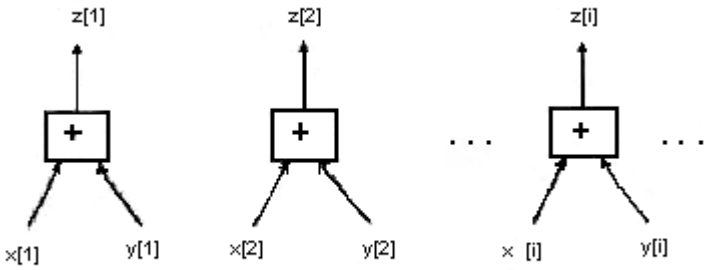


Рис 2.5

Рассмотрим другой пример:

```
n: = ... ;
for i = 1 to n do
    if P(x[i])> 0 then z[i]: = x[i] + y[i] else z[i]: = x[i] + y[i] ;
```

Этот же алгоритм может быть представлен множеством функциональных термов

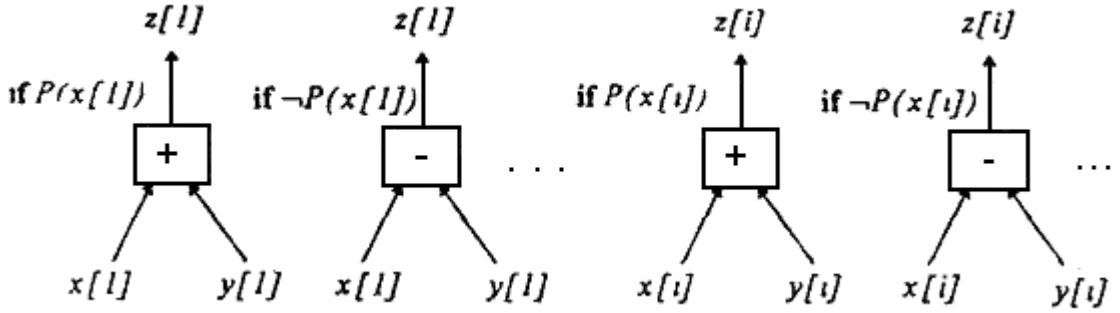


Рис 2.6

Это множество функциональных термов строится фиксацией значения предиката P. Если фиксировать P = true, то программе ставится в соответствие множество функциональных термов с использованием только операции "+". А если зафиксировать P = false, то такой программе соответствует множество функциональных термов с операцией "-" . Объединение этих множеств и даст множество, показанное на рис. 2.6.

Определение.

Функция f : Dk → D называется вычислимой функцией, если существует конечно порожденное множество T функциональных термов и выполняется:

1. ∀ (x1,x2, ... , xk, y) таких, что y = f (x1,x2, ... , xk, y), ∃ t (x1,x2, ... , xk) ∈ T такой, что dy= val(y) = val(t (x1,x2, ... , xk));
2. ∀ (x1,x2, ... , xk, y) ∀ t(x1,x2, ... , xk) ∈ T, если val(t (x1,x2, ... , xk)) = dy, то y = f (x1,x2, ... , xk, y), т.е. T не содержит "лишних" термов, не вычисляющих функцию f.

Множество термов T называется детерминантом Det$f$ функции f. Детерминант Det$f$ определяет алгоритм вычисления функции f. Если заданы два различных детерминанта Detl1f и Detl2f (существует счетное множество алгоритмов, вычисляющих f), то Detl1f ∪ Detl2f тоже является детерминантом. Ясно, что детерминант может содержать два различных терма, вычисляющих значение одной и той же переменной, и, следовательно, многозначность функции f не исключается в общем случае.

Рассмотренный ранее способ порождения множества T - представление вычислимой функции конечным операторным термом. Другой способ представления - вычислительные модели - будет рассмотрен позже. Интерес к представлению алгоритма множеством функциональных термов определяется тем, что в нем, в отличие от программы, не фиксируется способ/порядок исполнения операторов языка программирования, не задается распределение ресурсов, что оставляет много свободы для эффективной параллельной реализации алгоритма. Более детально этот вопрос будет рассмотрен в следующем разделе.

Предыдущий раздел ! | Следующий раздел !
Содержание

Содержание
Предыдущий раздел ! | Следующий раздел !

3. ЗАДАЧА КОНСТРУИРОВАНИЯ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

3.1. Представление алгоритма

Для точной формулировки задачи определим прежде всего понятие представления алгоритма A [II], вычисляющего функцию FA. Рассматриваться будут лишь такие представления алгоритма, в которых в явном виде используются преобразователи значений входных величин (переменных) в значения выходных. Такой преобразователь изображает элементарный шаг в интуитивном понятии алгоритма [9, 13].

Каждый преобразователь a (будем впредь называть его операцией) вычисляет функцию *f*a, значение которой есть результат выполнения преобразования (выполнение операции) a.

Представление алгоритма A (программа) есть набор S = (X, F, C, M), где X = {x, y, z, } - конечное множество переменных, F = {a, b, c,...} - конечное множество операций.

С каждой операцией a ∈ F связаны входной in(a) и выходной out(a) наборы переменных из X (набором называется здесь конечное линейно упорядоченное множество). Наборы in(a) и out(a) будут при необходимости рассматриваться и как множества со всеми определенными для множеств операциями.

Графическое представление операции a показано на рис 3.1. Мы будем говорить, что операция a вычисляет переменные out(a) = {y1, ..., ym} из переменных in(a) = (x1, ..., xn).

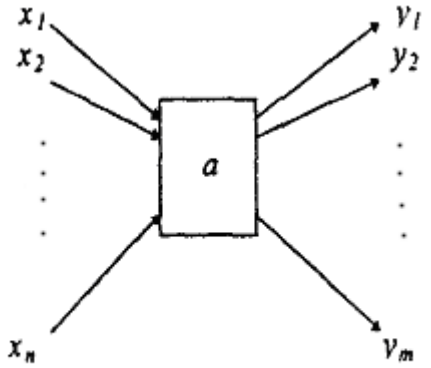


Рис 3.1

Заметим здесь, что функция *f*a, имеет n аргументов и m выходных переменных в отличие от обычно рассматриваемых в теории функций с одной выходной переменной. Такой вид операций более подходит для рассмотрения в программировании поскольку далее мы договоримся о том, что операция в программе реализуется процедурой и, следовательно, необходимо изображать преобразователи со многими выходными переменными.

C - управление, т. е. множество ограничений на порядок выполнения операций, M - функция, задающая отображение множеств переменных и операций в физические устройства многопроцессорной вычислительной системы (мультикомпьютер), т.е. M задает распределение ресурсов мультикомпьютера.

Более детальные определения C и M не приводятся, а их свойства и формы представления лишь поясняются на примерах.

Для определенности будем считать, что выполнение операции a реализуется в программе исполнением одноименной процедуры a, у которой есть входной in(a) = (x1, ..., xn) и выходной out(a) = (y1, ..., ym) наборы переменных, в программе обращение к a имеет вид a (x1, ..., xn, y1, ..., ym).

Отображение M может поставить в соответствие переменным x1 и y1 одну и ту же ячейку памяти, тогда как переменным x2, ..., xn, y2, ..., ym ставятся в соответствие

отдельные ячейки памяти каждой. Обращение в этом случае может иметь вид a (x1, ..., xn, y2, ..., ym), при этом переменная программы x1 попеременно содержит значения переменных алгоритма x1 и y1, как это обычно и делается в программах.

Одно из ограничений C может, к примеру, требовать, чтобы операция b начала выполняться непосредственно после операции a. Часть ограничений управления C, определенных информационной зависимостью между операциями (см. пример П2), называется потоковым управлением, остальные ограничения C задают прямое управление, связанное обычно с распределением ресурсов.

Отображение M назовем тождественным, если оно ставит в соответствие каждой переменной алгоритма отдельную ячейку памяти, а каждой операции - собственный, ни с кем не разделяемый процессор.

Реализацией алгоритма A, представленного в форме S, называется выполнение операций в некотором произвольном допустимом порядке, который не противоречит управлению C. Предполагается, что при любой реализации вычисляется функция FA. Множество всех реализации обозначим P(A, S).

Если P(A, S) есть одноэлементное множество, то S - последовательное представление. Будем говорить, что S - параллельное представление алгоритма A, если в множестве P(A, S) существует более одной реализации.

Пример 1

Алгоритм задан на языке C, каждый вычисляющий (не управляющий) оператор программы есть операция, выполнение операции есть выполнение оператора языка, порядок выполнения операций полностью фиксирован и, следовательно, представление последовательное.

Пример 2

Пусть F = f(x1, x2), x1 = g(y1, ..., yk), x2 = h(z1, ..., zm) - операции, out(h)={x2}, out(g) = x1, out(f) = {F}. Тогда алгоритм вычисления функции F = S(f, g, h) = f(g1(y1, ..., yk), h = (z1, ..., zm)) представлен в параллельной форме, f, g и h -операции. Допустим всякий порядок выполнения операций, при котором вычисление аргументов операции предшествует выполнению операции, управление только потоковое, сохраняющее информационную зависимость между операциями.

Уменьшить потоковое управление (например, разрешить операции f выполняться раньше операции g) нельзя, так как при этом будет вычисляться, вообще говоря, не функция F, а какая-то другая. В этом смысле потоковое управление есть минимальное управление, гарантирующее вычисление функции F. Множество реализаций состоит из элементов, показанных на рис. 3.2

Пример 3

Алгоритм представлен в виде асинхронной программы (см. подраздел 4.1), A-блок здесь операция, условие выполнения операции - истинность спусковой функции, чем и определяется множество реализаций.

Будем говорить, что представление S1 более непроцедурно, чем S2, если P(A,S1) $\supset$ P(A,S2). Например, пусть для хранения значений переменных используются ячейки памяти и заданы представления S1 = (X, F, C1, R1) и S2 = (X, F, C2, R2) алгоритма вычисления функции f = f(g (y1, ..., yk), h(z1, ..., zm)) (рис 3.3). В S2 потребуем, чтобы значения переменных алгоритма x1 и z1 хранились в одной и той же ячейке памяти.

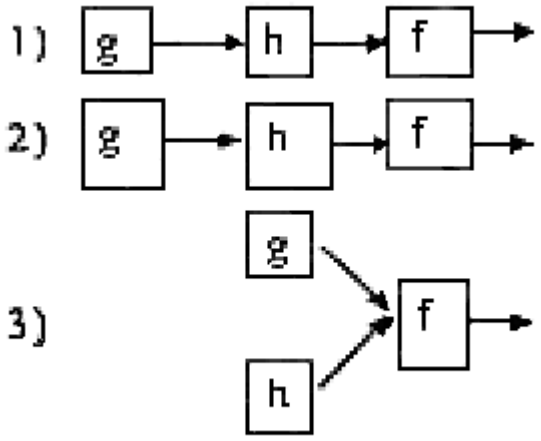


Рис 3.2

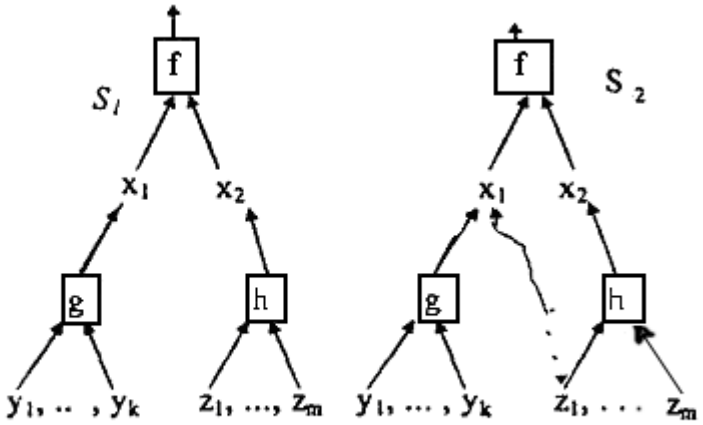


Рис 3.3

Тогда P(A,S1) ⊃ P(A,S2), так как для вычисления *f* прямое управление в C2 должно содержать требование, чтобы операция h выполнялась раньше g. При этом операция h, начав исполняться, использует значение переменной z1 до того, как в ту же ячейку памяти будет записано значение переменной x1, вычисленное операцией g.

Очевидно, что представление S алгоритма A с потоковым управлением C и тождественным распределением ресурсов R является максимально непроцедурным представлением A. Такое представление будем называть просто непроцедурным.

Рассмотрим еще несколько примеров программ, иллюстрирующих введенные определения.

Пример 4

Алгоритм A задан в форме программы P, записанной на некотором последовательном языке программирования. Операторы языка, вычисляющие значения переменных, есть операции. Прямое управление задается последовательностью операторов в программе либо специальными операторами управления типа go to.

Порядок выполнения операций полностью фиксирован. Распределение памяти задается идентификаторами переменных программы, которые фактически есть имена ячеек

памяти, имена переменных алгоритма в программе не сохраняются. Предполагается, что для выполнения алгоритма может быть использован только один процессор.

На рис. 3.4 приведен пример представления S = (X, F, C, M) алгоритма A и одного из его возможных представлений в форме программы P. Здесь X = {x, y, z, x1, z1}, F = {a, b, c}. C - потоковое управление (показано стрелками), M - тождественное распределение ресурсов.

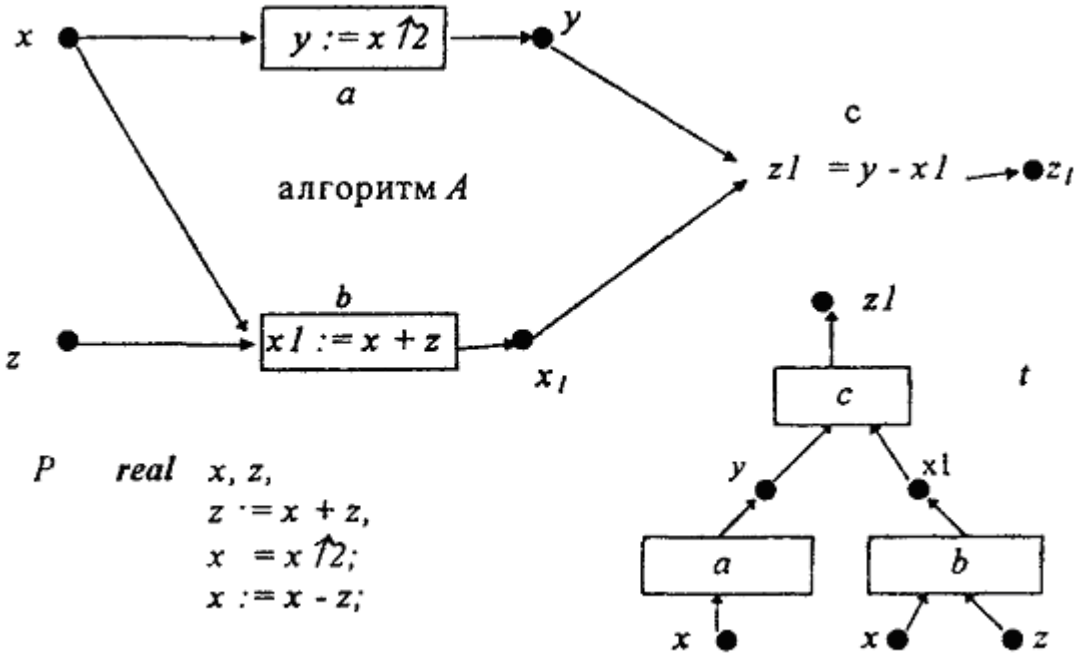


Рис 3.4

Программа P может быть построена следующим регулярным способом. Вначале строится функциональный терм t (рис. 3.4), определяющий алгоритм решения задачи. Устройство терма показывает, что после выполнения операции b переменная z более не используется, а переменная x еще будет нужна для корректного исполнения операции a. Поэтому переменной x1 может быть назначена та же ячейка памяти, что и для переменной z (что и сделано в программе P).

После исполнения операции a и переменная x тоже становится ненужной и потому для хранения значения переменной y может быть назначена та же самая ячейка, что и для переменной x.

После исполнения операции c сразу две переменные - y и xl - становятся ненужными (их значения полностью использованы) и потому значение переменной z1 может быть положено как в ячейку, где хранится значение y, так и в ячейку, где хранится значение переменной xl (в программе P значение переменной zl положено в ту же ячейку, где хранилось значение переменной y, а еще ранее хранилось значение переменной x.

Таким образом, в программе P переменная программы, помеченная идентификатором x (имя ячейки памяти), последовательно хранит значения переменных алгоритма x, y, z1 (т.е. задано отображение M(x)= M(y) = M(z1) = x, последний x есть имя ячейки), порядок выполнения операторов программы не может быть изменен в силу выбранного распределения памяти.

При необходимости реализовать алгоритм A на мультикомпьютере программа P распараллеливается, т.е. преобразуется в функционально эквивалентную программу P1,

вычисляющую ту же функцию FA, но допускающую параллельное выполнение некоторых операторов. Таким образом, задача распараллеливания заключается в нахождении параллельного представления A (конструировании новых C и M) на основе анализа P.

Пример 5

Алгоритм A задан в форме программы P, записанной на некотором параллельном языке, использующем конструкции типа fork и join для задания параллельных ветвей.

Например, fork V1, V2, V3 join, где V1, V2, V3 - последовательные участки программы, которые могут выполняться независимо. Участки V1, V2, V3 формируются таким образом, чтобы время их выполнения было одинаковым. Схематично исполнение программы показано на рис. 3.5, стрелки показывают поток управления. После оператора fork ветви V1, V2, и V3 могут исполняться в произвольном порядке, независимо друг от друга. Оператор join ожидает завершения всех ветвей.

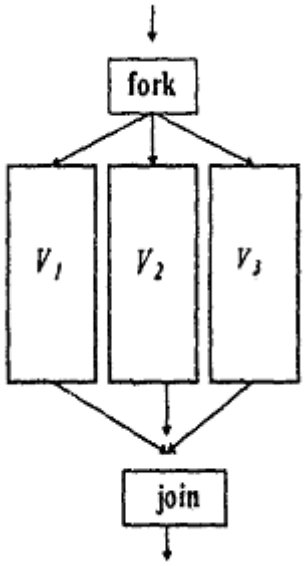


Рис 3.5

Программа хорошо выполняется на трех одинаковых процессорах и вдвое хуже на двух в силу излишне жестко заданного прямого управления. Во втором случае вначале будут выполнены две (доступны два процессора) любые ветви, например V1 и V3, затем ветвь V2 (один процессор простаивает) и лишь после этого оператор join разрешит продолжить исполнение P далее. Программа P должна преобразовываться при изменениях в структуре мультикомпьютера.

