МИНОБРНАУКИ РОССИИ

Орский гуманитарно-технологический институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Оренбургский государственный университет»
(Орский гуманитарно-технологический институт (филиал) ОГУ)

Кафедра программного обеспечения

Методические указания
для обучающихся по освоению дисциплины

«Б.1.В.ДВ.2.1 Параллельное программирование»

Уровень высшего образования

БАКАЛАВРИАТ

Направление подготовки
09.03.01 информатика и вычислительная техника
(код и наименование направления подготовки)

Программное обеспечение средств вычислительной техники и автоматизированных систем
(наименование направленности (профиля) образовательной программы)

Тип образовательной программы
Программа академического бакалавриата

Квалификация
Бакалавр

Форма обучения
Очная

Год начала реализации программы (набора)
2014, 2015, 2016, 2017

г. Орск 2017

Методические указания для обучающихся по освоению дисциплины «Б.1.В.ДВ.2.1 Параллельное программирование» предназначены для обучающихся очной формы обучения направления подготовки 09.03.01 информатика и вычислительная техника, профиля «Программное обеспечение средств вычислительной техники и автоматизированных систем»

Составитель _____ В.Н. Муллабаев

Методические указания рассмотрены и одобрены на заседании кафедры программного обеспечения, протокол № 9 от «07» июня 2017 г.

Заведующий кафедрой программного обеспечения _____ Е.Е.Сурина

1 Методические указания по проведению лекционных занятий

Лекционные занятия в высшем учебном заведении являются основной формой организации учебного процесса и должны быть нацелены на выполнение ряда задач:

- ознакомить студентов со структурой дисциплины;
- изложить основной материал программы курса дисциплины;
- ознакомить с новейшими подходами и проблематикой в данной области;
- сформировать у студентов потребность к самостоятельной работе с учебной, нормативной и научной литературой.

Лекционное занятие представляет собой систематическое, последовательное, монологическое изложение преподавателем-лектором учебного материала, как правило, теоретического характера.

Цель лекции – организация целенаправленной познавательной деятельности студентов по овладению программным материалом учебной дисциплины.

Чтение курса лекций позволяет дать связанное, последовательное изложение материала в соответствии с новейшими данными науки, сообщить слушателям основное содержание предмета в целостном, систематизированном виде.

В ряде случаев лекция выполняет функцию основного источника информации, когда новые научные данные по той или иной теме не нашли отражения в учебниках.

Организационно-методической базой проведения лекционных занятий является рабочий учебный план направления подготовки. При подготовке лекционного материала преподаватель обязан руководствоваться учебными программами по дисциплинам кафедры, тематика и содержание лекционных занятий которых представлена в рабочих программах, учебно-методических комплексах.

При чтении лекций преподаватель имеет право самостоятельно выбирать формы и методы изложения материала, использовать различные технические средства обучения.

Рекомендации по работе студентов с конспектом лекций.

Изучение дисциплины студенту следует начинать с проработки рабочей программы, особое внимание, уделяя целям и задачам, структуре и содержанию курса.

При конспектировании лекций студентам необходимо излагать услышанный материал кратко, своими словами, обращая внимание, на логику изложения материла, аргументацию и приводимые примеры. Необходимо выделять важные места в своих записях. Если непонятны какие-либо моменты, необходимо записывать свои вопросы, постараться найти ответ на них самостоятельно. Если самостоятельно не удалось разобраться в материале, впоследствии необходимо либо на следующей лекции, либо на лабораторном занятии или консультации обратиться к ведущему преподавателю за разъяснениями.

Успешное освоение курса предполагает активное, творческое участие студента путем планомерной, повседневной работы. Лекционный материал следует просматривать в тот же день. Рекомендуемую дополнительную литературу следует прорабатывать после изучения данной темы по учебнику и материалам лекции.

Каждая тема имеет свои специфические термины и определения. Усвоение материала необходимо начинать с усвоения этих понятий. Если какое-либо понятие вызывает затруднения, необходимо посмотреть его суть и содержание в словаре (Интернете), выписать его значение в тетрадь для подготовки к занятиям.

При подготовке материала необходимо обращать внимание на точность определений, последовательность изучения материала, аргументацию, собственные примеры, анализ конкретных ситуаций. Каждую неделю рекомендуется отводить время для повторения пройденного материала, проверяя свои знания, умения и навыки по контрольным вопросам и тестам.

2 Методические указания по лабораторным и практическим работам

Изучение дисциплины «Б.1.В.ДВ.2.1 Параллельное программирование» предполагает посещение обучающимися не только лекций, но и лабораторных работ. Лабораторные работы со студентами предназначены для проверки усвоения ими теоретического материала дисциплины.

Основные цели лабораторных работ:

- закрепить основные положения дисциплины;
- проверить уровень усвоения и понимания студентами вопросов, рассмотренных на лекциях и самостоятельно изученных по учебной литературе;
- научить пользоваться нормативной и справочной литературой для получения необходимой информации о конкретных технологиях;
- оказать помощь в приобретении навыков расчета точностных характеристик;
- восполнить пробелы в пройденной теоретической части курса и оказать помощь в его усвоении.

Для контроля знаний, полученных в процессе освоения дисциплины на лабораторных занятиях обучающиеся выполняют задания реконструктивного уровня и комплексное практическое задание.