Пример 6

Алгоритм A задан асинхронной программой P. Управление C задается спусковыми функциями и сетями Петри (см. подраздел 4.2). Распределение ресурсов находит отражение в устройстве C, поэтому, хотя P и в меньшей степени зависит от изменений в структуре мультикомпьютера в силу недетерминированности управления, ее также часто необходимо преобразовывать (конструировать новые C и M) при изменении структуры мультикомпьютера для получения хороших рабочих характеристик реализации A.

Обращаем внимание читателя на диаграмму (рис. 3.6). Она показывает, что для каждой вычислимой функция FA существует счетное множество различных алгоритмов A1,..., An, ..., вычисляющих FA, и для каждого алгоритма An существует счетное множество различных программ P1, ... , Pm, ..., реализующих An.

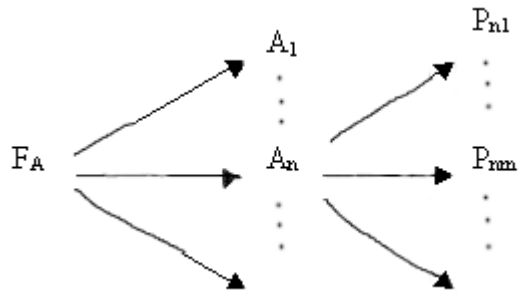


Рис 3.6

В приведенном определении представления алгоритма использовано лишь конечное число операций, при этом в реализующей программе будут отсутствовать циклы. Использование массовых операций [2] легко исправляет этот недостаток.

Не вдаваясь в детали определения массовых операций, рассмотрим алгоритм Afile, ввода записей файла *f* в память (в массив y), затем обработку всех его компонентов операцией a с посылкой результатов в компоненты массива z. С учетом того что в максимально непроцедурном представлении должен быть указан каждый вычислительный акт, каждое возможное выполнение операции, алгоритм будет иметь вид, показанный на рис. 3.7.

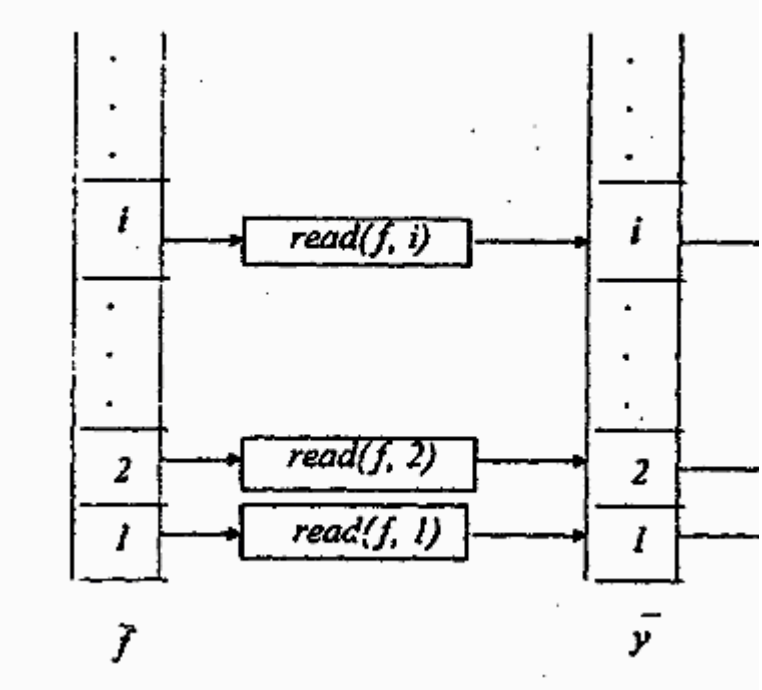


Рис 3.7

Операция read(*f*, i) считывает i-ю запись файла в компонент массива y[i], операция ai обрабатывает i - запись, а результат записывает в i-ю запись файла z. Если отображение M ставит в соответствие массиву y участок памяти, в котором может быть размещена лишь одна запись, тогда в прямом управлении C должен быть задан порядок выполнения операций read (*f*, 1), a1, read(*f*, 2), a2, ... (обычно задается циклом).

При таком порядке исполнения считанная запись файла будет потреблена операцией a, память освобождена и следующая запись может быть считана в тот же участок памяти, что и предшествующая запись. Алгоритм Afile будет правильно реализован, т.е. будет вычислена функция FAfile. Если две записи разрешается разместить в памяти, тогда может

быть использован режим ввода с двумя буферами и две операции - ai и ai+1 - смогут выполняться одновременно, когда есть доступные процессоры, и т.д.

3.2. ТРЕБОВАНИЯ К ПРЕДСТАВЛЕНИЮ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА

Примеры показывают, что к представлению S алгоритма A для последующей реализации предъявляются противоречивые требования.

С одной стороны, необходима непроцедурность S, так как она уменьшает зависимость S от конкретного мультикомпьютера, позволяет качественно реализовать алгоритм на различных мультикомпьютерах, а значит, есть возможность накапливать программный фонд для них.

С другой стороны, есть потребность в таком уменьшении непроцедурности S, при котором в множестве P(A, S) остаются лишь оптимальные относительно некоторого критерия реализации, потому что уменьшение непроцедурности S будет способствовать улучшению рабочих характеристик программы.

Противоречие может быть разрешено, если конструирование качественной параллельной программы решения задачи проводить в два этапа:

1. вывод алгоритма решения задачи и его непроцедурное представление;
2. конструирование по непроцедурному представлению алгоритма и описанию мультикомпьютера прямого управления и распределения ресурсов такой степени непроцедурности, которая позволит эффективно реализовать алгоритм на заданном мультикомпьютере.

При этом в программном фонде хранятся алгоритмы в непроцедурной форме, которая перед выполнением алгоритма на конкретном мультикомпьютере дополняется соответствующим прямым управлением и не тождественным распределением ресурсов для получения оптимальных относительно заданного критерия рабочих программ.

В такой общей постановке задача конструирования эффективной параллельной программы (конструирования C и M) в целом не имеет приемлемого решения, хотя к настоящему времени разработано много алгоритмов конструирования M (M - алгоритмов). Слишком велико оказалось разнообразие различных типов и структур вычислительных алгоритмов и мультикомпьютеров, и, к примеру, M-алгоритмы, используемые для реализации алгоритмов в MIMD мультикомпьютере, имеют мало общего с M-алгоритмами для реализации A в конвейерных мультикомпьютерах.

Какую систему параллельного программирования (СПП) хотелось бы иметь для крупноблочных мультикомпьютеров? Прежде всего СПП должна быть очень проста так, чтобы любой прикладной программист, прикладной математик смог сам без чрезмерных усилий создавать эффективные параллельные программы. Для этого СПП должна быть основана на некотором широко распространенном процедурном языке программирования, например Фортране или C, в который необходимо добавить непроцедурные средства представления A таким образом, чтобы усилия пользователя и СПП взаимно дополняли бы друг друга в попытках организовать эффективные параллельные вычисления.

Термин "эффективный" используется, конечно, неформально. Он может пониматься, например, так, что выгоды от использования мультикомпьютера компенсируют прикладному пользователю затраты на разработку параллельной программы, а потери производительности мультикомпьютера оказываются приемлемыми за счет повышения производительности труда программистов.

Обычно задача отображения A на ресурсы мультикомпьютера решается статически, до начала вычислений, и все принятые решения по распределению вычислительных ресурсов, в первую очередь процессоров, фиксируются в тексте программы.

Такой подход, однако, часто неприемлем, так как мультикомпьютер может иметь много ресурсов и далеко не каждая программа все их использует. Следовательно, вычислительные ресурсы мультикомпьютора надо разделять динамически, т.е. реализовать мультипрограммирование на вычислительных ресурсах мультикомпьютера.

Таким образом, M - алгоритм должен состоять их двух частей: статической, которая реализуется в трансляторе и производит распределение той части ресурсов мультикомпьютера, которая может быть отведена программе до начала вычислений (память, регистры), и динамической, которая реализуется в программе или в операционной системе и производит захват и перераспределение ресурсов мультикомпьютера в ходе исполнения прикладной программы (процессоры, каналы, память).

3.3. СРАВНИТЕЛЬНАЯ НЕПРОЦЕДУРНОСТЬ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Рассмотрим понятие непроцедурности языков программирования, чтобы иметь возможность сравнивать их по этому свойству.

Прежде всего отметим, что одной и той же конструкции языка, определенного некоторой грамматикой, можно ставить в соответствие разные правила реализации, т е. задавать для нее разные операционные семантики.

Наоборот, для разных конструкций языка могут быть определены правила исполнения, порождающие одно и то же множество вычислительных процессов (одно и то же множество реализаций).

И наконец, при сравнении языков с разными правилами порождения языковых конструкций и с разными правилами их реализации следует исходить из некоторого соответствия между конструкциями языка. Уточним теперь эти три концепции.

Пусть L - язык программирования, для которого заданы две операционные семантики S1 и S2, определенные как множества реализаций (другой возможный здесь термин - множество операционных историй) Si(k), i = 1, 2, где $k \in L$ - конструкция языка L. Операционная семантика определяет, собственно, возможные алгоритмы реализации конструкций языка. Именно операционная семантика выражает процедурность языка L, в отличие от семантики математической, связанной с непроцедурностью. Будем считать, что семантика S1 более непроцедурна, чем S2, если $\forall\ k \in L(S1(k) \supset S2(k))$, что соответствует нашему определению понятия непроцедурности представления алгоритма.

Если L - язык программирования, k1 и k2 - его конструкции, S - операционная семантика, то скажем, что k1 более непроцедурна, чем k2 в смысле семантики S, если $S(k1) \supset S(k2)$.

Пример 1

Пусть L - язык программирования, например Фортран или C, S1 - его семантика. Определим теперь его семантику S2, совпадающую с S1 всюду, кроме операторов цикла, для которых допустимо в S2 параллельное выполнение независимых итераций (интерацией цикла называется исполнение тела цикла с некоторым конкретным значением параметра цикла). Тогда семантика S2 более непроцедурна, чем S1 .

В частности, цикл

x[1]: = c1; x[2]: = c2;
for i = 1 to 100 do
(x[i+2]: = a(x[i]); x[i+3]: = b(x[i+l];)

определяет выполнение операций a и b в следующем порядке: a(x[1]), b(x[2], a(x[3]), b(x[4], ... Так как в семантике S2 допускается параллельное выполнение независимых итераций цикла, то допустим следующий (наряду со счетным множеством других, каких?) порядок выполнения итераций цикла:

1. : a(x[1]), b(x[2])
2. : a(x[3]), b(x[4])
3. : . . .

Пусть язык C дополнен циклом вида $\forall$ x $\in$ X | P(x) : : A, где X - массив, A - блок. Операционная семантика этого оператора не фиксирует порядок выполнения итераций цикла и допускает выполнение итерации, если вычислены все ее аргументы. Тогда мы скажем, что введенный оператор цикла более непроцедурен, чем цикл типа for.

Пусть теперь L1 и L2 - языки программирования с семантиками S1 и S2 соответственно, f - отношение функциональной эквивалентности, заданное на K (L1)× (L2), где K(L) - множество конструкций (операторов) языка программирования L. Дадим определения сравнительной непроцедурности языков L1 и L2.

Определение 1. Язык L1 более непроцедурен, чем L2 , если

f

$\forall$ k2 $\in$ L2 $\exists$ k1 $\in$ L1((k1~k2)& (S1(k1) $\supset$ S2(k2))).

В соответствии с определением 1 язык L1 считается более непроцедурным, чем L2, если любая программа на языке L2 может быть записана на языке L2 в более непроцедурной форме.

Определение 2. Язык L1 более непроцедурен, чем L2, если

f

$\forall$ k2 $\in$ L2 $\forall$ k1 $\in$ L1((k1~k2) $\rightarrow$(S1(k1) $\supset$ S2(k2))).

Согласно определению 2 язык L1 более непроцедурен, чем L2, в случае, если любая программа на L2 неизбежно записывается на L1 (алгоритм представляется в L1) в более непроцедурной форме.

Определение 3. Пусть задано некоторое отображение конструкций h:K(L1)$\rightarrow$K(L2). Если

f

$\forall$ k2 $\in$ L2 $\exists$ k1 = h-1(k2) ((k1~k2)& (S1(k1) $\supset$ S2(k2))).

то язык L1 более непроцедурен, чем L2, в смысле h.

Каждое отображение h задает класс конструкций языков L1 и L2, которые только и необходимо сравнивать. Следовательно, можно сравнивать языки только частично, на части конструкций.

Заметим в заключение, что использование аппаратов макрогенерации и процедур не увеличивает непроцедурности языка программирования, поскольку в итоге порождается то же множество реализаций.

4. ПАРАДИГМЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

4.1. Асинхронное программирование

В асинхронной модели вычислений предполагается, что программа выполняется на мультипроцессоре(несколько процессорных узлов над общей разделяемой памятью) или на

мультикомпьютере (несколько процессорных узлов, каждый из которых состоит из процессора, собственной оперативной памяти, устройства управления и т п. Таким образом, каждый процессорный узел является полным компьютером). Асинхронная программа представляет систему независимо исполняющихся взаимодействующих процессов. Существует много различных воплощений этой модели вычислений в различных языках программирования. Мы рассмотрим эту модель в весьма общем виде.

4.1.1 Понятие асинхронной программы

Асинхронная программа (А-программа) - это конечное множество А-блоков {Ak| k ∈ {1, 2, ..., m}}, определенных над информационной IM и управляющей CM памятями.

Каждый А-блок Ak = <tr(ak),ak, c(ak)> состоит из спусковой функции tr(ak) (trigger function), управляющего оператора c(ak) и операции ak, ak ∈ F.

Памятью называется конечное множество всех ячеек, способных хранить значения переменных.

Спусковая функция tr(ak) - это предикат, в программе обычно задается условным выражением либо логической функцией. Как обычно, операция ak реализуется в программе одноименной процедурой, вычисляющей функцию *f*a.

Управляющий оператор c(ak) - оператор присваивания или процедура, меняющая значения ячеек управляющей памяти CM (например, чтобы разрешить или запретить исполнение какого-нибудь А-блока).

Каждой переменной из in(ak) ∪ out(ak) в памяти IM соответствует ячейка, в которой хранится ее значение.

Выполнение А-программы состоит:

- в вычислении значения спусковой функции tr(ak) для всех или части А-блоков;
- выполнении одного, части или всех А-блоков Ak так что (tr(ak) = true при текущем состоянии памяти IM ∪ CM.

Исполняющая система, таким образом, имеет право инициировать любое подмножество А-блоков, готовых к исполнению (с истинной tr(ak)), в зависимости от имеющихся в наличии свободных на этот момент ресурсов. Понятно, что при разных исполнениях асинхронной программы на каждом этапе исполнения будут инициированы, вообще говоря, разные подмножества А-блоков. Это обстоятельство (недетерминизм) сильно усложняет отладку асинхронной программы.

А-программа считается завершенной, когда ни один блок не выполняется и ни один А-блок не может быть инициирован, так как для всех к = 1, ..., m значение tr(ak) = false.

Выполнение А-блока Ak состоит в исполнении:

- процедуры ak с теми значениями, которые имеют переменные из in(ak) в текущий момент, при этом вычисляются значения переменных из out(at);
- управляющего оператора c(ak)

Таким образом, спусковые функции и управляющие операторы - это те средства, с помощью которых задается управление в асинхронных программах.

Пример 1

```
integer x, y, z;
input (x,y,z)
asynchronous_do
% tr(ak) ak %
x < yVx < z → x: = x + 1;
y < zVy < x → y: = y + 1;
```

z < xVz < y → z: = z + 1;
end_do

Программа "примера 1" позволяет в некотором порядке уравнять значения переменных x, y, z, доведя их с помощью прибавления единицы до значения наибольшей их них (рис. 4.1). Это программа максимальной непроцедурности, в ней нет ограничений, которые бы не были обусловлены наличием информационной зависимости между операторами.

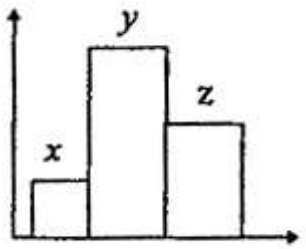


Рис 4.1

Пример 2

Большая непроцедурность часто мешает эффективно выполнить программу и в таких случаях требуется уменьшать непроцедурность программы (представления алгоритма). Для программы "примера 1" это можно сделать, убрав дизъюнктивные члены из спусковых функций, что усилит управление (увеличит множество ограничений в нем).

integer x, y, z;
input (x,y,z)
asynchronous_do
% tr(ak) ak %
x < y → x: = x + 1;
y < z → y: = y + 1;
z < x → z: = z + 1;
end_do

Пример 3

Программа данного примера отличается от программы "примера 2" тем, что в спусковые функции добавлены новые конъюнктивные и дизъюнктивные члены. Они добавляют новые ограничения в управление, и в результате получается почти последовательная программа. Параллельное выполнение двух А-блоков возможно лишь при условии равенства значений двух переменных (при равенстве значений всех трех переменных программа завершается). Попробуйте превратить эту программу в последовательную.

input (x,y,z)
asynchronous_do
% _____tr(ak)_____ _ak _ %
x < z & x < y V(x:=z&x<y)V(x=y&x<z) → x:=x + 1;
y < x & y < z V(y:=x&y<z)V(y=z&y<x) → y:=y + 1;
z < y & z < x V(z:=y&z<x)V(z=x&z<y) → z:=z + 1
end_do

Пример 4

В примерах 1...3 А-блоки не имели управляющего оператора (более точно, был пустой управляющий оператор). Рассмотрим теперь организацию конвейерного исполнения А-блоков. Схема программы показана на рис. 4.2.

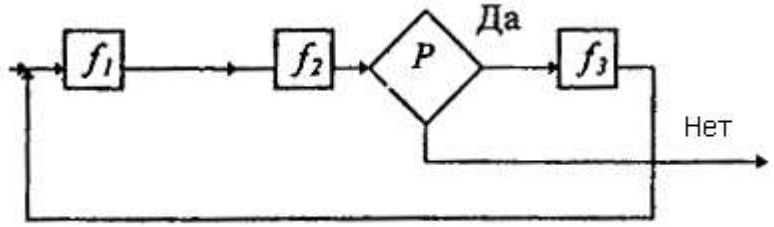


Рис 4.2

Асинхронная программа, реализующая этот конвейер (в программе не показаны переменные, обрабатываемые этой программой, программа демонстрирует только конструирование управления), может иметь вид

integer x,y, z;
asynchronous_do
x: = 1; y: = z: = 0;
% tr(ak) ak c(ak) %
x = 1 $\rightarrow f1$ [y: = 1; x: = 0];
y = 1 $\rightarrow f2$ [z: = 1; y: = 0];
P&z = 1 $\rightarrow f3$ [x: = 1; z: = 0];
end_do

В этой программе операторы x:=1; y:=z:=0 присваивают значения управляющим переменным. Они разрешают инициировать операцию $f1$ и запрещают инициирование операций $f2$ и $f3$. Таким образом, в начальный момент времени сможет быть инициирован только А-блок $f1$. После выполнения операции $f1$ управляющий оператор c1 (здесь это оператор [y:=1; x:=0]) запретит исполнение А-блока $f1$ и разрешит исполнение А-блока $f2$ и т.д. Цикл конвейера завершится, когда предикат P станет ложным и не позволит инициировать А-блок $f3$.

Широко известным примером асинхронного программирования является message passing interface. Этот метод программирования определяет программу как систему взаимодействующих процессов и стандартизует обмен данными. Обмен всегда происходит через каналы, связывающие два процесса. Такой канал реализуется очередью сообщений, каждый процесс сам определяет свою готовность начать исполнение или завершить его.

Асинхронное программирование оказалось весьма трудоемким делом, особенно отладка программы, в силу обилия возможностей совершить ошибки в синхронизации процессов. Рассмотрим некоторые проблемы асинхронного программирования.

4.1.2. НЕКОРРЕКТНОЕ ВЫЧИСЛЕНИЕ ДАННЫХ

Возникновение некорректных данных иллюстрирует следующий простой пример. Пусть необходимо начислить зарплату и вычислить сумму денег, подлежащих выдаче на руки. Оставляя в стороне излишние здесь детали, будем предполагать, что зарплату

составляют некоторая базисная зарплата N0 плюс надбавки N1, N2, ..., Nn (выражаются в процентах к базисной зарплате) минус налог Nn-1 (выражаются в процентах к начисленной сумме).

Пусть процессы P0, P1, P2, ... , Pn, Pn+1 соответственно выполняют эти операции. Понятно, что для вычисления корректного результата процесс Pn-1, вычисляющий сумму налога, должен выполняться последним, процесс P0 -первым, а процессы P1, P2, ..., Pn могут исполняться в любом порядке, в том числе и параллельно, хотя само суммирование должно, конечно, выполняться последовательно. Все другие варианты выполнения процессов приведут к некорректным результатам.

Такого сорта ошибки асинхронных программ крайне неприятны, трудно локализуемы и неповторимы. Если позволить процессам P1, P2,..., Pn, Pn-1 выполняться асинхронно, то при разных выполнениях асинхронной программы могут получаться разные результаты и повторить предшествующее тестирование удастся только случайным образом. Понятно, что (P0 + P1 + Pn-1 + P2 + ... + Pn) ≠ (P0 + Pn-1 + P1 + P2 + ... + Pn), здесь имеется в виду, что процессы выполняются в написанном порядке.

4.1.3. Некорректное считывание данных

Следующий пример иллюстрирует этот тип ошибки. Пусть в банке A есть счет acc1, на котором находится 500 тыс. руб., а в банке B - счет acc2, на котором находится 300 тыс. руб, и необходимо переслать 100 тыс. руб. со счета acc1 на счет acc2. Сумма денег на обоих счетах неизменна до и после выполнения пересылки и равна 800 тыс руб. Пусть процесс P1 посылает деньги из банка A в банк B, а процесс P2 принимает посланные деньги в банке B. Процессы схематически могут быть описаны так:

Первоначально

A.acc1 = 500 тыс. руб. B.acc2 = 300 тыс. руб.

Процесс P1	Процесс P2
A.acc1: = A.acc1 - 100;	receive (x,A,y);
x: = 100	
send (x, B, y);	B.acc2: = B.acc2 + y;

Результат

A.acc1 = 400 тыс. руб. B.acc2 = 400 тыс. руб.

В процессе P1 счет A.acc1 вначале уменьшается на 100 тыс. руб., а затем 100 тыс.руб. посылаются в банк B. В процессе P2 сначала принимаются 100 тыс. руб. из банка A, а затем увеличивается сумма на счету B.acc2.

Однако процесс P, контролирующий перевод денег и проверяющий сохранение значения суммы A.acc1 + B.acc2, может считывать ошибочные данные. Понятно, что если P будет считывать данные строго до или после выполнения операции перевода денег, то результат суммирования будет одинаков. Однако в ходе выполнения процессов P1 и P2 при разных вариантах считывания значений A.acc1 и B.acc2 сумма A.acc1 + B.acc2 может равняться 700 тыс.руб. (если значение A.acc1 было считано после его изменения, а значение B.acc2 - до изменения) , 800 тыс. руб. и 900 тыс.руб. при других возможнык вариантах считывания данных.

4.2. СЕТИ ПЕТРИ

Сети Петри - это математическая модель, которая имеет широкое применение для описания поведения параллельных устройств и процессов. В настоящее время определены

и изучены разнообразные классы сетей Петри. Мы рассмотрим самые общие понятия и возможности использования сетей Петри для задания прямого управления в параллельных программах.

4.2.1. Определение сети Петри

Определение. Сеть Петри есть двудольный ориентированный граф. Напомним, что двудольный граф - это такой граф, множество вершин которого разбивается на два подмножества и не существует дуги, соединяющей две вершины из одного подмножества. Итак, сеть Петри - это набор

$N = (T,P,A), \quad T \cap P = \varnothing,$

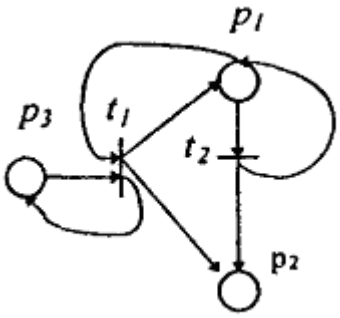
где $T = \{t_1, t_2, ..., t_n\}$ - подмножество вершин, называющихся переходами;
$P = \{p_1, p_2, ..., p_m\}$ - подмножество вершин, называющихся местами;
$A \subseteq (T \times P) \cap (P \times T)$ - множество ориентированных дуг.

По определению, дуга соединяет либо место с переходом, либо переход с местом.

Пример 1

На рис. 4.3,а приведен пример сети Петри в графическом представлении. Переходы обозначены черточками, а места - окружностями. Каждый переход *t* имеет набор входных *in*{t} и набор выходных *out*{t} дуг. Сети Петри могут представляться также в форме продукционных правил (рис. 4.3,б).

Наиболее интересны сети Петри тем, что они позволяют представлять и изучать в динамике поведение системы параллельных процессов в программе или в любом другом дискретном устройстве.



$t_1 : \{p_3, p_1\} \rightarrow \{p_1, p_2, p_3\}$

$t_2 : \{p_1\} \rightarrow \{p_1, p_2\}$

а б

Рис 4.3

4.2.2. Разметка сети

Сеть Петри можно понимать (интерпретировать) по-разному. Можно представить себе, что места представляют условия (буфер пуст, файл закрыт и т.п.), а переходы - события (посылка или получение сообщения в буфер, запись в файл).

Состояние сети Петри в каждый текущий момент определяется системой условий. Для того чтобы стало возможным и удобным задавать условие типа "в буфере находится 3 записи", в модель сети Петри добавляются *фишки*. Фишки изображаются точками внутри места. В применении к программированию можно представлять себе переходы как процедуры, а места - как переменные или буфер.

Фишка свидетельствует о том, что переменная/буфер имеет значение, а если место имеет, к примеру, 3 фишки, то это может интерпретироваться как наличие трех разных значений в буфере. Если место содержит фишку, то место маркировано и сеть называется маркированной. Начальное распределение фишек задает начальную маркировку M_0 сети. Маркировка сети определяет ее текущее состояние.

Сеть на рис. 4.4 в начальном состоянии содержит одну фишку в месте p_3. Маркировка формально задается функцией M: P → I, I = {0,1,2,..}, а функция M представляется вектором, в котором i-й компонент задает маркировку места p_i. Например, начальная маркировка сети на рис. 4.4 представляется вектором M_0 = {1,0,0}.

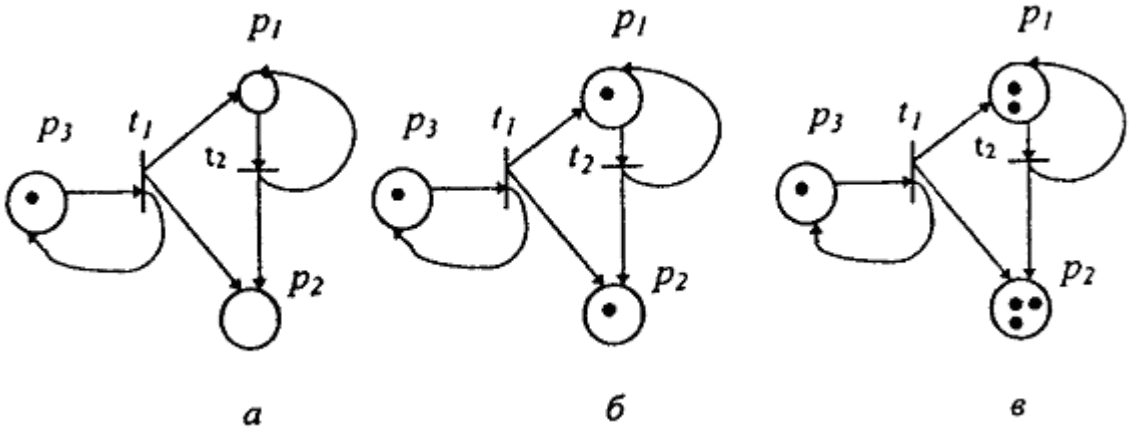


Рис 4.4

На рис. 4.4 показана последовательность состояний сети Петри в ходе срабатывания переходов. Начальная разметка M_0 = (1,0,0) показана на рис. 4.4,а. В этом состоянии может сработать только переход t_1. Разметка сети M_1 = (1,1,1) после срабатывания t_1 показана на рис. 4.4,б. Последняя позволяет одновременно сработать переходам t_1 и t_2 , разметка M_2 = (1,2,3) после их срабатывания показана на рис. 4.4,в.

Сеть переходит из одного состояния в другое (от одной маркировки к другой), когда происходит событие - срабатывание перехода. Переход может сработать, если есть хотя бы одна фишка во всех его входных местах (рис. 4.5)

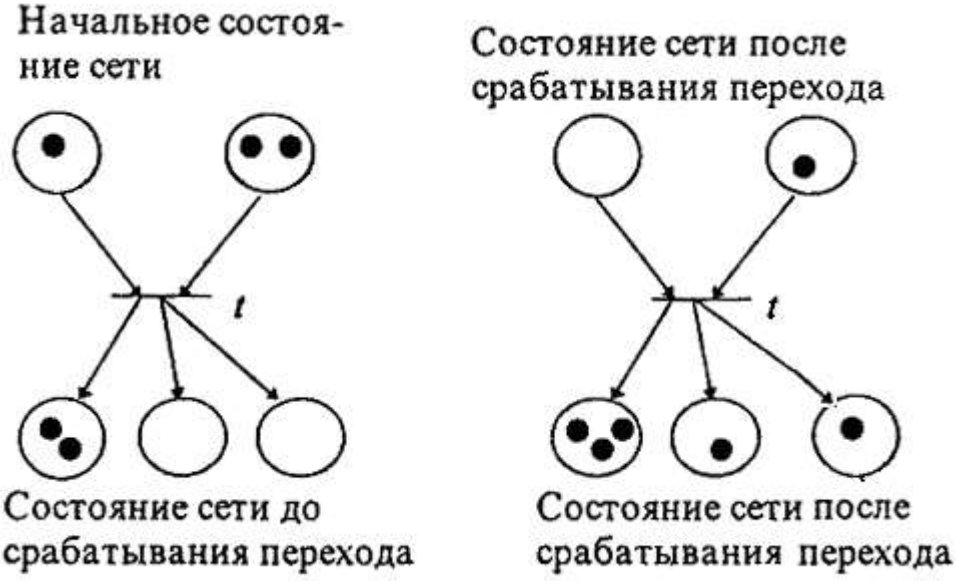


Рис 4.5

Срабатывание перехода состоит из того, что из всех входных мест забирается по одной фишке и во все выходные места добавляется по одной фишке. Если представить себе переход как процедуру, то она корректно выполняется при наличии значений всех своих аргументов и вырабатывает значения всех выходных переменных.

В другой интерпретации переход может представлять некоторое устройство. Устройство может (но не должно) сработать, если выполнились все входные условия. Если несколько переходов готовы сработать, то срабатывает один из них (любой), или некоторые из них, или все (рис. 4.6).

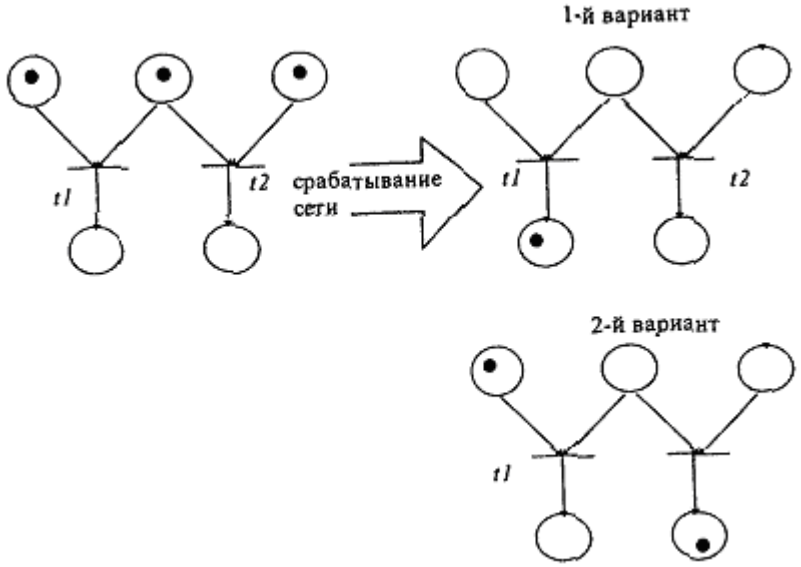


Рис 4.6

Пример 2

Рассмотрим пример конвейера. Пусть есть три обрабатывающих устройства t_0, t_1, t_2 организованные в виде конвейера. Это могут быть, например, станки на заводе или функциональные устройства конвейерного процессора и вообще любой конвейер, в котором каждое обрабатывающее устройство выполняет лишь часть общей работы, а результат будет выработан лишь последним из них.

Особенностью нашего конвейера является ограниченность емкости мест p_1 и p_2; место p_1 может вместить лишь два результата (место p_1 сети является *2-ограниченым*) предшествующего этапа работы конвейера (вырабатывается переходом t_0), а место p_2 - 3-ограниченным. Символ *n* в месте p_0 означает наличие *n* фишек в нем, *n* - целое положительной число.

Сеть Петри, обеспечивающая необходимое прямое управление, приведена на рис. 4.7. Понятно, что в месте p_1 не может накопиться более 2 фишек при любых порядках срабатывания переходов сети. Места p_1 и p_2 часто еще называют асинхронными каналами, с их помощью реализуется программирование средствами асинхронного *message passing interface* (см. подраздел 4.3).

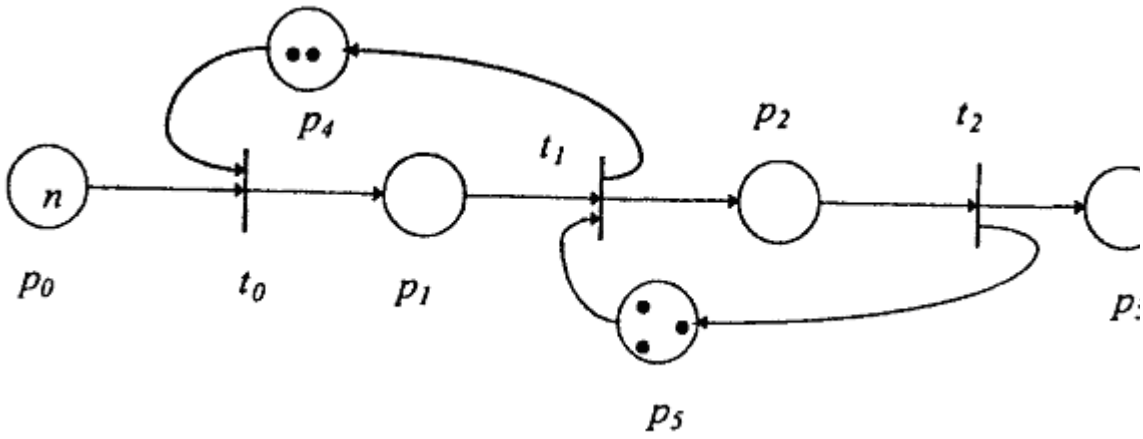


Рис 4.7

Сеть Петри, в которой все места 1-ограничены, называется *безопасной*. Такой сетью можно задавать прямое управления в программах. Безопасная сеть никогда, не допустит, чтобы в переменную было положено новое значение, если старое еще не было использовано по назначению. Нарушения этого правила часто являются причиной ошибок в параллельных программах.

При использовании сетей Петри в языках программирования стандартные вычисления (например, сеть управления конвейерным исполнением процедур) может быть описана как управляющая процедура, например:

control procedure *pipiline* (t_0, p_4, t_1, p_5, t_2);
< описание сети примера 1>;

Теперь, при необходимости задать в программе конвейерное вычисление некоторых процедур, их имена подставляются в обращение к управляющей процедуре *pipiline* вместо формальных параметров t_0, t_1, t_2.

call pipiline (t_0 = procl, p_4 = 2, t_1 = proc2, p_5 = 3, t_2 = proc3.)

Наличие библиотеки стандартных управляющих процедур способно облегчить отладку параллельной программы.

Сеть *примера 2* может быть использована также для управления асинхронным каналом при описания и реализации *message passing interface* в языках с асинхронными взаимодействиями.

4.2.3. ГРАФ ДОСТИЖИМОСТИ

Для использования сетей Петри необходимо знать их свойства, такие, как безопасность, а для этого следует изучить множество всех разметок сети с заданной начальной разметкой.

Разметка М называется достижимой, если при некоторой последовательности срабатываний сети, начиная с начальной разметки M_0, она переходит к разметке М.

Множество разметок, достижимых в порядке срабатывания сети, представляется *разверткой сети*.

Множество достижимых разметок удобно представлять *графом достижимости*, который показывает все достижимые разметки и последовательности срабатываний переходов, приводящих к ним.

Пример 3

Следующая сеть (рис. 4.8,а.) определяет управление двумя параллельно протекающими процессами с синхронизацией. Граф достижимости показан на рис. 4.8,б. Граф развертки сети бесконечен (рис 4.8,б), однако после состояния M_3 в каждой ветви повторяется один и тот же фрагмент и потому возможно конечное представление графа достижимости (рис. 4 8,в).