Целью выполнения задания реконструктивного уровня и комплексного задания студентами является систематизация, закрепление и расширение теоретических знаний, полученных в ходе изучения дисциплины.

Ниже приводятся общие методические указания, которые относятся к занятиям по всем темам:

- в начале каждого лабораторного занятия необходимо сформулировать цель, поставить задачи;
- далее необходимо проверить знания студентами лекционного материала по теме занятий;
- в процессе занятия необходимо добиваться индивидуальной самостоятельной работы студентов;
- знания студентов периодически контролируются путем проведения текущей аттестации (рубежного контроля), сведения о результатах которой доводятся до студентов и подаются в деканат;
- время, выделенное на отдельные этапы занятий, указанное в рабочей программе, является ориентировочным; преподаватель может перераспределить его, но должна быть обеспечена проработка в полном объеме приведенного в рабочей программе материала;
- на первом занятии преподаватель должен ознакомить студентов с правилами поведения в лаборатории и провести инструктаж по охране труда и по пожарной безопасности на рабочем месте;
- преподаватель должен ознакомить студентов со всем объемом лабораторных работ и требованиями, изложенными выше;
- преподаватель уделяет внимание оценке активности работы студентов на занятиях, определению уровня их знаний на каждом занятии.

На лабораторных работах решаются задачи из всех разделов изучаемой дисциплины.

2. Практическая часть.
Введение

Лабораторные работы по дисциплине ««Б.1.В.ДВ.2.1 Параллельное программирование» включают в себя изучение двух технологии параллельного программирования. Первая технология – PVM (Parallel Virtual Machine), Параллельная Виртуальная Машина. Вторая технология –MPI (Message Passing Interface), Интерфейс Передачи Сообщений.

2.1. PVM (Parallel Virtual Machine). Краткие теоретические сведения

PVM представляет унифицированную среду, внутри которой разрабатывается параллельная программа, которая позволяет прозрачно выполнять маршрутизацию всех сообщений, преобразование данных и распределение задач по сети различными компьютерами. Задачи имеют доступ к ресурсам PVM через библиотеку стандартных программ. Они позволяют запускать, останавливать задачи, обеспечивают связь и синхронизацию между задачами. PVM задача обрабатывает либо управляющую, либо зависимую структуру. Т.е. в любой момент выполнения параллельных вычислений, любая задача может запустить либо остановить другие задачи; добавить либо удалить компьютеры в PVM. Любой процесс может связываться и синхронизироваться с любым другим. Все управляющие и зависимые структуры разрабатываются в PVM системе с использованием PVM конструкции и соответствующего языка программирования (C или Fortran).

PVM – это ПО, которое позволяет пользователю рассматривать множество ЭВМ с различной архитектурой, объединенной сетью как один параллельный компьютер.

Основные компоненты PVM машины:

1) Процесс DVM – демон.
2) Библиотеки PVM подпрограмм (lib PVM3.a, lib PVM3.lib, lib f PVM3.a, lib f PVM3.exe, lib g PVM3.exe, lib g PVM3.a)

DAEMON :

- PVMD3 – это процесс, который управляет пользовательскими процессами в рамках PVM приложения и координирует межмашинные PVM коммуникаторы.
- Пользователь, создающий PVM машину должен запустить Demon PVMD3 на каждом компьютере, включённом в состав его параллельной машины. Если несколько пользователей включают один и тот же компьютер в состав своей PVM машины, то на этом компьютере могут быть запущены несколько Demon PVMD3.
- Каждый Demon PVMD3 поддерживает свою таблицу конфигураций и обрабатывает информацию, относящуюся к параллельной машине в состав которой он входит.
- Первый запущенный Demon PVMD3 параллельной машины является ведущим. Он запускает остальных «демонов» (подключает компьютер в состав параллельной машины), которые являются рабочими «демонами». Обычно ведущий «демон» запускается на компьютере, куда залогинился пользователь, создающий PVM.
- Пользовательские программы обмениваются данными друг с другом через демонов, т.е.: а) каждая задача общается со своим локальным «демоном» PVMD3 через библиотечные вызовы lib.PVM.

б) Локальный Demon PVMD3 передаёт/получает сообщения к/от Demon PVM D3 удалённых хостов.

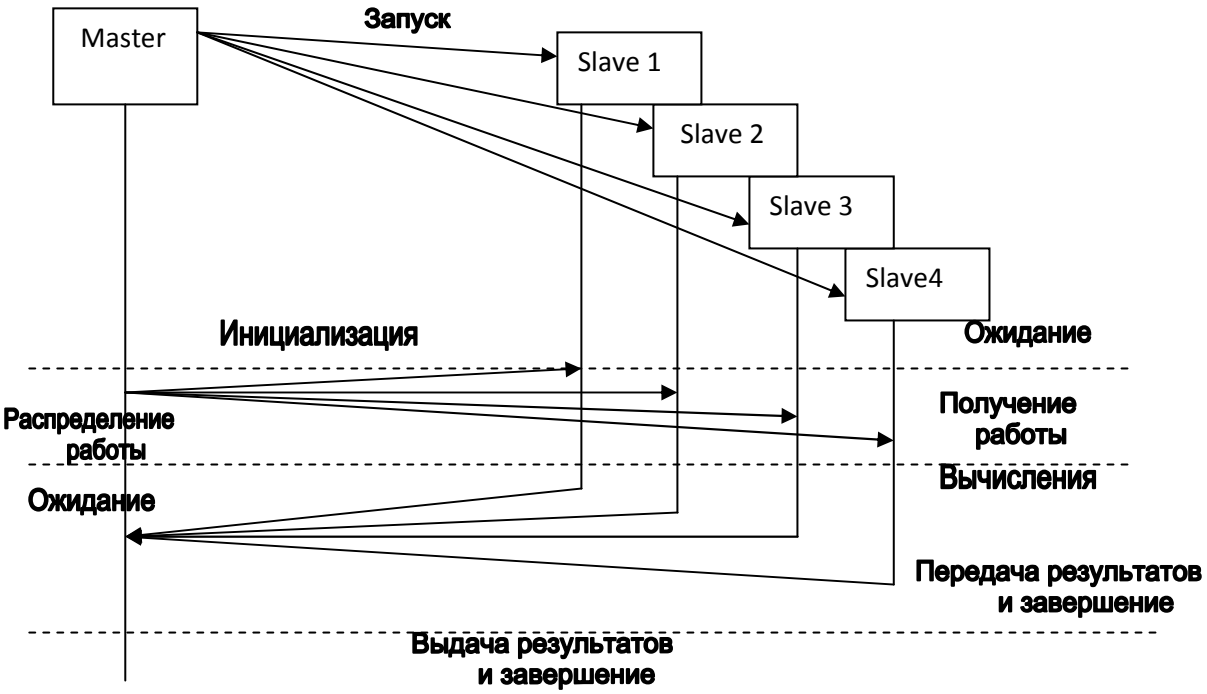
- На каждом компьютере, входящем в состав параллельной машины должен быть установлен тот демон pvmd3, версия которого относится к архитектуре конкретного компьютера.

PVM широко распространена и имеются версии практически для любых компьютеров, включая суперкомпьютеры Cray и SGI.

Компоненты PVM библиотеки.

1. Существует три библиотеки PVM:
   - libpvm3.a – библиотека подпрограммы на языке С, требуется всегда.
   - libfpvm3.a - требуется для выполнения программ на языке Fortran.
   - libgpvm3.a – требуется для работы с динамическими группами.
2. Содержит подпрограммы, которые программист вставляет в свой параллельный код. Они дают возможность:
   - запускать и восстанавливать задачи;
   - проводить упаковку, прием/передачу и широкое вещание сообщений;
   - синхронизироваться с другими задачами через механизм барьеров.
   - просматривать и динамически изменять конфигурацию параллельной машины.
3. Запущенные задачи не обмениваются данными непосредственно друг с другом Вместо этого, они, используя библиотеку подпрограмм, передают соответствующие команды локальным «демонам» и в ответ получают статус выполнения команды.
4. Для упаковки обмениваемыми данными по умолчанию используют формат XDR, который понимает компьютеры с различной архитектурой, входящей в состав PVM.

Схема выполнения параллельного приложения:



Пишется ведущая программа (Master) и одна или несколько ведомых программ (Slave) на С, С++ или Fortran с использованием библиотечных функций PVM. Программа Master запускается программистом напрямую из командной строки и становится Master-task. Программа Slave запускается программой Master с помощью PVM вызовов и становится Slave – task. Slave задачи в свою очередь могут запускать другие Slave задачи, для которых, сами будут являться Master. Каждая задача имеет уникальный

идентификатор tid, который известен ей и родителю (Master). Каждая задача знает tid родителя.

Лабораторная работа №1

2.2. Написание и отладка первой ознакомительной программы.

Пример: Необходимо рассчитать сумму кубов $Y=\sum_{i=1}^{n} X3$

Псевдоалгоритм:
Задачи Master:

1) Master должен запустить n задач.
2) Должен раздать всем задачам задания.
   for i = 1 to n Send i to i.th task
3) Получить от каждой задачи выполненную работу и подсчитать общую сумму:
   for i = 1 to n { receive X from i.th task Sum :=sum +X }

Задачи Slave:

1. Получить задание от родителя receive i from parent
2. Рассчитать куб x=i*i*i
3. Отправить ответ родителю send x to parent

Программный код для первой ознакомительной задачи:

1. Программный код для Master:

```
#include<pvm3.h>
#define mastertag 1
#define slavetag 2
#define slave_exe "slave1"

main()
{
int nt,i,x,sum,N,*tid;
printf("vvedite znach N \n");
scanf("%d",&N);
tid=malloc(N+sizeof(nt));
nt=pvm_spawn(slave_exe,(char**) 0,1,"2-07",N,tid);
if (nt==N)
{for (i=0;i<N;i++)
{
pvm_initsend (PvmDataDefault);
pvm_pkint (&i,1,1);
pvm_send (tid[i],mastertag);

}
sum=0;
for (i=0;i<N;i++)
{ pvm_recv (tid[i],slavetag);
pvm_upkint(&x,1,1);
sum=sum+x;
```

```
}
printf ("Сумма кубов равна:%d\n",sum);
}
else
printf("Не удалось запустить все задачи!\n");
pvm_exit();
return 0;
}
```

2. Программный код для Slave:

```
#include <pvm3.h>
#define mastertag 1
#define slavetag 2
main()
{
int ptid,i,x;
ptid=pvm_parent();
pvm_recv(ptid,mastertag);
pvm_upkint(&i,1,1);
x=i*i*i;
pvm_initsend(PvmDataDefault);
pvm_pkint(&x,1,1);
pvm_send(ptid,slavetag);
pvm_exit();
return 0;
}
```

Объяснение синтаксиса команд:

Операция запуска Slave задач:

- int numt = pvm_spawn (char *task, char **argv, int flag, char *where, int ntask, int tids)- операция запуска Slave задач.