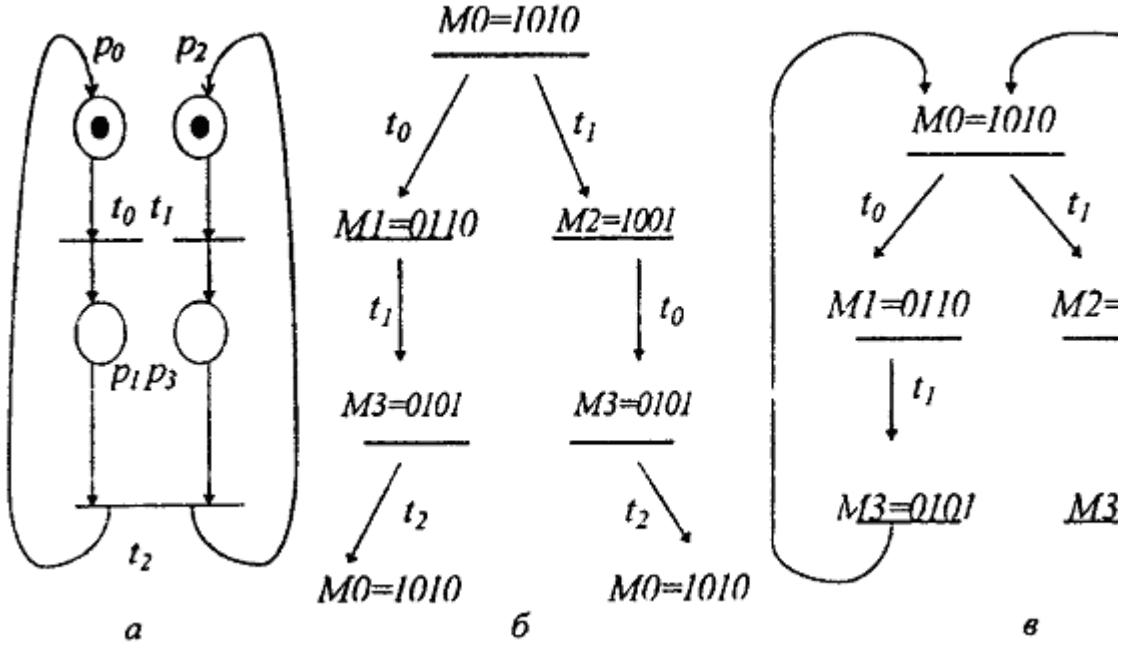


Рис 4.8

Для неограниченной сети и граф развертки и граф достижимости бесконечны. Такой неограниченный граф показан на рис. 4.9,а.

Вообще всякое конечное дискретное устройство, вырабатывающее бесконечный результат (бесконечный язык), обнаруживает некоторого рода регулярность в поведении, что и обеспечивает конечную его представимость (в теории формальных языков этот факт выражается в *pumping* теоремах). Это демонстрируют и примеры графов достижимости.

4.2.4. Дедлоки

Сети Петри оказались удобным средством для анализа такого свойства параллельной системы, как наличие или отсутствие дедлоков. Состояние дедлока возникает, когда запрос ресурсов в системе не может быть удовлетворен и система останавливается (ни один переход не может сработать). Это может быть нормальный останов сети, а может быть и следствие конкуренции за ресурсы.

Пример 4

Рассмотрим пример на рис. 4.9,а. Сеть А определяет циклическое неограниченное срабатывание переходов t_1, t_2, t_3. При срабатывании переходы t_2 и t_3 потребляют единицу ресурса из места p_3 каждый. Можно представить себе для определенности, что место p_3 содержит два участка памяти (буферы), которые операционная система может динамически выделять программе по ее запросу. Пока процесс, управляемый сетью А, выполняется один, ситуации дедлока не возникает. Но если появляется идентичный процесс, выполняющийся параллельно А и управляемый сетью В (рис. 4.9,б), тогда возникает конкуренция за ресурс в месте p_3.

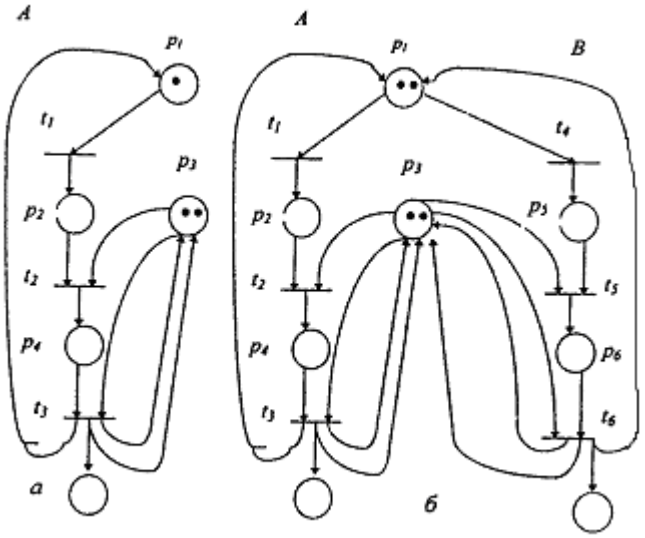


Рис 4.9

Ситуация дедлока возникает при такой последовательности срабатываний переходов сети: t_1, t_4, t_2, t_5. И в этом состоянии имеем дедлок: ни один переход не может сработать. Однако сеть будет нормально функционировать, если в месте p_i оставить одну фишку, т.е. разрешить выполняться либо процессу А, либо процессу В, но не обоим одновременно.

Опасность при таком динамическом (в ходе вычислений) захвате ресурсов вызывает дозахват ресурса (запрос дополнительной порции без освобождения уже захваченных ресурсов). Поэтому чаще всего необходимый ресурс захватывается сразу в нужном количестве, хотя при этом часть захваченного ресурса не может быть сразу использована и простаивает.

Пример 5

Рассмотрим классическую задачу о пяти обедающих философах. Суть задачи такова. Пять философов, прогуливаясь и размышляя, время от времени испытывают приступы голода. Тогда они заходят в столовую, где стоит круглый стол, на нем всегда приготовлены пять блюд. Между соседними блюдами лежит одна вилка (всего лежат ровно пять вилок). Голодный философ:

a. входит в столовую, садится за стол и берет вилку слева;
b. берет вилку справа;
c. ест (обязательно двумя вилками);
d. кладет обе вилки на стол, выходит из столовой и продолжает думать.

При конструировании управления в этой задаче следует учитывать самые разнообразные варианты поведения философов. Рассмотрим некоторые из них.

a. Необходимо организовать действия философов так, чтобы они все были накормлены и не случилось бы так, что пять философов одновременно войдут в столовую, возьмут левую вилку и застынут в ожидании освобождения правой вилки. Голодная смерть всех философов неминуема, если никто из них не захочет расстаться па время со своей левой вилкой. Будет не лучше, если они одновременно положат левые вилки, а затем вновь одновременно попытаются завладеть необходимыми двумя вилками. Результат, понятно, тот же. Типичный дедлок в результате попытки дозахватить ресурс (вилку)!

Грубым (неэффективным) приемом, чтобы избежать этой ситуации, является введение ограничения на число философов, допущенных одновременно в столовую. Например, можно пускать в столовую не более четырех философов одновременно. Тогда один из них, по крайней мере, сможет захватить две вилки, поесть и освободить ресурсы (вилки).

b. Необходимо также предусмотреть, чтобы два философа одновременно не хватали одну и ту же вилку (зная вспыльчивый характер философов, нетрудно предсказать результат).

c. Стеснительный философ не должен умирать голодной смертью из-за того, что его вилки постоянно раньше него хватают напористые соседи.

d. Легко представить себе ситуацию, когда банда сговорившихся философов завладеет всеми вилками и, передавая их только в своей среде, уморит голодом всех прочих.

Сеть Петри, задающая управление распределением вилок обедающим философам, строится пошагово. Вначале была сконструирована сеть, управляющая поведением одного (любого) философа (рис. 4.10). Далее она используется для конструирования сети, управляющей поведением нескольких философов. Пример такой сети для случая трех философов показан на (рис. 4.11). Разделяемые ресурсы - левые и правые вилки - разных фрагментов сети совмещаются. Это управление не решает задач а) - г).

Рис 4.10

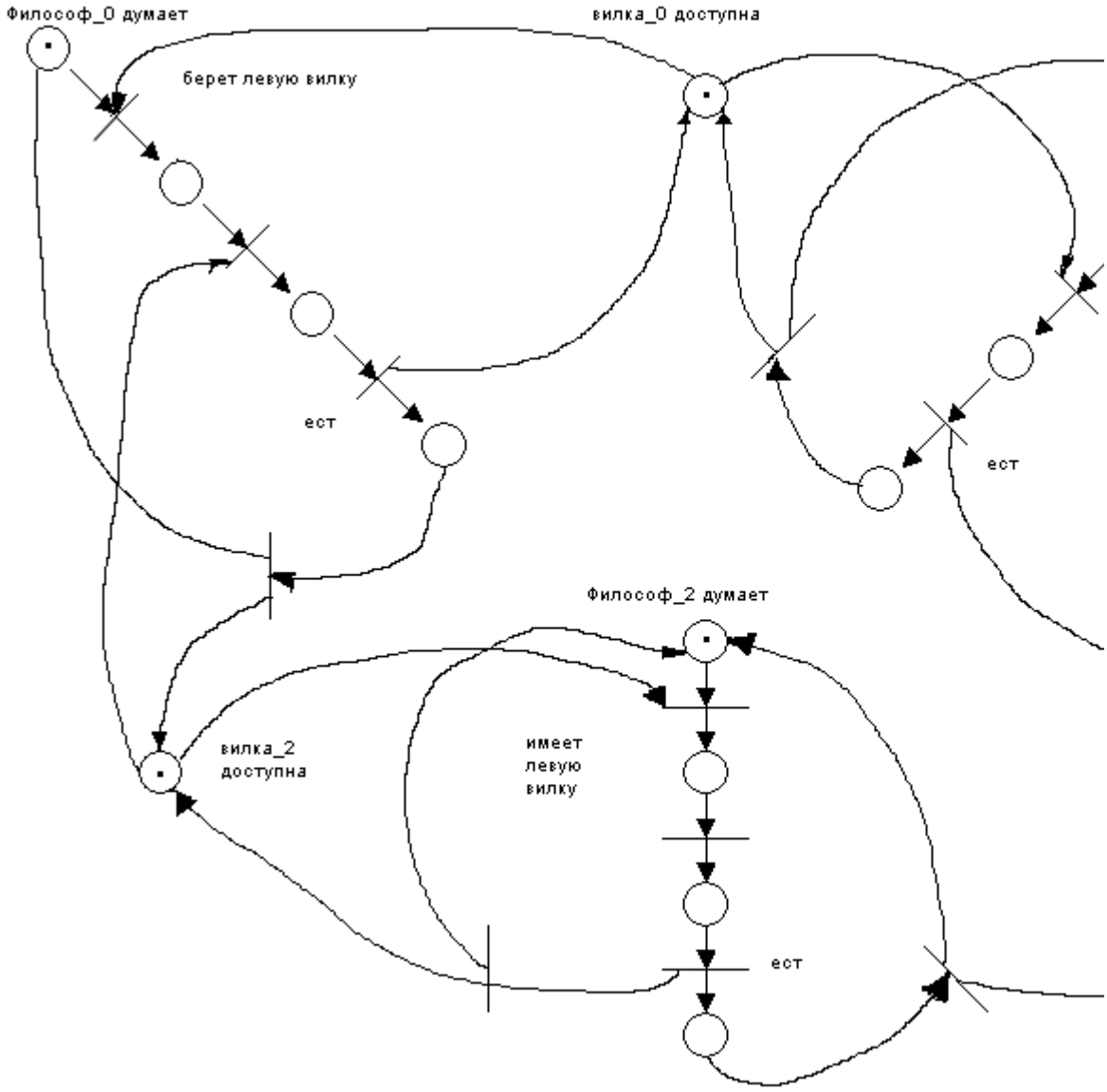


Рис 4.11

Пример 6

Другой пример взаимодействующих процессов показан на рис. 4.12,а. Здесь производитель П производит детали и оставляет их на складе t_5, а потребитель Т забирает их со склада, когда они там есть. Регулирует это взаимодействие место t_5. Если необходимо принять во внимание ограниченную вместимость склада, тогда в сеть добавляется место t_6 (рис 4.12,б). Оно определяет наличие места для хранения 10 деталей. Взятие фишки из места t_6, может интерпретироваться как взятие разрешения поместить очередную деталь на склад (есть свободное место для хранения).

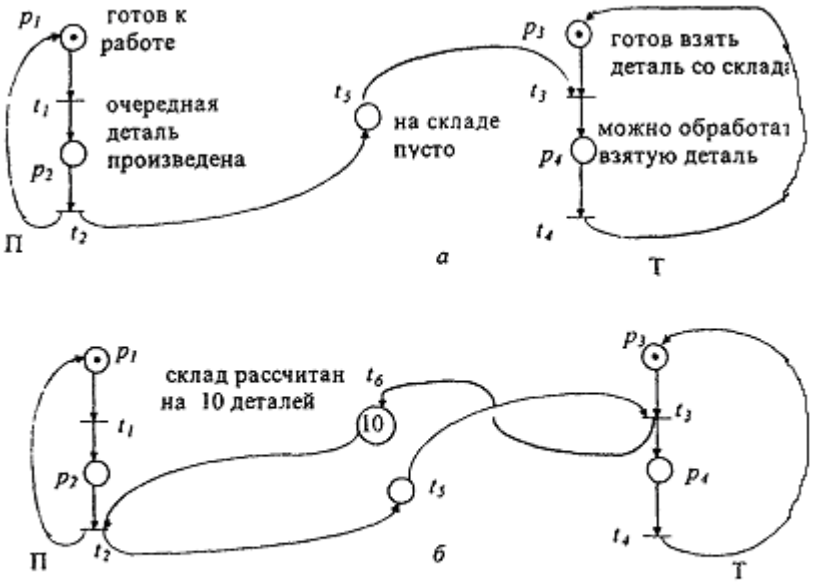


Рис 4.12

Сети Петри очень удобны для задания прямого управления в теоретических работах при исследовании параллелизма. К сожалению, их трудно оказалось использовать в языках программирования как в силу большой времяемкости моделирования их поведения, так и в силу сложности конструирования и отладки заданного ими прямого управления.

4.3. MESSAGE PASSING INTERFACE

Message passing interface (MPI) - это метод программирования межпроцессных коммуникаций. В разных формах MPI изучался в асинхронных вычислениях с конца 60-х годов как у нас в стране, так и за рубежом. MPI использовался в нескольких коммерческих вычислительных системах. В настоящее время можно назвать рабочую станцию SP2 фирмы IBM с мультикомпьютерным ускорителем на базе микропроцессора PowerPC (до 128 процессорных элементов), в которой MPI является основным средством параллельного программирования. Именно на базе этого мультикомпьютера создана, в частности, шахматная программа IBM Deep Blue.

4.3.1. Определение MPI

Предполагается, что программа определяется как система независимых, параллельно протекающих взаимодействующих процессов. Как обычно, здесь процесс - это исполняющаяся программа со всеми обрабатываемыми данными. Исполнение этой же программы с другими данными определяет другой процесс.

Взаимодействие определяется как посылка сообщения из одного процесса в другой и прием сообщения. Сообщение - любая последовательность битов, которая чаще структурируется и определяется как значение набора (кортежа, структуры) переменных. Сообщение посылается в канал в одном процессе и выбирается из канала в другом процессе.

Канал может рассматриваться как очередь. Оператор send, записывает сообщение в канал, в канале может накопиться очередь сообщений. Оператор receive выбирает

сообщение из канала, если оно есть в канале. Если очередь канала не ограничена, мы говорим об асинхронном MPI.

В программе для использования MPI следует прежде всего описать каналы, например:

ch queue (<описание_переменной>).

Описание определяет канал queue, который в состоянии хранить значения переменной канала <описание_переменной>. Переменная канала может быть простой или структурированной. Возможны описания вида:

ch symbol (char);
ch data (day, month, year : integer);
ch personal_date (name : string, age : integer);
Запись сообщения в канал производится оператором send;
send < имя канала > (< выражение >);

Выражение должно вырабатывать значение переменной, описанной в определении канала. Например:

send data (d, m, y);

Переменные d, m и y задаются в процессе-отправителе и должны содержать целые значения, как это и определено в описании канала data. Данные выбираются из канала оператором receive.

receive personal_date (n, a);

Переменные n и a определяются в процессе-получателе и должны быть описаны так же, как переменная канала. После выполнения оператора receive переменная n может содержать значение "Иванов И.И.", а переменная a - значение "30".

Если канал предполагается неограниченным (асинхронный MPI), то оператор send может быть выполнен неограниченно много раз (неблокируемый оператор).

Оператор receive получает значение из канала, если он не пуст. Если в канале нет значения, выполнение оператора receive задерживается (блокируемый оператор) до появления значения в канале. Канал асинхронного MPI реализуется как асинхронный канал.

Каналы описываются как глобальные объекты, поскольку они разделяются процессами программы. Можно определить массив каналов, например:

ch data [l:k] (day, month, year : integer);

Допускается, чтобы несколько процессов посылали сообщения в один и тот же канал. Аналогично, несколько процессов могут получать данные из одного канала. Описанный канал в начальный момент пуст.

Пример 1

Одно из типичных межпроцессных взаимодействий определяет взаимодействия клиента и производителя. Производитель создает продукт и продает его многим клиентам. Программа их взаимодействий может иметь вид

ch request (integer, <описание_запроса_клиента>);
ch reply (integer, <описание_результата>);

<описание_запроса_клиента> - это описание типов переменных, значения которых специфицируют запрос клиента к производителю. Значение переменной, описанной как integer, определяет уникальный номер клиента.

< описание_результата > - описание переменных, специфицирующих ответ производителя на запрос клиента %

Producer :: var index, ...;
do true → receive request (index, ...); % ожидание запроса клиента %
..... % обработка запроса клиента и формирование ответа %

send reply [index] (...); % ответ клиенту % od;
Client [i : 1...n] :: send request (i,...); % обращение к Producer % receive reply [i](...);% ожидание ответа Producer %

В программе определены n+1 независимых параллельно протекающих процессов: один процесс-производитель Producer и n процессов-клиентов Client. Именно так могут быть описаны взаимодействия файл-сервера и программ, запрашивающих (открывающих) нужные им файлы. При этом файл-сервер ответствен за то, чтобы только одна программа получала доступ к файлу по записи. Это же управление годится для программирования удаленного доступа к дисковому файлу нескольких программ в сети. Много подобных задач в разных вариантах возникает при организации обработки данных в сети.