Параметры:

task – имя исполнимого файла.

argv – аргументы исполняемого файла, если нет аргумента, то нул.

flag - pvm task Default 0

pvm task Host 1 – в этом случае параметр Where указывает конкретный Host

pvm task Arch 2 – параметр Where указывает на архитектуру машины.

Если flag = 0, то параметр Where игнорируется.

ntask – количество пропускаемых исполнимых файлов.

tids – массив целых размеров ntask. В этот массив помещаются идентификаторы tids задач, запущенных вызовом pvm_spawn.

numt – количество удачно запущенных задач. Если значение <0, то системная ошибка.

Если numt< ntask, то запустились не все задачи.

Операции передачи сообщения - выполняется в 3 этапа:

1) Инициализация передачи (Указание формата данных): int info = pvm_ init send (int encoding), где encoding – схема кодирования следующего передаваемого сообщения.

pvm Data Defoult 0:XDR (код).
pvm Data Raw 1: без кодирования.
pvm Data In Place 2: данные остаются на месте.

2) Упаковка сообщений:

int info = pvm_pkint (int *ip, int nitem, int stride), где
ip – массив пакуемых данных (адрес – указатель первого значения для упаковки);
nitem – общее количество данных для упаковки.
stride – параметр упаковки. Первые – пакуются все данные. Вторые – каждые вторые данные. Третьи – все третьи данные и т.д.
info – код завершения, info<0 – ошибка!

3) Передача сообщений:

int info = pvm send (int tid,int msg tag), где
tid – идентификатор задачи процесса назначения.
msg tag – tag сообщения (неотрицательно число).
Задача может иметь разные сообщения, которые помечаются tag-ом.
info – код завершения, info<0 – ошибка!

Операции получения сообщения - выполняется в 2 этапа:

1) Получение сообщений:

int info = pvm_recv (int tid, int msg teg) – получение сообщений, где
tid – IP задача передающего процесса;
msg teg – teg сообщения (>=0);
info – код завершения, info<0 – ошибка!

2) Распаковка полученного сообщения:

int info = pvm_upkint (int *ip, int nitem, int stride) – распаковка сообщений, где
ip – массив данных для распаковки;
nitem – общее количество данных для распаковки;
stride – указывает какие данные распаковывать;
info – распаковывает код, info<0 – ошибка!

Лабораторная работа №2

Включает в себя две программы подобные предыдущей для закрепления усвоения материала.

А) Рассчитать способом параллельного программирования следующее выражение:

$$y = 1 + \frac{1}{tg(\frac{x0}{\sqrt{1+x0}})}$$

, где x меняет свое значение в диапазоне от x0 до xn с шагом h. Тип данных – вещественное число типа float.

Программный код для Master:

```
#include <pvm3.h>
#include <math.h>
#define mastertag 1
#define slavetag 2
```

```
#define slave_exe "slave2"
#define N 10
main ()
{
float x0,xn,h,y;
int nt,*tid,i,n;
printf("Введите x - начальное, x - конечное, h - шаг \n");
scanf("%f",&x0);
scanf("%f",&xn);
scanf("%f",&h);
n=floor((xn-x0)/h);
tid=malloc(n*sizeof(int));
nt=pvm_spawn(slave_exe,(char**)0,0," ",n,tid);
if(nt==n)
{
pvm_initsend(PvmDataDefault);
pvm_pkfloat(&x0,1,1);
pvm_send(tid[0],mastertag);

for(i=1;i<N;i++)
  {
x0=x0+h;
pvm_initsend(PvmDataDefault);
pvm_pkfloat(&x0,1,1);
pvm_send(tid[i],mastertag);
}

for(i=0;i<N;i++)
{pvm_recv(tid[i],slavetag);
pvm_upkfloat(&y,1,1);
printf("\n 4%d",i,"=%f",y);
}

}
else printf("ne vse zadanija zapysheni\n");
pvm_exit();
return(0);
}
```

Программный код для Slave:

```
#include <pvm3.h>
#define mastertag 1
#define slavetag 2
main()
{
float x0,y;
int ptid;
ptid=pvm_parent();
pvm_recv(ptid,mastertag);
pvm_upkfloat(&x0,1,1);
y=1+1/(tan(x0/sqrt(1+x0)));
```

```
pvm_pkfloat(&y,1,1);
pvm_send(ptid,slavetag);
pvm_exit();
return(0);
}
```

Б) Используя приемы параллельного программирования подсчитать сумму элементов массива, где большой массив делится на части. Каждая ведомая задача получает от ведущего свою порцию массива для подсчета суммы. Ведущая задача подсчитывает окончательную сумму. <u>В отличие от предыдущих примеров здесь показана как передается целый массив данных</u>:

Программный код для Master:

```
#include <pvm3.h>
#include <stdio.h>
#define size 1000
#define NPROCS 5
main()
{int mytid,task_ids[NPROCS];
int a[size],results[NPROCS],sum=0;
int i,msgtype,num_data=size/NPROCS;
mytid=pvm_mytid();

for(i=0;i<size;i++)
a[i]=i % mytid;

pvm_spawn ("slave3",(char**)0,1,"2-07",NPROCS,task_ids);

for(i=0;i<NPROCS;i++)
{pvm_initsend(PvmDataDefault);
pvm_pkint(&num_data,1,1);
pvm_pkint(&a[num_data*i],num_data,1);
pvm_send(task_ids[i],4);

}
msgtype=7;
for(i=0;i<NPROCS;i++)
{pvm_recv(task_ids[i],msgtype);
pvm_upkint(&results[i],1,1);
sum+=results[i];
}
printf("cymma=%d\n",sum);
pvm_exit();
}
```

Программный код для Slave:

```
#include <pvm3.h>
#include <stdio.h>

main()
```

```
{int mytid,i,sum,*a;
int num_data,master;
mytid=pvm_mytid();
pvm_recv(-1,-1);
pvm_upkint(&num_data,1,1);
a=(int*)malloc(num_data*sizeof(int));
pvm_upkint(a,num_data,1);
sum=0;
for(i=0;i<num_data;i++)
sum+=a[i];
master=pvm_parent();
pvm_initsend(PvmDataDefault);
pvm_upkint(&sum,1,1);
pvm_send(master,7);
pvm_exit();

}
```

Лабораторная работа №3

Рассматривается умножение матриц по алгоритму Кэнона с использованием приемов параллельного программирования.

Краткая теория

Элемент матрицы произведения C находится на пересечении i строки и j столбца представляет собой сумму парных произведений элементов i строки первой матрицы A на элементы j столбца второй матрицы B.

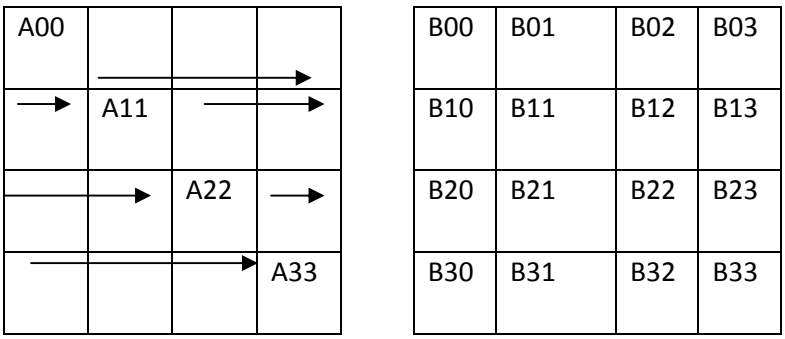
Алгоритм Кэнона

Начальное состояние

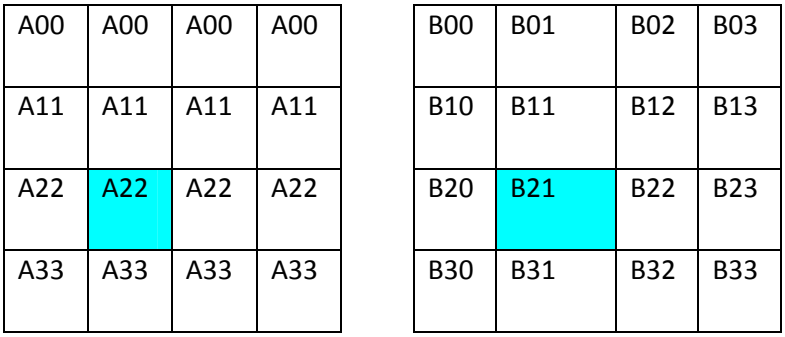
A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

B00	B01	B02	B03
B10	B11	B12	B13
B20	B21	B22	B23
B30	B31	B32	B33

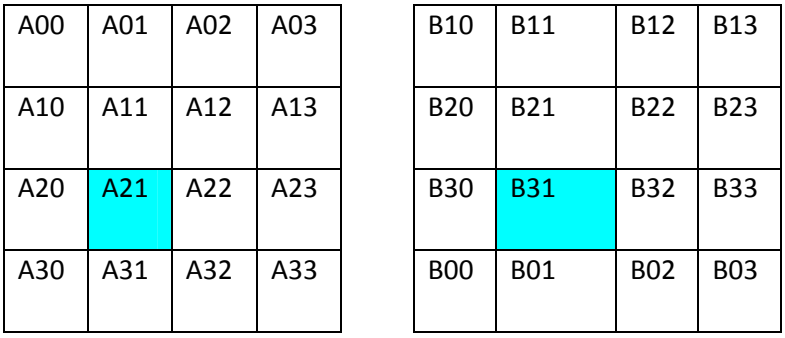
Фаза широковещательной рассылки элементов строк матрицы A(матрица A не изменяется значением широковещательной рассылки, а копируется во временный массив каждого процесса)