4.3.2. ПАРАЛЛЕЛЬНАЯ ПРОГРАММА РАЗДЕЛЕНИЯ МНОЖЕСТВ

Определения MPI очевидны и просты. Так же просты и очевидны средства программирования в MPI. К сожалению, их простота и очевидность только кажущиеся.

В качестве примера рассмотрим программу разделения множеств, разработанную Э.Дейкстрой. Она много обсуждалась в публикациях, ее частичная корректность была доказана разными авторами, однако лишь недавно (в 1996 г) Ю.Г.Карпов (Санкт-Петербургский технический университет [Ю.Г.Карпов. Анализ корректности параллельной программы разделения множеств. - Программирование, 1996. - N 5.]) формально доказал отсутствие свойства тотальной корректности. Программа называется *корректной*, если при остановке она вырабатывает правильный результат. Программа называется *тотально корректной*, если она всегда останавливается и вырабатывает правильный результат.

И это при том, что программа совершенно тривиальна, в ней попросту не на что смотреть! Это хороший пример, показывающий, что простое тестирование, без формального обоснования правильности программы, не в состоянии обеспечить правильность программы.

Но и применение одних лишь формальных методов не дает хороших результатов по той причине, что правильно применить формальный метод почти столь же трудно, как и разработать сам формальный метод. В литературе известны примеры некорректных программ, чья корректность была формально доказана именно по причине неправильного применения формального метода самими авторами метода.

На практике следует комбинировать и тестирование и формальное доказательство правильности. Следует также обратить внимание на опасную кажущуюся правильность частично корректных, но не тотально корректных программ. Нередко такая программа долгое время нормально работает и обнаруживает ошибку в самый неподходящий момент.

Пусть заданы два множества натуральных чисел S и T. Сохраняя мощность множеств S и T, необходимо собрать в S наименьшие элементы множества $S \cup T$, а в T - наибольшие.

Последовательный алгоритм и программа очевидны: множества S и T сливаются, затем слитое множество упорядочивается и вновь разделяется на множества S' и T', удовлетворяющие условиям задачи.

Для параллельного асинхронного решения задачи используется следующий алгоритм.

1. Определяются два параллельно протекающих процесса *Small* и *Large*.

2. Процесс *Small* выбирает максимальный элемент в множестве S, а процесс *Large* параллельно (в то же самое время) находит минимальный элемент во множестве T.

3. Процессы *Small* и *Large* синхронизируются и обмениваются данными: наибольшее значение множества S пересылается процессом *Small* процессу *Large* для включения в множество T, а наименьшее значение множества T пересылается процессом *Large* процессу *Small* для включения в множество S.

4. Далее циклически повторяются шаги 3 и 4.

5. Программа останавливается, когда наибольший элемент в множестве S окажется меньше наименьшего элемента в множестве T.

По завершении программы все элементы множества S должны оказаться не больше любого элемента множества T, а мощности этих множеств не изменяются.

Программа состоит из двух параллельных процессов, P = [Small\\Large], P = [*Small**Large*]:

Small ::	*Large* ::
mx:=max(S); α! mx; S:=S-{mx};	α?y; T:=T ∪ {y}; mn:=min(T);
β? x; S: = S ∪ {x}; mx:=max(S);	β! mn; T:=T-{mn}; mn:=min(T);
*[mx >x → α! mx; S:=S-{mx};	*[mn<y → α? y; T=T ∪ {y};
β? x; S:=S ∪ {x};	mn:=min(T); β! mn; T:=T-{mn};
mx:=max(S);] stop	mn:=min(T)] stop

Программу можно прокомментировать следующим образом. Определены два процесса *Small* и *Large*. Символ || разрешает параллельное исполнение процессов *Small* и *Large*, оператор * задает циклическое исполнение (итерацию), пока истинно условие цикла. Процессы связаны однонаправленными каналами α и β. По каналу α процесс *Small* передает данные в процесс *Large*, а данные из процесса *Large* передаются в процесс *Small* по каналу β.

Оператор ! задает передачу данных (аналог оператора *send*), а оператор ? - их прием (аналог оператора *receive*). В частности, оператор α!mx в процессе *Small* задает передачу значения переменной mx в канал α, а оператор α?y в процессе *Large* определяет прием значения из канала α и присваивание этого значения переменной *y*.

В программе [Small\\Large] одновременное выполнение оператора α!mx в процессе *Small* и оператора α?y в процессе *Large* (их выполнение синхронизуется, т.е., выполнение одного из них в одном из процессов задерживается до тех пор, пока другой оператор не начнет выполняться в другом процессе) имеет семантику "удаленного присваивания" y:=mx. Аналогична семантика взаимодействия по каналу β.

Обозначим S^0 и T^0 - начальные множества, а S^{Term} и T^{Term} - заключительные их значения. При правильном завершении программы ожидается выполнение соотношений (в соответствии с начальными условиями задачи):

(C1). Объединение множеств не изменилось: $S^{Term} \cup T^{Term} = S^0 \cup T^0$; (C2). Мощности множеств сохранились: $|S^{Term}| = |S^0|$, $|T^{Term}| = |T^0|$; (C3). Каждый элемент S^{Term} не больше любого элемента T^{Term} : $\max(S^{Term}) \le \min(T^{Term})$.

Частичная корректность этой программы состоит в том, что если множества S^0 и T^0 конечны и непусты, то после нормального *завершения* программы (т.е. когда каждый из

процессов выходит на свой *stop*) свойства (C1), (C2) и (C3) выполняются. Тональная корректность ее состоит в том, что если множества S^0 и T^0 конечны и непусты, то программа завершается правильно и свойства (C1), (C2) и (C3) непременно выполняются после этого завершения.

Оставляя в стороне формальные детали, весьма поучительно рассмотреть технологические приемы, приводящие к пониманию того, что программа не является тотально корректной.

4.3.3. КОММУНИКАЦИОННО-ЗАМКНУТЫЕ СЛОИ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

Это понятие вводится для упрощения верификации (доказательства правильности) параллельных программ. Основная идея здесь - это разбиение каждого процесса Pi параллельной программы P:: = [P1|| ... || Pn] на последовательность его операторов: Pi = Qi,1;Qi,2; ... Qi,k (k может быть выбрано одним и тем же для всех процессов, если допустить возможность использовать в качестве Qi,r пустой оператор). Таким образом, параллельная программа P может быть представлена как

P = [(Q1,1; Q1,2; ...; Q1,k)|| ... ||(Qi,1; Qi,2; ...; Qi,k)|| ... || (Qn,1; Qn,2; ...; Qn,k)].

Эту программу можно изобразить матрицей (рис. 4.13).

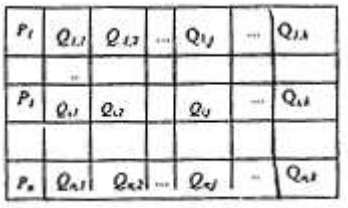


Рис. 4.13

Каждая i-я строка изображает процесс Pi как последовательность операторов:Pi = Qi,1; Qi,2; ...; Qi,k.

Параллельная программа Lj = [Q1,j||Q2,j|| ... || Qn,j] называется j-м слоем параллельной программы P (j-й столбец матрицы).

Слой Lj называется коммуникационно-замкнутым, если при всех вычислениях P взаимодействие процессов P1|| ...|| Pn происходит только внутри этого слоя, или, иными словами, ни одна команда взаимодействия среди операторов Qrj при всех выполнениях P не будет синхронизироваться (сочетаться) с командами взаимодействия из операторов Q1,i при $i \neq j$.

Тогда последовательность слоев L1; ...; Lk представляет собой параллельную программу:

P* = L1; ...; Lk = [Q1,1||Q2,1 || ... ||Qn,1]; ...; [Q1,k|| Q2,k|| ... ||Qn,k].

В программе P* все Lj исполняются последовательно в порядке перечисления, а операторы каждой Lj исполняются параллельно. Программа P* называется безопасной, если и только если все ее слои коммуникационно-замкнуты.

Если программа P* безопасна, то вместо верификации всей параллельной программы P можно проводить ее послойную верификацию, т.е., доказывать утверждение {p0}L1{p1}, ... ,{pk-1}Lk{pk} вместо утверждения {p0}P{pk}. Здесь, как обычно, {s}P{q} обозначает

утверждение, что программа P частично корректна по отношению к предусловию s и постусловию q (вход-выходные соотношения), при этом если до начала исполнения программы P предикат s истинен, то после исполнения P предикат q тоже истинен.

Таким образом, если параллельную программу P удается разбить на последовательность коммуникационно-замкнутых слоев, то доказательство ее (частичной) корректности сводится к последовательному доказательству вход-выходных соотношений для каждого слоя. Это существенно упрощает анализ корректности параллельной программы.

Процессы Small и Large разбиваются на коммуникационно-замкнутые слои совершенно естественно. Разбиение приведено на рис. 4.14.

На рисунке показаны только "синхронизационные скелеты" параллельных процессов. В первый слой входят операторы до цикла, во второй слой - операторы внутри цикла, третий слой составляет оператор stop после выхода из цикла. Процесс правильно завершается, если он заканчивает вычисления в заключительном состоянии: процесс Small в состоянии ES, а процесс Large в состоянии EL. В общем случае процессы могут:

a. оба завершиться нормально;

b. оба не завершиться;

c. один завершится, а другой нет, например, при невозможности выполнить операцию синхронного взаимодействия.

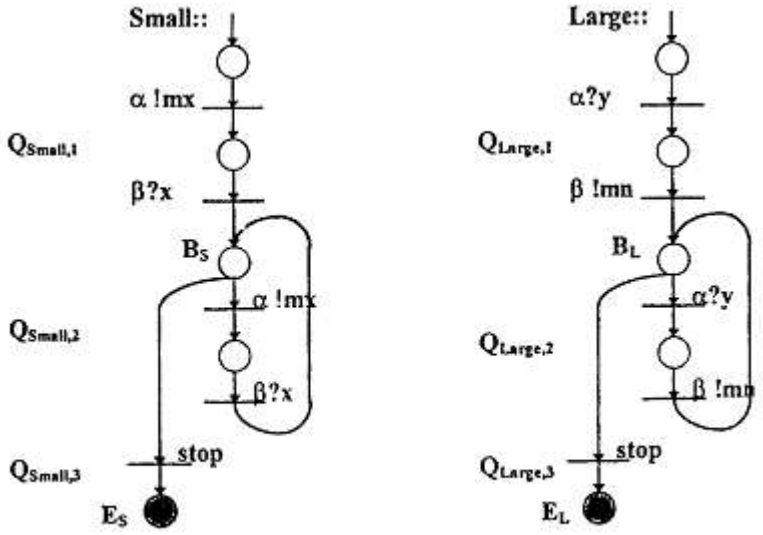


Рис. 4.14

Условия BS и BL определяют условия окончания циклов в процессах; они равны соответственно $mx \le x$ и $mn \ge y$.

4.3.4. Когерентность параллельных программ

Для упрощения верификации параллельной программы уже при проектировании на нее можно наложить дополнительные требования, облегчающие ее понимание и анализ, и заранее устраняющие некоторые типы возможных ошибок. Одним из таких требований является когерентность параллельной программы.

Неформально требование когерентности означает:

- каждый раз, когда процесс хочет послать сообщение другому процессу, его партнер готов принять это сообщение, т.е. обязательно выходит на оператор приема сообщения;

• каждый раз тогда, когда процесс хочет получить сообщение некоторого типа от другого процесса, его партнер посылает ему это сообщение.

В когерентной программе невозможна ситуация, когда в одном процессе выполняется оператор α!mx, а процесс-партнер не может выйти на исполнение оператора α?y.

Если параллельная программа P:: = [P1||...|| Pn] разбита на коммуникационно-замкнутые слои Pi = Q1,i;Qi,2; ...; Qi,k, то требование когерентности состоит не только в том, что при всех вычислениях P ни одна команда взаимодействия среди операторов Qrj при всех выполнениях P не будет синхронизироваться (сочетаться) с командами взаимодействия из операторов Q1,i при i ≠ j, но и в том, что каждая такая команда взаимодействия обязательно будет сочетаться с некоторой командой взаимодействия из операторов того же самого слоя. Для параллельной программы P = [Small \\ Large] такая синхронизация показана на рис. 4.15.

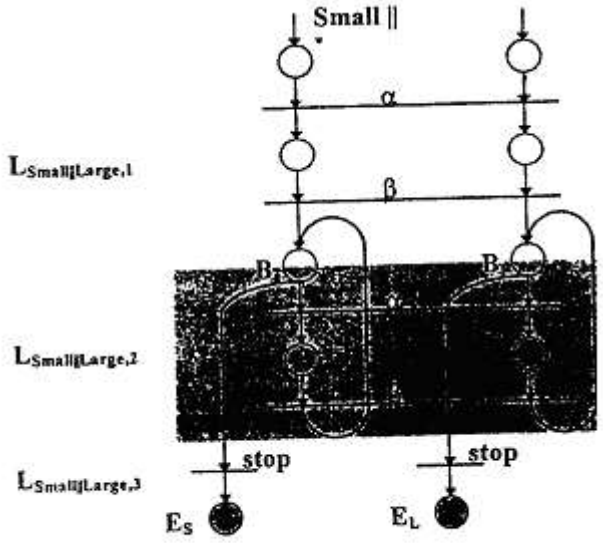


Рис. 4.15

Очевидно, что требование когерентности для этой программы выполняется тогда и только тогда, когда условия прекращения циклов в программах Small и Large тождественны при всех прохождениях циклов.

Некогерентность, с другой стороны, ведет к блокировке этой программы, т.е. к возникновению ситуации, когда один из процессов выходит из цикла и переходит в заключительное состояние, а другой не выходит из своего цикла и "зависает" на операции синхронного взаимодействия внутри цикла, бесконечно ожидая партнера.

Таким образом, условие BS ≡ BL ≡ И, или, что то же, mx ≤ x ≡ mn ≥ y при каждой проверке условий циклов в обеих программах, является необходимым условием тотальной корректности этой параллельной программы.

4.3.5. АНАЛИЗ ПРОГРАММЫ РАЗДЕЛЕНИЯ МНОЖЕСТВ

Для анализа программы используем "истории" взаимодействий. Вводятся вспомогательные переменные, которые хранят истории взаимодействия по каждому каналу программы.

Историческая переменная - это просто массив значений, последовательно переданных по соответствующему каналу. Пусть hα и hβ такие исторические переменные для каналов α и β соответственно, тогда компонент hα[k] содержит k-е значение, посланное по каналу α при выполнении операции α!e.

Проведем анализ первого слоя :

$Q_{Small,1}$ ::	$Q_{Large,1}$ ::
mx:=max(S); α! mx; S:=S-{mx};	α?y; T:=T ∪ {y}; mn:=min(T);
β? x; S: = S ∪ {x};	β! mn; T:=T-{mn};
mx:=max(S);	mn:=min(T)

Для того чтобы процессы QSmall,1 и QLarge,1 завершились, необходимо и достаточно, чтобы множество S содержало хотя бы один элемент, т.е. |S| > 0. По завершении каждого из этих параллельных процессов первого слоя будут справедливы следующие соотношения:

для $Q_{Small,1}$ ::	для $Q_{Large,1}$ ::
$mx^0=max(S^0)$;	$y^1 = h_\alpha[0]$
$h_\alpha[0] = mx^0$;	mn^0; = $min(T^0 \cup \{y^1\})$;
$x^1 = h_\beta[0]$	$h_\beta[0] = mn^0$
$S^1 = (S^0 - \{max (S^0)\}) \cup \{x^1\}$;	$T^1 = T^0 \cup \{y^1\} - \{ min(T^0 \cup \{y^1\})\}$;
$mx^1 = max(S^1)$;	$mn^1=min(T^1)$;

Эти соотношения просто описывают, что было сделано при исполнении операторов первого слоя.

Рассмотрим теперь второй слой:

$Q_{Small,2}$ ::	$Q_{Large,2}$ ::
*mx > x → α! mx; S:=S-{mx};	*[mn < y → α?y; T:=T ∪ {y}; mn:=min(T);
β? x; S: = S ∪ {x};	β! mn; T:=T-{mn};
mx:=max(S);	mn:=min(T)
]	]

Перед i-м выполнением каждого цикла для процессов QSmall.2 и QLarge 2 истинны следующие инварианты, что можно проверить непосредственно (где ai -значение переменной a перед i-м выполнением цикла):

для $Q_{Small,2}$ ::	для $Q_{Large,2}$ ::
$I_{Small,2} \equiv h_\alpha [i-1] = mx^{i-1} \wedge$	$I_{Large,2} \equiv y^i = h_\alpha[i-1] \wedge$
$x^i = h_\beta[i-1] \wedge$	$h_\beta[i-1] = min(T^{i-1} \cup \{y^i\}) \wedge$
$S^i = (S^{i-1} - \{max (S^{i-1})\}) \cup \{x^i\}$;	$T^i = T^{i-1} \cup \{y^{i-1}\} - \{ min(T^{i-1} \cup \{y^i\})\}$
$\wedge$	$\wedge$
$mx^i = max(S^i)$;	$mn^i=min(T^i)$;

Как уже говорилось, требование когерентности в этой программе соответствует требованию общезначимости формулы mxi ≤ xi ≡ mni ≥ yi. При некоторых значениях исходных множеств S и T она нарушается. Возможны два случая некогерентности:

a. mxi ≤ xi; mni < yi; или, что то же: max(Si) ≤ min(Ti-1 ∪ { max(Si-1)}); min(T i) < max (S i-1);

при этом процесс Small завершается, а процесс Large продолжает выполнять цикл, что приводит к его блокировке;

b. mxi > xi или, что то max(Si) > min(T i-1 ∪ {max(Si-1)}); mni ≥ yi же: min(Ti) ≥ max(S i-1);

при этом процесс Large завершается, а процесс Small продолжает выполнять цикл и блокируется, бесконечно ожидая взаимодействия с процессом Large.

Учитывая, что min(Ti) ≥ min(Ti) и max(Si) ≤ max(S i-1), условия некорректного поведения параллельной программы разделения множеств можно записать проще:

а) $\max(S^{i}) \le \min(T^{i-1})$; $\min(T^{i}) < \max(S^{i-1})$;

б) $\max(S^{i}) > \min(T^{i-1})$; $\min(T^{i}) \ge \max(S^{i-1})$.

Поясним полученный результат. Исследуемая параллельная программа разделения множеств имеет целью собрать все минимальные элементы объединения двух множеств S и T в множестве S, а максимальные элементы S ∪ T - в множестве T, причем мощности множеств не должны измениться. Упорядочим элементы исходных множеств: множества S по убыванию, а множества T по возрастанию. На рис. 4.16,а. показано, что процесс Small, работая на множестве S, пересылает его максимальные элементы в множество T, а процесс Large, работая на множестве T, пересылает его минимальные элементы в множество S.

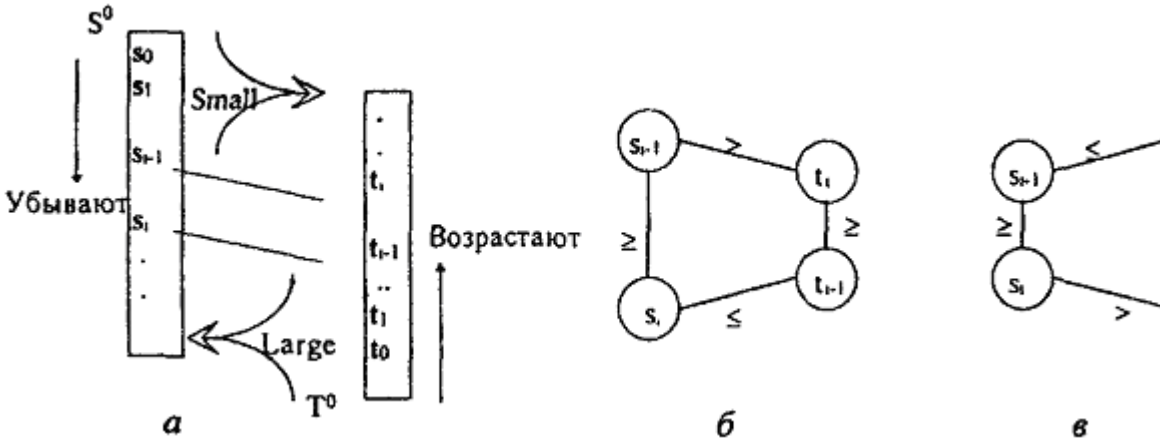


Рис. 4.16

Полученные выше условия некорректного поведения программы определены для (i-l)-го и i-го максимальных значений множества S и для (i-l)-го и i-го минимальных значений множества T, (i =1,2,...). Эти условия представлены диаграммами на рис. 4.16,б) и 4.16,в).

Иными словами, если между упорядоченными по убыванию элементами множества S и упорядоченными по возрастанию элементами множества T на одном и том же расстоянии от начала выполнится одно из отношений рис. 4.16,б) или рис. 4.16,в), то исследуемая программа будет работать некорректно: она входит в дедлок.

Можно указать тестовый пример, на котором эта программа работает некорректно: S = {5,10,15,20}, T = {17,18,30,40,60}. Этот пример относится к первому типу некорректностей при i = l: первое же вхождение процесса Large в цикл приводит к дедлоку, поскольку Small завершится, не входя в цикл.

Однако полагаться на возможность обнаружения этой некорректности с помощью тестов нельзя. Добавление, например, к множеству T любого числа элементов, меньших 15, нарушит это соотношение и программа будет работать корректно.

В частности, ручной прокруткой можно проверить, что на множествах S = {5,10,15,20}, T = {14,17,18,30,40,60} программа работает правильно: сортирует множества и завершается после однократного прохождения циклов в обоих процессах.

Список литературы

1. Анисимов В. Программирование распределенных вычислительных систем / Под ред. В.Е.Котова. Системная информатика 3. - М.: Наука, 1993. - с. 210-247.
2. Вальковский В., Малышкин В.А. Синтез параллельных программ и систем на вычислительных моделях. - Новосибирск: Наука, 1988. - 128 с.
3. Евстигнеев А. Введение в параллельные архитектуры ЭВМ. - Изд-во НГУ, 1994. - 78 с.
4. Евстигнеев А. VLIW-машины: развитие архитектуры и принципов построения программного обеспечения. Системная информатика N 4 / Под редакцией И. Поттосина. - Новосибирск: Наука - С. 304-333.
5. Евстигнеев А. Основы параллельной обработки. Анализ программных зависимостей. - Новосибирск: Изд-во НГУ, 1996. - 75 с.
6. Ершов А. Вычислимость в произвольных областях и базисах // Семиотика и информатика. - 1982. - Вып.19. - С. 3-58.
7. Клини С. Введение в метаматематику. - М.: Изд-во иностр. лит., 1957 (S.Kleene. Introduction to metamathematics/ - New York: Van Nostrand, 1952).
8. Лавров И.А., Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов. - М.: Наука, 1984. - 223 с.
9. Малышев А. Алгоритмы и рекурсивные функции. - М.: Наука, 1986.
10. Программирование на параллельных вычислительных системах / Под редакцией Р. Бэбба. - М.: Мир, 1991.
11. Малышкин В. Линеаризация массовых вычислений // Системная информатика N 1, 1991 / Под редакцией В.Е. Котова. - Новосибирск: Наука -С.229-259.
12. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. - М.: Мир, 1985.
13. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. - М.: Мир, 1972. (Hartley Rogers. Jr, Theory of Recursive Functions and Effective Computability, Me Graw - Hill, 1967).
14. Сачков В.Н. Комбинаторные методы дискретной математики. - М. Наука, 1977.-317 с.
15. Успенский В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. - М.:Наука, 1987. - 288 с.
16. Шашкин Ю. Неподвижные точки. - М.: Наука, 1989. - 79 с.
17. Brawl T. Parallel Programming. An Introduction. Prentice Hall, 1993
18. Endrews G. Concurrent Programming. Principles and Practice, Benjamin /Cummings Publishing, 1991.
19. Greene D., Knuth D. Mathematics for analysis of algorithms, Birkhauser, 1982 (Д.Грин, Д.Кнут. Математические методы анализа алгоритмов. - М.: Мир, 1987,119 стр)
20. Hoare G. Communicating sequential processes, Prentice-Hall (Ч. Хоар. Взаимодействующие последовательные процессы. - М.: Мир, 1989. - 264 с.).
21. Lester B. Art of Parallel Programming. Prentice Hall, 1993.

Quinn M. Parallel Computing. McGraw-Hill, 1994.

Лабораторные задания

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Методические указания по выполнению лабораторных работ для магистров направлений подготовки «ПОВТАС» по профилю «Информатика и вычислительная техника»

СОДЕРЖАНИЕ

1. ЛАБОРАТОРНЫЕ РАБОТЫ И ИХ НАИМЕНОВАНИЕ

Тема 1. РАЗРАБОТКА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ АЛГОРИТМОВ И ПРОГРАММ, ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ИХ ВЫПОЛНЕНИЯ НА СОВРЕМЕННЫХ ЭВМ

Цель работы: Научиться оценивать скорость выполнения программ и освоить приемы повышения производительности

Описание работы: При написании и оптимизации программ, содержащих большой объем вычислений, необходимо располагать информацией о времени выполнения того или иного фрагмента кода. На основании этих данных можно выбрать как наиболее быстродействующий алгоритм для данной вычислительной платформы, так и наилучшую вычислительную платформу для данного алгоритма.

Для получения временных характеристик выполнения программы могут быть использованы как подпрограммы из стандартных библиотек языка программирования (функции *time(), clock()* стандартной библиотеки языка С), так и специальные программные средства производителей аппаратного обеспечения *(Intel VTune, AMD Code Analyst).* Однако, первые могут не обладать требуемой точностью, а вторые обычно используются на этапе тонкой оптимизации критических к скорости выполнения участков кода.

Удобным средством контроля над выполнением программ в *ОС Windows* является *Task Manager* (Диспетчер задач). В числовом и графическом виде представлена информация о загрузке процессора(-ов), времени выполнения процессов, объеме доступной и использованной памяти и многое другое. Пользователь может изменять приоритет выполнения процессов. На многопроцессорной ЭВМ есть возможность выбора процессора, на котором будет выполняться программа.

На программном уровне операционная система *Windows* содержит стандартные функции для измерения времени выполнения процесса. В даль-

нейшем приведены примеры вызовов функций и программ на языке С. Для работы с данными функциями необходимо подключить заголовочный файл <Windows.h>.

Счетчик производительности. Служит для измерения интервалов времени в масштабе всей системы с высокой точностью.

Прототип функции:

```
BOOL QueryPerformanceCounter( LARGE INTEGER* IpPerfor-
manceCount );
```

Возвращаемое значение: не нуль, если аппаратное обеспечение поддерживает счетчик производительности, нуль - в противном случае.
lpPerformanceCount: 64 битный выходной параметр - значение счетчика производительности.

Прототип функции:

```
BOOL QueryPerformanceFrequency( LARGE INTEGER* IpFrequency);
```

Возвращаемое значение: не нуль, если аппаратное обеспечение поддерживает счетчик производительности, нуль - в противном случае.
lpFrequency: 64 битный выходной параметр - частота счетчика производительности, число тактов в секунду.

Пример 1.

```
//Подключения заголовочного
файла win32 api #include
<Windows.h>

LARGE INTEGER freq;  //64-битная структура
if (!QueryPerformanceFrequency(&freq)) return -1;
        // Выход если счетчик производительности не
        поддерживается

LARGE INTEGER Count1, Count2;
QueryPerformanceCounter(&Count1); // Значение счетчика в
начале

Участок кода для тестирования();
    // Участок кода, время выполнения которого необхо-
димо измерить

QueryPerformanceCounter(&Count2); // Значение счетчика в
конце


// Вычисление времени выполнения, используется
явное приведение типа float time spent = (float)
(Count2.QuadPart - Count1.QuadPart) /
freq.QuadPart;
```

Примечание. Ошибки реализации BIOS или HAL многопроцессорной системы могут привести к неверным результатам, возвращаемым данными функциями.

Пример 2. Использование инструкции_rdtsc процессора. Оценка тактовой частоты процессора.

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

float GetHz(const unsigned int ms)
{
      int64 tStart,
      tEnd; asm
      {
          rdtsc
          mov dword ptr
          [tStart], eax mov
          dword ptr
          [tStart+4], edx
      }

      Sleep(ms);

      asm
      {
          rdtsc
          mov dword ptr
          [tEnd], eax mov
          dword ptr [tEnd+4],
          edx
      }

      exit();

      return (float)(tEnd - tStart)*1000.0f/ms;

}

int main()
{

      for (unsigned int i = 1; i< 401; i++)
      printf(" Current CPU speed is ~%5.2f mhz (%d
ms estimation)\n", GetHz(i)/1.0e6f, i);
      printf("Press <Enter> to
      exit..."); char ch =
      getchar(); return 0;
}
```

Примечание 1. Измерения времени следует проводить на машине, которая не выполняет одновременно другие вычисления, т.к. одновременный доступ нескольких процессов к ресурсам ЭВМ (памяти, процессорному кэшу и т.д.) может сильно повлиять на результаты.

Примечание 2. В зависимости от исследуемого фрагмента программы время его выполнения может сильно отличаться. Поэтому необходимо провести серию вызовов подпрограммы и провести статистическую обработку результатов.

Производительность ЭВМ на конкретном алгоритме можно приближенно измерять в количестве операций, выполняемых в единицу времени. Например, при вычислении произведения матриц по стандартной формуле

$$c_{ij} = \mathbf{X}^{m}_{l=1} a_{il} b_{lj}, =$$

выполняется приблизительно по *n*m*k* операций умножения, сложения и присваивания. Если матрицы содержат значения с плавающей запятой, то можно вычислить следующий показатель - количество операций с плавающей точкой в секунду:

$$„ \ 3nmk_{л} \wedge\_6 - P = \text{—-—}$$

10^6 MFlops

t - время выполнения в секундах.

Особенности современных архитектур: CISC и RISC

Время на решение задачи t = C*T*I, C - число тактов процессора на инструкцию, T = 1/F - время такта, F - тактовая частота, I - число инструкций на задачу

CISC: уменьшаем фактор I, проблемы с C и T

RISC: уменьшаем C и T, проблемы с I Что сейчас? Внутри RISC, снаружи CISC

Что мы можем сделать? Использовать оптимизированные библиотеки и компиляторы, минимизировать количество “дорогих” инструкций. Конвейеризация: разбиваем операцию на стадии, например, исполнение инструкции состоит из Выборки, Декодирования, Исполнения, Записи результатов

При правильной реализации n-стадийный конвейер может одновременно исполнять n последовательных инструкций. В идеале n-стадийный конвейер дает прирост производительности в n раз. На практике: переходы, исключения, прерывания, задержки подсистемы памяти, взаимозависимые инструкции могут разрушить поток инструкций и остановить конвейер.

Что мы можем сделать?

А) Положиться на компилятор Б)

Развернуть циклы

В) Выполнить потактовую ассемблерную оптимизацию

Развертка циклов (loop unroll)

Бонус: внутренний параллелизм, вычисления внутри цикла могут быть выполнены одновременно на суперскалярном процессоре.

Кэш-память. Факт: пропускной способности памяти не хватает процессору (докажите!) Решение: кэш память - быстрое статическое ОЗУ, вставленное между процессором и системной памятью, предназначенное для сохранения последних использованных инструкций и данных. Единый и разделяемый (гарвардский) кэш, уровни: TLB, L1D, L1I, L2, L3 ...

Кэш с прямой записью (write-through) и кэш с обратной записью (writeback). Характеристики, важные для программиста: длина строки, объём кэшпамяти.

Базовый пример: матричное умножение.

Задание 1. Написать и откомпилировать тестовую последовательную программу с достаточно большим временем выполнения, например, с бесконечным циклом. Запустить ее на многопроцессорной ЭВМ. С помощью Диспетчера задач *Windows* получить график загрузки процессоров. Последовательно переключать выполнение задачи на различные процессоры, изучить изменение графика загрузки. Завершить выполнение процесса с помощью Диспетчера задач. Сделать выводы о целесообразности использования стандартных последовательных программ на многопроцессорных системах.

Задание 2. Составить подпрограмму MuitMat перемножения матриц по стандартной формуле. Написать подпрограмму тестирования TestMuitTime, которая по заданным значениям размера *N* квадратных матриц и количества запусков *NT* выводит следующую статистику: минимальное, максимальное и среднее время выполнения, среднее квадратическое отклонение. Использовать функцию QueryPerformanceCounter. По среднему значению вычислить рейтинг производительности в *MFlops*. Провести вычисления при различных значения *N* и *NT*.

Исследовать варианты

1. Статическая память, динамическая память.

2. Оптимизация доступа к памяти (последовательный доступ).

3. Улучшение загрузки вычислительного конвейера (развертка циклов)

4. Оптимизация доступа к памяти (локализация часто используемых данных для систем с многоуровневой иерархией памяти).

5. Оптимизация доступа к памяти (предвыборка данных - software prefetch, hardware prefetch)

6. Использование векторных расширений набора команд процессора (sse2, avx).

Алгоритмы для оптимизации - задачи плотной линейной алгебры:

1. Перемножение матриц и векторов

2. Копирование больших объемов данных

3. Решение СЛАУ, факторизации, обращения матриц

Задание 3. Модифицировать алгоритм перемножения матриц для увеличения быстродействия:

$$b_{ij}^{T} = b_{ji},$$

$$c_{ij} = \sum_{l=1}^{m} a_{il} b_{lj}. \quad (1)$$

Сравнить со стандартным алгоритмом. Вычислить прибавку в скорости.

В формуле (1) вынести вычисление скалярного произведения строк в отдель-

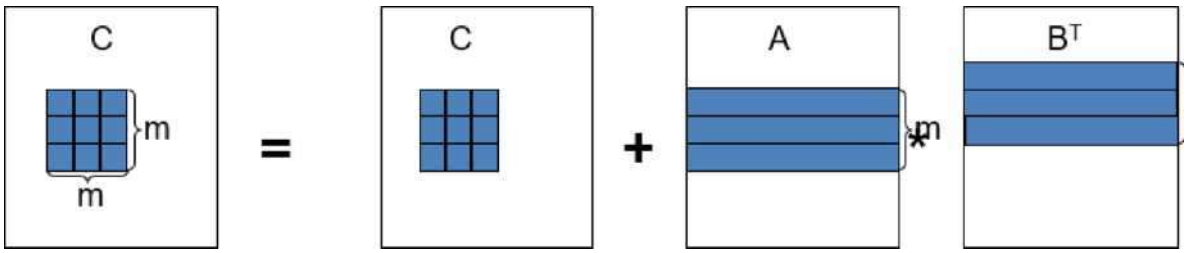
ную подпрограмму DotProd. При вычислении скалярного произведения сделать развертку цикла.

Пример 3.