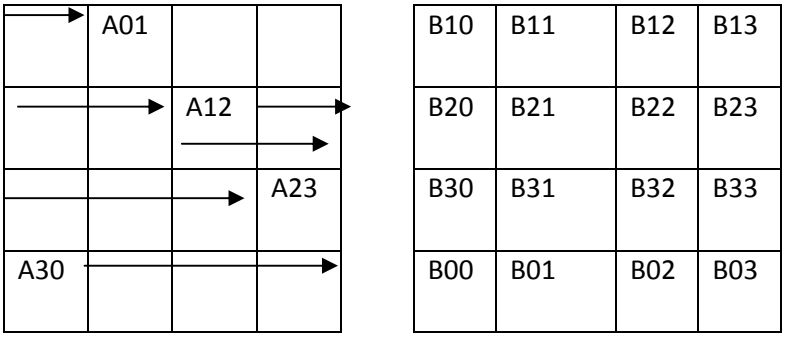
Фаза умножения (A22*B21)



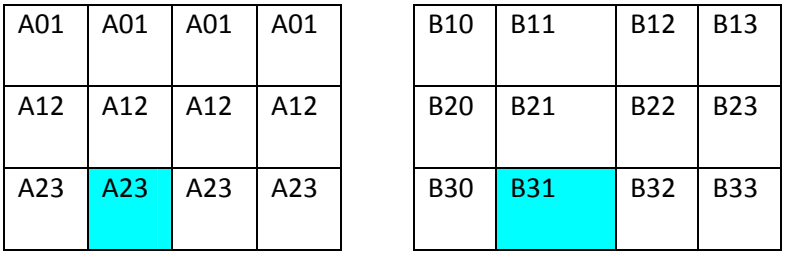
Фаза циклического сдвига матрицы B(каждый столбец вверх на один шаг)



Фаза широковещательной рассылки



Фаза умножения(A23*B31)



A30	A30	A30	A30

B00	B01	B02	B03

Фаза циклического сдвига матрицы B(каждый столбец вверх на один шаг)

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

B20	B21	B22	B23
B30	B31	B32	B33
B00	B01	B02	B03
B10	B11	B12	B13

Фаза широковещательной рассылки

		A02	
			A13
A20			
	A31		

B20	B21	B22	B23
B30	B31	B32	B33
B00	B01	B02	B03
B10	B11	B12	B13

Фаза умножения(A20*B01)

A02	A02	A02	A02
A13	A13	A13	A13
A20	A20	A20	A20
A31	A31	A31	A31

B20	B21	B22	B23
B30	B31	B32	B33
B00	B01	B02	B03
B10	B11	B12	B13

Фаза циклического сдвига матрицы B(каждый столбец вверх на один шаг)

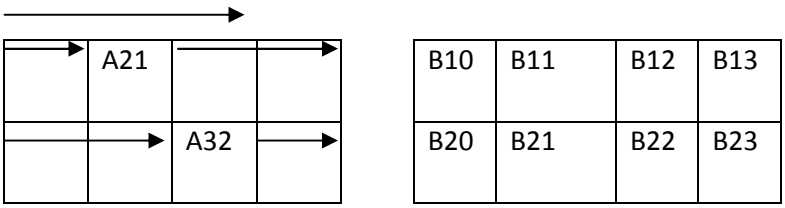
A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

B30	B31	B32	B33
B00	B01	B02	B03
B10	B11	B12	B13
B20	B21	B22	B23

Фаза широковещательной рассылки

			A03
A10			

B30	B31	B32	B33
B00	B01	B02	B03



Фаза умножения(A21*B11)

A03	A03	A03	A03
A10	A10	A10	A10
A21	A21	A21	A21
A32	A32	A32	A32

B30	B31	B32	B33
B00	B01	B02	B03
B10	B11	B12	B13
B20	B21	B22	B23

Фаза циклического сдвига матрицы B(каждый столбец вверх на один шаг)

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

B00	B01	B02	B03
B10	B11	B12	B13
B20	B21	B22	B23
B30	B31	B32	B33

Для успешного выполнения задачи используются дополнительные возможности PVM.

Динамические группы.

Demon PVM выполняет групповую функцию. Для этого используется отдельная библиотека libgpvm3.a, подпрограммы которой должны быть включены в пользовательскую программу на этапе компиляции. Отдельный сервер управления группами запускается автоматически при первом вызове любой групповой функции (н-р: pvm_joingroup()).Любая pvm задача в любое время может входить и выходить из группы, не информируя об этом другие задачи. Задачи могут передавать широковещательные сообщения группам, в состав которых они сами могут не входить.

Любая задача может вызвать любую групповую функцию, за исключением функций:

pvm_lvgroup()
pvm_barrier()
pvm_reduce().

Функции:

1) int inum = pvm_joingroup (char *group);

int info = pvm_ lvgroup (char *group);

Это функции входа и выхода названной группы group. Первый вызов pvm_joingroup – создаёт группу с именем group и включает вызывающую задачу в её состав, возвращает число – номер экземпляра процесса в данной группе. Номера от 0 до (n-1) для n экземпляров. Одна задача может входить в несколько групп. Если задача покидает группу, а затем снова входит в неё, то может получить другой номер экземпляра. Сервер группы всегда выдаёт новому члену наименьший возможный номер. Если

несколько задач покидает группу, то появляются проблемы нумерации членов группы. Если необходима неприрывная нумерация, то об этом должен позаботиться программист.

2) int tid = pvm_gettid (char *group, int inum);

Функция возвращает идентификатор задачи tid для экземпляра inum, группы group. Позволяет задачам, входящим в одну группу, узнавать соответствующие идентификаторы.