```
TReal dot prod 8( const unsigned int n, const TVector a,
const TVector b                                          )
{
    TReal s1=0.0, s2=0.0, s3=0.0, s4=0.0;
    unsigned int n8 = (n/8) * 8;
    for (unsigned int i=0; i<n8; i+=8){
        s1 += a[i] * b[i] + a[i+1] *
        b[i+1]; s2 += a[i+2] *
        b[i+2] + a[i+3] * b[i+3]; s3
        += a[i+4] * b[i+4] + a[i+5]
        * b[i+5]; s4 += a[i+6] *
        b[i+6] + a[i+7] * b[i+7];
    }
    for (i=n8; i<n; i++){
        s4 += a[i] * b[i];
    }
    return (s4+s3)+(s2+s1);
}
```

Сравнить время выполнения с предыдущими вариантами. Вычислить ускорение.

Задание 4. Для увеличения скорости перемножения больших матриц модифицируем алгоритм перемножения для локализации данных в процессорном кэше второго уровня. Будем вычислять значения c[i][j] не построчно, а в пределах блоков размерности m x m



Если пренебречь необходимостью чтения и записи блока матрицы C, то m <= 0, 5M/n, M - объем быстрой памяти (число вещественных чисел).

Сравнить время выполнения с предыдущими вариантами. Вычислить ускорение.

Задание 5*. В качестве дополнительного задания повышенной сложности студенту может быть предложено сравнить полученные результаты с вы- сокооптимизированными функциями линейной алгебры из состава свободно распространяемых реализация BLAS, LAPACK в Intel MKL или Boost.

Тема 2. РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ И ПРОГРАММ ДЛЯ ЭВМ С ОБЩЕЙ ПАМЯТЬЮ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ OPENMP

Цель работы: Научиться создавать программы, использующие модель параллельных вычислений с общей адресной областью памяти и технологию параллельного программирования OpenMP.

Платформа: Многопроцессорная или многоядерная ЭВМ с установленной средой разработки для языка C/C++, OpenMP-совместимый транслятор с языка высокого уровня.

Описание работы:

Технология OpenMP применяется при разработке параллельных программ для систем с общей памятью. Предоставляет простой способ использовать для вычислений ресурсы нескольких вычислителей, без необходимости использования специфичного API операционной системы. Менее гибкое решение, чем прямое использование многопоточного API операционной системы.

Переносимость - компилятор, поддерживающий OpenMP создаст код, который нужен для работы с конкретной ОС. Стандарт де-факто для систем с общей памятью. Текущая версия стандарта - 4.5 (http://www.openmp.org). Поддерживается несколькими компиляторами, например: Intel, GCC, SUN, Microsoft.

OpenMP представляет собой расширение исходного языка программирования. Конструкции OpenMP должны поддерживаться компилятором. Расширение исходного языка выполнено с учетом обратной совместимости - если компилятор не поддерживает OpenMP, то в результате получится обычная последовательная программа. Стандарт предусматривает поддержку двух языков программирования - Fortran и C/C++. Директивы OpenMP представляют собой для C/C++ - директивы препроцессора #pragma omp, для Fortran

- специального вида комментарии.

Более подробное изложение теоретической части в лекциях, рекомендованной литературе в конце данного пособия, а также в тексте стандарта OpenMP.

В качестве задания студентам предлагается распараллелить и повысить быстродействие последовательных программ, разработанных в лабораторной работе №1.

Тема 3. РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ И ПРОГРАММ ДЛЯ ЭВМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ MPI

Цель работы: Научиться создавать программы, использующие модель параллельных вычислений с разделяемой адресной областью памяти и библиотеку обмена сообщений *MPI*.

Платформа: Многопроцессорная ЭВМ либо вычислительный кластер с установленной средой параллельного программирования *MPI, MPI*— совместимый транслятор с языка высокого уровня.

Описание работы:

Компьютеры с массовым параллелизмом (MPP) представляют собой многопроцессорные системы с распределенной памятью, в которых с помощью некоторой коммуникационной среды объединяются однородные вычислительные узлы. Каждый из узлов состоит из процессора, собственной оперативной памяти, коммуникационного оборудования, подсистемы ввода/вывода, т.е. обладает всем необходимым для независимого функционирования. Доступ к памяти других процессоров в таких системах, как правило, возможен только с помощью механизма передачи сообщений. Основное достоинство таких систем - это высокая степень масштабируемости. Однако межпроцессорные обмены данными в компьютерах этого типа выполняются намного медленнее, чем локальная обработка данных самими процессорами. Поэтому написание эффективных программ для таких компьютеров представляет собой сложную задачу, а для некоторых алгоритмов вообще невозможно.

Кластерные технологии стали логическим продолжением развития идей, заложенных в архитектуре MPP систем. Если процессорный модуль в MPP системе представляет собой законченную вычислительную систему, то следующий шаг напрашивался сам собой: почему бы в качестве таких вычислительных узлов не использовать обычные серийно выпускаемые компьютеры. Развитие коммуникационных технологий, а именно, появление высокоскоростного сетевого оборудования и специального программного обеспечения, такого как MPI, реализующего механизм передачи сооб-

щений над стандартными сетевыми протоколами, сделали кластерные технологии общедоступными. Сегодня не составляет большого труда создать небольшую кластерную систему, объединив вычислительные мощности компьютеров отдельной лаборатории или учебного класса.

Кластер - это связанный набор полноценных компьютеров, используемый в качестве единого ресурса. Существует два подхода при создании кластерных систем:

- в единую систему объединяются полнофункциональные компьютеры, которые могут работать, в том числе, и как самостоятельные единицы, например, компьютеры учебного класса или рабочие станции лаборатории;
- целенаправленно создается мощный вычислительный ресурс, в котором роль вычислительных узлов играют промышленно выпускаемые компьютеры, и тогда нет необходимости снабжать такие компьютеры графическими картами, мониторами, дисковыми накопителями и другим периферийным оборудованием, что значительно удешевляет стоимость системы.

Для создания кластера используются компьютеры, которые могут представлять собой как простые однопроцессорные системы, так и обладать сложной архитектурой SMP.

Разработано множество технологий соединения компьютеров в кластер. Наиболее простым вариантом является использование технологии Ethernet, однако за эту простоту приходится расплачиваться заведомо недостаточной скоростью обменов. В MPP системах на каждом вычислительном узле функционирует собственная копия операционной системы, под управлением которой выполняется независимая программа. Единственно возможным механизмом взаимодействия между ними является механизм передачи сообщений. В 1994 г. был принят стандарт механизма передачи сообщений MPI (Message Passing Interface). Основная цель, которую ставили перед собой разработчики MPI - это обеспечение полной независимости приложений, написанных с использованием MPI, от архитектуры многопроцессорной системы. Реализации MPI успешно работают не только на классических MPP системах, но также на SMP системах и на сетях рабочих станций (в том числе и неоднородных).

MPI - это библиотека функций, обеспечивающая взаимодействие параллельных процессов с помощью механизма передачи сообщений, с интерфейсом для языков C и FORTRAN. Она включает в себя множество функций передачи сообщений типа точка - точка, функции для выполнения коллективных операций и управления процессами параллельного приложения. Основное отличие MPI от предшественни-

ков в том, что явно вводятся понятия групп процессов, с которыми можно оперировать как с конечными множествами, областей связи и коммуникаторов, описывающих эти области связи. Это предоставляет программисту очень гибкие средства для написания эффективных параллельных программ.

Однако, несмотря на значительные успехи в развитии технологии программирования с использованием механизма передачи сообщений, трудоемкость программирования с использованием этой технологии все-таки велика.

WMPI - одна из реализаций MPI, функционирующая в ОС Windows.

Краткая информация о настройке wmpi для запуска параллельной программы:

Конфигурационный файл wmpi.clusterconf должен находиться в каталоге с исполняемым файлом, если не указано другое имя в переменной окружения WMPI_CLUSTER_CONF_FILE.

Формат файла wmpi.clusterconf:

Сначала задается конфигурация машин после строки /Machines

```
<Windows machine name>
<device1> <machine id using the device1>
<device2> <machine id using the device2>
<device3> <machine id using the device3>
```

Имя машины можно посмотреть в Start->Settings->Control Panel- >Network->Identification. Для каждой машины указываются устройства взаимодействия.

Например, для компьютера TS, у которого есть устройства взаимодействия tcp и shmem:

```
TS
tcp
TS.povt.npi-
tu.ru shmem TS
```

Затем необходимо указать, как процессы на каждой машине будут взаимодействовать друг с другом. По умолчанию могут быть назначены устройства, работающие на одной машине (internal), и на другой машине (external)

```
[internal device <device>]
[external device <device>]
[<Windows machine1> <Windows machine2> <device>]
```

Пример:

```
/Machines
TS
tcp TS.povt.npi-tu.ru
shmem TS
107w2k01
shmem 107w2k01
```

```
tcp 107w2k01.povt.npi-tu.ru
111w2k03
tcp 111w2k03.povt.npi-tu.ru
#Note: e.g. this machine has no
shared memory device /Devices
internal device
shmem external
device tcp
111w2k03
111w2k03 tcp
```

В данном примере описаны 3 машины кластера, причем для машины `111w2k03` не используется shmem устройство.

Для указания компьютеров, которые будут участвовать в вычислениях, для каждой программы создается файл, который должен иметь то же имя, что и исполняемый файл и расширение *.pg.

Process Group File.

```
<Windows Machine Name> <number of procs> <executable path> [arguments] <Windows Machine Name> <number of procs> <executable path> [arguments]
```

Например, `pi_calc.pg 107w2k01 1 "\\MAIN\Predmets\Buzalo\parallel\lab4\pi calc\pi calc.exe"`