3) int inum = pvm_get inst (char *group, int tid);

Функция, обратная pvm_gettid, возвращающая номер экземпляра задачи с указанным идентификатором tid в группе group.

4) int size = pvm_gsize (char *group) –возвращает количество членов, входящих в группу group.

5) int info = pvm_barrier (char *group, int count)- функция синхронизации задач. При вызове данной функции программа блокируется, пока count члена группы group не вызовут функцию pvm_barrier. Обычно в count содержится общее количество членов группы, функцию pvm_barrier может вызвать только член группы. Все члены, вызвавшие функцию pvm_barrier должны указывать одинаковое количество count, иначе ошибка.

6) int info =pvm_bcast (Char *group, int msgtag); - Функция помечает сообщения идентификатора msg_tag и широковещательно посылает всем членам группы group кроме себя (если является членом группы). Если в момент рассылки какая – либо задача выходит из группы или входит, то она не получит сообщение.

7) int info = pvm_mcast (int *tids, int ntask, int msgtag) – Команда передачи данных от одного ко многим, помечает сообщения идентификатором msg_tag и широковещательно рассылает всем задачам, идентификаторы которых перечислены на массив tids, за исключением себя. Размер tids = ntask.

8) int info = pvm_reduce (void (*fund)(),void *data, int nitem, int datatype, int msgtag, char *group, int root) – Выполняет глобальные арифметические операции всеми членами группы. Н-р: вычисление глобальной суммы или максимального значения; в результате вычисления возвращается в root. Существует четыре предопределённые функции:

- pvm_Max – расчёт максимального значения элементов.
- pvm_Min – расчёт минимального значения.
- pvm_Sum – расчёт суммы всех значений.
- pvm_Product – расчёт произведений всех значений, производит поэлементные вычисления.(Н-р: массив Data содержит два числа с плавающей запятой и выполняется расчет pvm_Max, то результат содержит тоже два числа; глобальное максимальное число первого числа среди всех членов группы и второго числа среди всех членов группы). Является не блокирующей функцией. Если какая –либо задача в момент вычисления покидает группу, то возникают ошибки.

Программный код mmult.c:

```
#include "pvm3.h"
#include <stdio.h>
#define MAXNTIDS 100
#define MAXROW 10
#define ATAG 2
#define BTAG 3
#define DIMTAG 5

void
InitBlock(float *a, float *b, float *c, int blk, int row, int col)
{
```

```
int len,ind;
int i,j;

srand(pvm_mytid());
len=blk*blk;
for(ind=0;ind<len;ind++)
 {
  a[ind] =(float)(rand()%1000)/100.0; c[ind]=0.0;}
for(i=0;i<blk;i++){
 for(j=0;j<blk;j++) {
  if(row==col) b[j*blk+i] = (i==j)?1.0:0.0;
   else b[j*blk+i]=0.0;
  }
 }
}

void BlockMult(float *c, float *a, float *b, int blk)
{
 int i,j,k;
 for(i=0;i<blk;i++)
  for(j=0;j<blk;j++)
   for(k=0;k<blk;k++)
    c[i*blk+j]+=(a[i*blk+k]*b[k*blk+j]);
}

int main(int argc, char* argv[])
{
time_t start,end;
int ntask=4, kint;
int info;
int mytid, mygid;
int child[MAXNTIDS-1];
int i,m,blksize;
int myrow[MAXROW];
float *a,*b,*c,*atmp;
int row,col,up,down;
mytid=pvm_mytid();
pvm_setopt(PvmRoute, PvmRouteDirect);

if(mytid<0) {
 pvm_perror(argv[0]);
 return -1;
}

mygid = pvm_joingroup("mmult");
if(mygid<0) {
pvm_perror(argv[0]);
pvm_exit();

return -1;
}
```

```
if(mygid == 0) {
if (argc == 3) {
m=atoi(argv[1]);
blksize=atoi(argv[2]);
}
if(argc<3) {
fprintf(stderr,"usage: mmult m blk\n");
pvm_lvgroup("mmult"); pvm_exit(); return -1;
}

ntask=m*m;
if ((ntask <1) || (ntask > MAXNTIDS)) {
fprintf(stderr, "ntask = %d not valid.\n",ntask);
pvm_lvgroup("mmult");pvm_exit(); return -1;
}

if(ntask == 1) goto barrier;
info = pvm_spawn("mmult",(char**)0,1,"2-04",ntask-1,child);
if(info != ntask-1) {
printf("info != NTASK-1\n");
pvm_lvgroup("mmult");
pvm_exit();
return -1;
}

pvm_initsend(PvmDataDefault);
pvm_pkint(&m,1,1);
pvm_pkint(&blksize,1,1);
pvm_mcast(child,ntask-1,DIMTAG);
}
else
{
pvm_recv(pvm_gettid("mmult",0),DIMTAG);
pvm_upkint(&m,1,1);
pvm_upkint(&blksize,1,1);
ntask=m*m;
}
barrier:
start=time(NULL);
info=pvm_barrier("mmult",ntask);
if (info<0) pvm_perror(argv[0]);

for (i=0; i<m; i++)
 myrow[i] = pvm_gettid("mmult",(mygid/m)*m+i);

a = (float*)malloc(sizeof(float)*blksize*blksize);
b = (float*)malloc(sizeof(float)*blksize*blksize);
c = (float*)malloc(sizeof(float)*blksize*blksize);
atmp = (float*)malloc(sizeof(float)*blksize*blksize);
if(!(a && b && c && atmp)) {
 fprintf(stderr,"%s: out of memmory\n",argv[0]);
free(a);
```

```
free(b);
free(c);
free(atmp);
pvm_lvgroup("mmult");
pvm_exit();
return -1;
}

row = mygid/m; col=mygid % m;
up=pvm_gettid("mmult",((row)?(row-1):(m-1))*m+col);
down = pvm_gettid("mmult",((row==(m-1))?col:(row+1)*m+col));

InitBlock(a,b,c,blksize,row,col);

for(i=0; i<m;i++) {
if(col==(row+i)%m) {
pvm_initsend(PvmDataDefault);
pvm_pkfloat(a,blksize*blksize,1);
pvm_mcast(myrow,m,(i+1)*ATAG);
BlockMult(c,a,b,blksize);
}
else {
pvm_recv(pvm_gettid("mmult",row*m+(row+i)%m), (i+1)*ATAG);
pvm_upkfloat(atmp,blksize*blksize,1);
BlockMult(c,atmp,b,blksize);
}

pvm_initsend(PvmDataDefault);
pvm_pkfloat(b,blksize*blksize,1);
pvm_send(up,(i+1)*BTAG);
pvm_recv(down,(i+1)*BTAG);
pvm_upkfloat(b,blksize*blksize,1);
}

for(i=0;i<blksize*blksize;i++)
 if (a[i] != c[i])
printf("Error a[%d]  (%g) != c[%d] (%g) \n",i,a[i],i,c[i]);
printf("done.\n");
free(a);free(b);free(c);free(atmp);
info=pvm_barrier("mmult",ntask);
if(info<0) pvm_perror(argv[0]);
pvm_lvgroup("mmult");
end=time(NULL);
printf("time = %f",difftime(end,start));

pvm_exit();
return 0;
}
```