```
TS 2 "\\MAIN\Predmets\Buzalo\parallel\lab4\pi calc\pi calc.exe"
```

В данном примере всего будет создано 4 (не 3!) процесса, причем первый главный процесс выполняется на машине, на которой произошел запуск программа. В прописанных в Process Group файле путях должны содержаться следующие библиотеки.

wmpi.dll - служебная библиотека среды WMPI.

wmpi_shmem.dll - библиотека устройства shmem (общая память) (опционально).

wmpi_tcp.dll - библиотека устройства tcp (надстройка для взаимодействия по tcp протоколу).

Кроме того, для запуска на выполнение программы, каждая удаленная машина должна исполнять программу tcp_daemon.exe, обеспечивающую подключение к данной машине.

Задание 1. Настроить WMPI под необходимую конфигурацию кластера.

Задание 2. Параллельное вычисление числа Pi путем численного интегрирования.

pi = int(0, 1, f(x)) ~ h*sum(f(x[i+1/2]), i=0..n-1), f(x) = 4/(1+x*x).

Задание 3. Оценка латентности и пропускной способности устройств WMPI (shmem и tcp).

Латентность - время (сек) на передачу сообщения, минимального размера. Пропускная способность - максимальный объем передаваемых данных в единицу времени (MB/sec)

Варианты заданий

Помимо классических задач, решение которых представляет интерес в рамках каждой лабораторной работы, студентам выдаются индивидуальные задания, которые выполняются в рамках изучения курса параллельных вычислений с 1 по 3 лабораторную работу. Представляет практический интерес сравнение особенностей и эффективности реализации одной задачи с использованием трех разных подходов: а) высокоэффективная последовательная реализация, б) параллельная реализация в модели с общей памятью, в) параллельная реализация в модели с распределенной памятью.

Задание 1

Умножение разрежённых матриц. Матрица хранится ненулевыми элементами строки. Сравнить производительность программы для матриц разного размера и при разном количестве рабочих процессов.

Задание 2

Пусть дана разрежённая матрица, представленная ненулевыми элементами строки. Найти транспонированную матрицу. Сравнить производительность программы для матриц разного размера и при разном количестве рабочих процессов.

Задание 3

Задача аппроксимации интеграла непрерывной функции $f(x) = \sin x e^x$ на интервале $[a,b]$. Интервал разбить на подынтервалы по количеству процессоров, для каждого использовать алгоритм адаптивной квадратуры. Алгоритм адаптивной квадратуры:

1. Вычислить m — середину отрезка $[a,b]$.

2. По правилу трапеций или по правилу Симпсона найти аппроксимацию интеграла на отрезках *[a,m]*, *[m,b]* и [a,b].
3. Если сумма меньших площадей равна большей с некоторой заданной точностью, то аппроксимацию считаем достаточной.
4. Если нет, то процесс повторяется для отрезков *[a,m]* и [m,b].

Задание 4

Задача о 8 ферзях. Нужно расставить на шахматной доске 8 ферзей так, чтобы они не атаковали друг друга. Найти все 92 решения (или требуемое количество), используйте модель «управляющий-рабочие». Управляющий задаёт начальные позиции ферзей. Рабочий процесс ищет все решения, результат либо передаёт управляющему, либо выводит в файл.

Задание 5

Задача подсчёта количества уникальных слов в словаре (ни одна буква не повторяется). Использовать модель «управляющий-рабочие» или «взаимодействующие равные». Кроме этого вывести самые длинные слова.

Задание 6

Задача выделение областей изображения. Представить изображение матрицей чисел, определяющих цвет пикселя. Определить количество областей, составляющих изображение, используя алгоритм пульсации для межпроцессорного взаимодействия. Два пикселя принадлежат одной области, если они являются соседями по горизонтали или вертикали. Для решения задачи можно использовать матрицу меток областей, назначая метке максимальное значение среди соседей.

Алгоритм пульсации применяется в итерационных приложениях, параллельных по данным. Данные разделяются между процессорами (разбивая матрицу на полосы или блоки). Работа каждого процессора строится по схеме (см. рис. 1).

инициализация локальных переменных; пока
не выполнено, выполнять передать значения соседям ; получить значения от соседей ; пересчитать локаль-
- ные переменные ;

Рис. 1: Алгоритм пульсации

Задание 7

Задача сглаживания изображения. Представить изображение матрицей чисел, определяющих цвет пикселя (1 — есть цвет, 0 — нет). Нужно убрать «пики» и «зазубрины», затемнить все светлые пиксели, у которых как минимум *d* соседей не освещены, то же сделать и для тёмных пикселей. Использовать алгоритм пульсации (см. задание 6). Рассмотреть задачу для разных изображений, разного количества процессоров, числа *d*.

Задание 8

Задача «эволюция». Дано двумерное поле клеток, каждая из которых либо содержит организм (1), либо пуста (0). Каждая клетка проверяет состояние своих соседей (их 8) и изменяет своё по правилам:

1. Живая клетка, вокруг которой < 2 живых клеток, умирает от одиночества.
2. Живая клетка, вокруг которой есть 2 или 3 живых клеток, выживает.
3. Живая клетка, вокруг которой > 3 живых клеток, умирает от перенаселения.
4. Пустая клетка, рядом с которой равно 3 живых соседа, оживает.

С помощью алгоритмы пульсации (см. задание 6) показать *n* шагов эволюции жизни. Константы «2» и «3» можно заменить своими значениями.

Задание 9

Задача «жизнь волков и зайцев». Дано двумерное поле клеток, каждая из которых либо содержит волка (1), либо зайца (2), либо пуста (0). Каждая клетка проверяет состояние своих соседей (их 8) и изменяет своё по правилам:

1. Волки и зайцы живут по правилам задачи 8, для зайцев число 3 можно заменить на 4 или 5.
2. Волк съедает одного зайца-соседа, в любом порядке на ваш выбор, за один шаг.
3. Порядок следования правил также выбирается вами (волк ли съест быстрее зайца, или заяц умрёт сам).
4. Использовать алгоритм пульсации (задание 6).

Задание 10

Задача «пожар». Дано двумерное поле клеток, каждая из которых либо содержит дерево (1), либо куст (2), либо пуста (0). Пожар начинается в одной клетке (либо выбирается случайным образом, либо задаётся явно). Теперь клетка может гореть (5),

или быть потушенной (0). Каждая клетка проверяет состояние своих соседей (их 4) и изменяет своё по правилам:

1. Куст или дерево загорается, если любой из соседей горит.
2. Куст горит 1 поколение, дерево 2 поколения.
3. Куст или дерево загорается, если его окружают 3 горящие клетки по диагонали.

С помощью алгоритма пульсации (задание 6), показать картину пожара поля, пока пожар не закончится.

Задание 11

Нахождение простых чисел < *n* решетом Эратосфена. Использовать конвейер процессов-фильтров, каждый получает поток чисел от соседа слева и отправляет поток соседу справа (последний — первому). Первое число, полученное фильтром, будет простым; отсылаем соседу числа, не кратные первому. Конец списка можно отслеживать специальным маркером, тогда процесс завершает работу.

Задание 12

Генерация простых чисел < *n* с помощью портфеля задач. Используется алгоритм «управляющий-рабочие». Управляющий передаёт рабочим процессам числа, кандидаты в простые (например, 5, 7, 9, ...). Рабочие процессы имеют начальный массив простых чисел (2, 3), получает кандидата на простоту, деля его на меньшие простые числа, полученное простые числа передаются управляющему, который и может рассылать их остальным процессам.

Задание 13

Решение СЛАУ методом исключений Гаусса. Сравнить производительность программы для систем разного размера и при разном количестве рабочих процессов.

Задание 14

Выполнить LU-разложение матрицы . Проверить корректность вычислений, т. е., что $LU = A$. Здесь L — нижняя треугольная матрица, U — верхняя треугольная матрица. Сравнить производительность программы для матриц разного размера и при разном количестве рабочих процессов.

Задание 15

Найти обратную матрицу. Проверить корректность вычислений. Сравнить производительность программы для матриц разного размера и при разном количестве рабочих процессов.

Задание 16

Гравитационная задача *n* тел. Дано большое число астрономических тел галактики (звезд, пылевых облаков, чёрных дыр). Каждое тело имеет массу, начальное положение (на плоскости, т. е. задача двухмерная) и скорость. Гравитация вызывает перемещение и ускорение тел. Показать эволюцию системы, т. е. движение тел за заданное число шагов изменения времени. Величина силы гравитации между двумя телами *i* и *j* равна

ГГЦ *и* т«

^{3} где— массы тел;

— расстояние между ними;

$G = 6{,}67 \cdot 10^{-11}$ —значение гравитационной постоянной.

Направление силы задается единичным вектором от *i* к *j* и противоположным вектором от *j* к *i.* Общая сила, действующая на тело, есть сумма сил воздействия всех остальных тел. Приведём формулы для вычисления скорости и положения тела: ускорение: *a = F/m,* изменение скорости: dv/ = *a* dt, изменение положения тела: *dp/ = v/* + dv/2^ dt на интервале времени dt.

При реализации задачи использовать модель «взаимодействующие равные», или «круговой конвейер», или алгоритм Барнса-Хата (Barnes-Hut), или быстрый метод мультиполей.

Задание 17

Промоделировать движение шаров на бильярдном столе. Задать начальные положения и скорости всех шаров, считая движение идеальным, без трения.

Задание 18

Решить уравнение Лапласа (эллиптическое дифференциальное уравнение в частных производных) методом Якоби. Известны значения в точках на границе, нужно аппроксимировать решение во внутренних точках области, покрыв область

равномерной сеткой точек (сеточные вычисления). В методе Якоби новое значение в каждой точке сетки равно среднему от предыдущих значений её соседних точек (4 точки). Вычисления выполняются по достижению заданной точности (разница между текущей и предыдущей итерацией). Использовать алгоритм пульсации (см. задание 6).

Задание 19

Решить уравнение Лапласа (эллиптическое дифференциальное уравнение в частных производных) при помощи метода последовательной сверхрелаксации по схеме «красное-чёрное». Известны значения в точках на границе, нужно аппроксимировать решение во внутренних точках области, покрыв область равномерной сеткой точек (сеточные вычисления). Вычисления выполняются по достижению заданной точности (разница между текущей и предыдущей итерацией). Использовать алгоритм пульсации (см. задание 6).

Задание 20

Найти кратчайшие пути между всеми парами вершин непустого взвешенного графа, если в нём нет циклов отрицательной суммарной длины, либо обнаружить наличие таких циклов. Для решения задачи использовать параллельную реализацию алгоритма Флойда.

Задание 21

Для взвешенного неориентированного графа найти его поддерево, соединяющее все вершины и обладающее минимальным суммарным весом рёбер (минимальное остовное дерево). Для решения задачи использовать параллельную реализацию алгоритма Прима.

Задание 22

Найти разбиение множества вершин заданного неориентированного графа на непересекающиеся подмножества с максимально близким количеством вершин в каждом из подмножеств и минимальным количеством рёбер, проходящих между полученными подмножествами. Для решения задачи использовать параллельную реализацию алгоритма Кернигана-Лина.

Библиографический список

1. Топорков В.В. Модели распределенных вычислений.ГЭлектронный ресурс] - Издательство: ФИЗМАТЛИТ, 2011. Режим доступа: http://www.knigafund.ru.
2. Intel Parallel Programming Professional (Introduction) [Электронный ресурс] - Национальный Открытый Университет «ИНТУИТ» • 2015 год • 569 с. Режим доступа: http://www.knigafund.ru.
3. Гергель В. П. Теория и практика параллельных вычислений: учебное пособие. [Электронный ресурс] - Интернет-Университет Информационных Технологий 2007 г. 424с. Режим доступа: http://www.knigafund.ru.
4. Архитектуры и топологии многопроцессорных вычислительных систем: курс лекций: учебное пособие. [Электронный ресурс] - Издательство: Интернет- Университет Информационных Технологий 2004 г. Режим доступа: http://www.knigafund.ru/books/178882.
5. Назаров С.В., Широков А.И. Современные операционные системы. ГЭлектронный ресурс] - Интернет-Университет Информационных Технологий 2011 г. 280 с. Режим доступа: http://www. knigafund.ru.
6. Антонов А. С. Параллельное программирование с использованием технологии MPI: курс [Электронный ресурс] - Интернет-Университет Информационных Технологий 2008 г. 71 с Режим доступа: http://www.knigafund.ru.
7. Макогоненко Г. И. Параллельное программирование: метод. пособие к лаб. работам по дисц. "Параллельные и распределенн. вычисления" / Макогоненко Г. И.;ЮРГТУ(НПИ). - Новочеркасск: Изд-во ЮРГТУ(НПИ), 2010. -107 с.:95-00

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

Ш КИЙ Г УД ЫЙ У И И

ФАКУЛЬТЕТ МАТЕТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА “АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ и ЦИФРОВЫХ ТЕХНОЛОГИЙ”

УТВЕРЖДЕНО
на заседании кафедры АСЦТ «3»
сен6тября 2020 г. протокол №1
Заведующий кафедрой
Молдояров У.Д.

Фонд заданий предназначен для контроля знаний магистрантов на экзамене направления 710100 «Информатика и вычислительная техника» по дисциплине
«Параллельные вычисления»
на 2020-2021 учебный год

Курс - 1 курс, Семестр – 1 семестр
Всего часов 120, из них часов аудиторных 48,
72 часов самостоятельных работ

Лекционные занятия	24
Лабораторные занятия	24
Самостоятельные работы	72
Экзамен	1 семестр

Составитель: Атырова Р. _____

Ош. 2020

Фонд оценочных средств

1. Описание назначения и состава фонда оценочных средств

Настоящий фонд оценочных средств (ФОС) входит в состав рабочей программы дисциплины «Параллельные вычисления» и предназначен для оценки планируемых результатов обучения по дисциплине

Оценочные материалы представлены для проведения текущего контроля успеваемости и промежуточной аттестации обучающихся в форме: экзамена.

Для оценки планируемых результатов обучения используются следующие оценочные материалы:

- вопросы для защиты лабораторных работ (текущий контроль);
- вопросы к экзамену (итоговый контроль).

4.1. Показатели и критерии оценивания лабораторных работ и ответы на контрольные вопросы

Оценка	Показатели оценивания	Критерии оценивания
«5» (отлично, зачтено)	Качество выполнения всех заданий лабораторной работы; оформление, и структура; самостоятельность выполнения, выполнение и сдача лабораторной работы в установленные сроки.	Выполнены все задания лабораторной работы; работа выполнена в срок; оформление, структура и стиль работы образцовые; лабораторная работа выполнена самостоятельно.
«4» (хорошо, зачтено):		Выполнены все задания лабораторной работы с незначительными замечаниями; работа выполнена в срок; в оформлении, структуре и стиле работы нет грубых ошибок; работа выполнена самостоятельно.
«3» (удовлетворительно, зачтено)		Выполненные задания лабораторной работы имеют значительные недочеты, устраненные после проверки преподавателем; работа выполнена с нарушениями графика; имеются недостатки по оформлению структуре и стилю работы; лабораторной работа выполнена самостоятельно.
«2» (неудовлетворительно, не зачтено)		задания в лабораторной работе решены не полностью или решены неверно; лабораторной работа выполнена не самостоятельно

4.2. Показатели и критерии оценивания контрольной работы

Оценка	Показатели оценивания	Критерии оценивания
«5» (отлично, зачтено)	Качество выполнения всех разделов контрольной работ; полнота раскрытия темы, правильность формулировок; оформление, структура и стиль контрольной работы; выполнение и сдача контрольной работы в	Полное раскрытие темы;указание точных названий и определений;правильная формулировка понятий и категорий;приведение формул и соответствующей статистики и др.
«4» (хорошо, зачтено):		Недостаточно полное, по мнению преподавателя, раскрытие темы;несущественные ошибки в определении понятий и категорий, формулах, статистических данных и т.п., кардинально не меняющих суть изложения;наличие грамматических и стилистических ошибок и др.

«3» (удовлетвори-тельно, зачтено)	установленные сроки.	Отражение лишь общего направления изложения лекционного материала;наличие достаточного количества несущественных или одной-двух существенных ошибок в определении понятий и категорий, формулах, статистических данных и т.п.;наличие грамматических и стилистических ошибок и др.
«2» (неудовлетвори-тельно, не зачтено)		Нераскрытые темы;большое количество существенных ошибок;наличие грамматических и стилистических ошибок и др.

1. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Для проверки качества освоения программы дисциплины и оценки результатов обучения по дисциплине, соотнесенных с установленными в программе индикаторами достижения компетенции проводится текущий контроль успеваемости и промежуточная аттестация обучающихся в форме зачета.

Контроль успеваемости обучающихся осуществляется с использованием модульнорейтинговой системы. Рейтинг-план по дисциплине «Параллельное программирование» включен в состав ЭУМКД [3].

Текущий контроль проводится регулярно на всех видах групповых занятий по дисциплине. В конце семестра на основании поэтапного контроля процесса обучения суммируются баллы текущих, рубежных рейтингов (контрольные недели), подсчитываются дополнительные баллы (за посещаемость и активность на занятиях).

Результаты рейтинговой аттестации объявляются преподавателем на последнем занятии в зачетную неделю и служат основой для итогового результата промежуточной аттестации обучающегося по дисциплине.

5.1. Соответствие балльной шкалы оценок по дисциплине уровню сформированности компетенций обучающегося

Уровень сформированности компетенций	Оценка	Пояснение
Высокий	«5» (отлично) зачтено	Теоретическое содержание курса освоено полностью, без пробелов, все предусмотренные программой обучения учебные задания выполнены, планируемые результаты обучения по дисциплине, соотнесенные с установленными в программе индикаторами достижения компетенций, достигнуты.
Средний	«4» (хорошо) зачтено	Теоретическое содержание курса освоено полностью, все предусмотренные программой обучения учебные задания выполнены с незначительными замечаниями, планируемые результаты обучения по дисциплине, соотнесенные с установленными в программе индикаторами достижения компетенций, достигнуты.
Удовлетворительный	«3» (удовлетворительно) зачтено	Теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, большинство предусмотренных программой обучения учебных задач выполнено, но отмечены ошибки, планируемые результаты обучения по дисциплине, соотнесенные с установленными в программе индикаторами достижения компетенций, в целом достигнуты.

Неудовлетворительный	«2» (не удовлетворительно) не зачтено	Теоретическое содержание курса не освоено, большинство предусмотренных программой обучения учебных заданий либо не выполнено, либо содержит грубые ошибки; дополнительная самостоятельная работа над материалом не приведет к какому-либо значимому повышению качества выполнения учебных заданий. Планируемые результаты обучения по дисциплине, соотнесенные с установленными в программе индикаторами достижения компетенций, не достигнуты.

Оценочные средства для текущего контроля успеваемости и промежуточной аттестации 2.1Фонд оценочных средств для проведения текущего контроля.

1. Задачи

В ходе выполнения лабораторных работ студентам предлагается с помощью пяти технологий параллельного программирования выполнить четыре задания: умножение вектора на число, скалярное произведение векторов, умножение матрицы на число, произведение матриц. Технологии: потоки Windows, потоки Pthreads, потоки Java, OpenMP, MPI. Результаты времени выполнения на одном и том же объеме вычислений необходимо замерять на одном, двух и четырех потоках, записать в таблицу и проанализировать.

1. Проекты.

Студентам предлагается выполнить проекты в командах 2-3 человека, подготовить и защитить доклад по теме проекта, подготовить тезисы для участия в студенческой конференции ФКТиПМ или статью для публикации в журнале, опубликовать исходный код на GitHub.

Темы проектов:

1. MMX/SSE

С помощью векторных инструкций процессора MMX или SSE выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

2. OpenCL

С помощью OpenCL выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

3. Nvidia Cuda

С помощью Nvidia Cuda выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

4. WebCL

С помощью WebCL выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

5. Web Workers

С помощью Web Workers выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;

- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

6. Cilk

С помощью Cilk выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

7. TBB

С помощью TBB выполнить одну из операций:

- скалярное произведение векторов;
- умножение матриц;
- нахождение минимума;

Протестировать на большом объеме данных и сравнить с однопоточной реализацией.

8*. Написать программу, переносящую запущенный процесс с одной виртуальной машины под управлением ОС Linux на другую.

9*. Реализовать пример параллельной обработки данных на платформе Android.

10*. Реализовать пример параллельной обработки данных средствами Trik DSP или STM32F4 DSP.

11*. Реализовать пользовательские потоки для ОС Linux или Windows.

12*. Реализовать многопоточность на микроконтроллере STM32 (ядро Arm Cortex).

13*. Задача в каталоге 1.NStars - поиск оптимального маршрута и параметров миссии по исследованию звезд и поиску внеземной жизни.

14*. Реализовать программу поиска оптимальной разрезки заданного прямоугольника (лента) или множества заданных прямоугольников (листы) на множество заданных выпуклых многоугольников. При разрезке все заданные выпуклые многоугольники не должны пересекаться. Более оптимальной считается та разрезка, при которой высота использованного прямоугольника меньше (лента) или количество использованных прямоугольников меньше (листы).

15*. Задача в каталоге 2.MedClassify - классификация медицинских услуг.

16*. Реализовать параллельное обучение и распознавание изображений сверточными нейронными сетями по базе ImageNet.

17*. Реализовать прозрачно параллельное вычисление findall, label или истинности предиката в SWI Prolog с использованием потоков на CPU или GPU.

18*. Реализовать библиотеку для прозрачных параллельных вычислений на GPU в Clojure.

19*. Написать приложение, определяющее название компании или имя физ. лица по почтовому адресу.

20*. Написать веб-сервис, определяющий пол владельца аккаунта в инстаграмме.

21*. Написать приложение для автоматического управления кампаниями в Яндекс Директ для интернет-магазинов.

22*. Реализовать обучение и распознавание голосовых команд для мобильных роботов.

23*. Написать приложение для поиска схожих по параметрам аккаунтов в соц. сетях с заданным множеством аккаунтов.

24*. Реализовать локализацию и определение размеров ближайшего объекта по изображению с двух камер и данным датчиков расстояния в реальном времени.

25*. Реализовать чтение дорожной разметки в реальном времени для мобильного робота.

26*. Реализовать машинное обучение оптимальной походки на местности для многоагентной системы хексаподов.

27*. Реализовать машинное обучение оптимальных параметров езды по линии для многоагентной системы колесных роботов.

28*. Реализовать систему распознавания жестов по данным от перчатки с датчиками изгиба на пальцах.

29*. Реализовать систему распознавания жестов по данным от датчиков смартфона.

30*. Реализовать систему составления оптимальных расписаний для факультета.

31*. Реализовать подсистему планирования отпусков для CRM.
32*. Реализовать систему планирования заказов поставщикам для МойСклад, Класс365 или 1С: Предприятие.
33*. Реализовать сервер для упрощенной версии протокола ModBus по TCP/IP (бинарный и ASCII вариант), UDP, веб-сокеты и клиентов на C/C++, Java, Android и JavaScript для одновременной записи и чтения данных по разным протоколам.

4.2 Фонд оценочных средств для проведения промежуточной аттестации.

Вопросы для промежуточной аттестации по итогам освоения дисциплины:

1. Уровни распараллеливания вычислений: независимость узлов, конвейеризация, параллельные и распределенные вычисления.
Определение понятий: параллельные вычисления, суперкомпьютер, кластер.
2. Необходимость высокопроизводительных вычислений. Причины ограничения роста тактовой частоты процессоров. Классификация Флинна (SISD, SIMD, MISD, MIMD), понятие мультипроцессоров и мультикомпьютеров.
3. CISC- и RISC-процессоры. Основные черты RISC-архитектуры.
4. Повышение производительности процессоров за счет конвейеризации. Условия оптимального функционирования конвейера.
5. Суперконвейерные и суперскалярные процессоры. Выделение независимо работающих устройств: IU, FPU, MMU, BPU.
6. Повышение производительности процессоров за счет введения кэш-памяти. Кэши: единый, Гарвардский, с прямой записью, с обратной записью.
7. Согласование кэшей в мультипроцессорных системах с общей памятью.
8. Программа, процессор, процесс. Основные составляющие процесса, состояния процесса.
9. Виды ресурсов: аппаратные, программные, активные, пассивные, локальные, разделяемые, постоянные, временные, некритичные, критичные.
10. Типы взаимодействия процессов: сотрудничающие и конкурирующие процессы. Критические секции, взаимное исключение процессов (задач).
Проблемы, возникающие при синхронизации задач.
11. Механизмы взаимодействия процессов. Разделяемая память.
12. Механизмы взаимодействия процессов. Семафоры.
13. Механизмы взаимодействия процессов. Сигналы, события.
14. Потоки (нити, threads). Сравнение с процессами. Ресурсы, приоритеты.
15. Алгоритм Деккера.
16. Создание, уничтожение и ожидание завершения работы потоков Windows.
17. Синхронизация потоков. Критические секции Windows.
18. Создание, уничтожение и ожидание завершения работы потоков POSIX Threads. Барьерыв POSIX Threads.
19. Синхронизация потоков. Мьютексы: виды и операции над ними. Мьютексы в POSIX Threads.
20. Система MPI. Понятие параллельной программы MPI. Общая характеристика. Создание и завершение MPI-программы. Получение ранга процесса и количества процессов. Структура MPI-программы.
Пример.
21. Точечные обмены данными между процессами MPI-программы. Базовые типы MPI. Пример. Одновременные прием и передача в MPI.
Буферизованый, блокирующий, неблокирующий методы передачи данных.
22. Коллективные взаимодействия процессов MPI. Передача данных от одного процесса всем процессам. Сбор данных на одном процессе со всех процессов.
23. Коллективные взаимодействия процессов MPI. Обобщенная передача данных от одного процесса всем процессам. Обобщенный сбор данных на одном процессе со всех процессов.

24. Коллективные взаимодействия процессов MPI. Синхронизация вычислений. Аварийное завершение.
25. Создание групп в MPI и управление ими.
26. Управлениекоммуникаторами в MPI. Создание и уничтожение коммуникатора.
27. Производные типы в MPI. Карта типа. Характеристики типа.
28. Производные типы в MPI. Непрерывный способ создания. Регистрация и освобождение типа.
29. Производные типы в MPI. Векторный способ создания. Регистрация и освобождение типа.
30. Производные типы в MPI. Индексный способ создания. Регистрация и освобождение типа.
31. Производные типы в MPI. Структурный способ создания. Регистрация и освобождение типа.
32. Виртуальные топологии в MPI: декартова топология.
33. Виртуальные топологии в MPI: топология графа.
34. Технология OpenMP: общая характеристика, модель программирования, модель данных, последовательные и параллельные области, замер времени.
35. Технология OpenMP: мастер-поток, получение ранга, количества потоков, количества процессоров, задание количества порождаемых потоков, директивы master и single
36. Технология OpenMP: параллельные циклы, сокращенная запись, редукция.
37. Технология OpenMP: синхронизация потоков, барьеры, критические секции, атомарные операции, замки.
38. Характеристики типовых схем коммуникации в многопроцессорных вычислительных системах. Примеры топологий сети передачи данных. Топология сети вычислительных кластеров. Характеристики топологий. 39.Оценка трудоемкости основных операций передачи данных. Передача данных между двумя процессорами сети. Передача данных от одного процессора всем процессорам сети.
40.Оценка трудоемкости основных операций передачи данных. Передача данных от всех процессоров всем процессорам сети.
41.Оценка трудоемкости операций передачи данных для кластерных систем. Модель Хокни.
42. Граф«операции и операнды» и возможности распараллеливания вычислений, минимально возможное время выполнения параллельного алгоритма. Пример.
43. Потоки и примитивы синхронизации Java.
44. Параллельное программирование и транзакционная память в Clojure.
Критерий оценивания:

Экзамен по дисциплине преследует цель оценить работу студента за курс, получение теоретических знаний, их прочность, развитие творческого мышления, приобретение навыков самостоятельной работы, умение применять полученные знания для решения практических задач.

Форма проведения экзамена: письменно, устно.

Экзаменатору предоставляется право задавать студентам дополнительные вопросы по всей учебной программе дисциплины.

Результат сдачи экзамена заноситься преподавателем в экзаменационную ведомость и зачетную книжку.

Оценивание уровня освоения дисциплины основывается на качестве выполнения студентом задач за семестр, индивидуального задания / проекта и ответов на вопросы экзамена. За ответы на вопросы экзамена дается до 20 баллов.

0 баллов: непонимание сущности излагаемых вопросов, грубые ошибки в ответе, неуверенные и неточные ответы на дополнительные вопросы экзаменаторов.

0-10 баллов: знание и понимание основных вопросов программы, студент указал

направление решения индивидуальной задачи; частично ответил на два вопроса билета или достаточно полно ответил хотя бы на один вопрос.

11-15 баллов: твёрдые и достаточно полные знания всего программного материала, последовательные, правильные, конкретные ответы на поставленные вопросы при свободном реагировании на замечания по отдельным вопросам; достаточно полно ответил на два вопроса.

16-20 баллов: глубокие исчерпывающие знания всего программного материала, логически последовательные, полные, грамматически правильные и конкретные ответы на вопросы экзаменационного билета и дополнительные вопросы.

Критерии оценки:

Оценка		
Удовлетворительно/зачтено	Хорошо/зачтено	Отлично/зачтено
от 70 до 79 баллов	от 80 до 89 баллов	90 или более баллов

Оценочные средства для инвалидов и лиц с ограниченными возможностями здоровья выбираются с учетом их индивидуальных психофизических особенностей.

- при необходимости инвалидам и лицам с ограниченными возможностями здоровья предоставляется дополнительное время для подготовки ответа на экзамене;
- при проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья предусматривается использование технических средств, необходимых им в связи с их индивидуальными особенностями;
- при необходимости для обучающихся с ограниченными возможностями здоровья и инвалидов процедура оценивания результатов обучения по дисциплине может проводиться в несколько этапов.

Процедура оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья по дисциплине предусматривает предоставление информации в формах, адаптированных к ограничениям их здоровья и восприятия информации:

Для лиц с нарушениями зрения:

- в печатной форме увеличенным шрифтом,
- в форме электронного документа.

Для лиц с нарушениями слуха:

- в печатной форме,
- в форме электронного документа.

Для лиц с нарушениями опорно-двигательного аппарата:

- в печатной форме,
- в форме электронного документа.

Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.