Варианты индивидуальных заданий:

1. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны две матрицы A и B с данными типа float и размером mxm, где m-кратно 2. Матрицы организованы в виде одномерного массива. Разбить эти матрицы на подматрицы (блоки) размером 2x2 и вычислить определители 2-го порядка для всех блоков и поместить их в массивы da и db размером (m*m)/4. Далее определить значения элементов матрицы C размером mxm по следующему алгоритму:

Если $da[i] > db[i]$ выполнить:

$$C[i] = A[i] * (da[i])^2 + A[i+1] * da[i]$$

$$C[i+1] = A[i] * (da[i])^2 - A[i+1] * da[i]$$

$$C[i+2] = A[i+2] * (da[i])^2 + A[i+3] * da[i]$$

$$C[i+3] = A[i+2] * (da[i])^2 - A[i+3] * da[i]$$

Иначе, выполнить:

$$C[i] = B[i] * (db[i])^2 + B[i+1] * db[i]$$

$$C[i+1] = B[i] * (db[i])^2 - B[i+1] * db[i]$$

$$C[i+2] = B[i+2] * (db[i])^2 + B[i+3] * db[i]$$

$$C[i+3] = B[i+2] * (db[i])^2 - B[i+3] * db[i]$$

Размер m задавать с клавиатуры, значения матриц A и B заполнять случайными числами со знаком.

2. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны k-массивов $X_0, X_1, \ldots X_{k-1}$ и k-массивов $Y_0, Y_1, \ldots Y_{k-1}$ размерами m и данными типа float. Для каждой i-ой пары массивов X ,Y вычислить значение

$$z = \frac{e^{|x_{max}|} - e^{|y_{max}|}}{\sqrt{|x_{max} * x_{min}|}} * \sqrt{|y_{max} * y_{min}|}$$

, где x_{max}, x_{min} и y_{max}, y_{min} – максимальные и минимальные значения массивов X и Y.

Количество k и размеры массивов X и Y задавать с клавиатуры, значения массивов X и Y заполнять случайными числами со знаком.

3. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны 3 массива A и C размером n и массив B размером m, где m=2n и данными типа float. Вычислить значение функции:

$$z = \frac{\sum_{i=1}^{n} \frac{\sqrt{a_i}}{a^2_{min_i}} + \sqrt{\sum_{i=1}^{m} \frac{|b_i|}{b^2_{min_i}}} + \sqrt[3]{\sum_{i=1}^{n} \frac{\sqrt{c_i}}{c^2_{min_i}}}}{3}$$

Где a_{min_i}, b_{min_i}, c_{min_i} – минимальные значения первых i элементов массивов A, B, C.

Размеры массивов A, B, C задавать с клавиатуры, значения массивов заполнять случайными числами со знаком.

4. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны две матрицы A1 и B1, каждая размером nxm, организованные в виде массивов и данными типа float. Вычислить матрицу произведение C=A1*B1 по ленточному алгоритму. Далее из матриц A1 и B1 надо создать матрицы A2 и B2 по следующему алгоритму:

- если сумма всех элементов матрицы C >= 0, значения элементов матрицы A2 получаются из матрицы A1 по формуле $a2_i = \frac{1}{a1_i^3}$, а матрица B2 представляет собой транспонированную матрицу B1;

- если сумма всех элементов матрицы C < 0, значения элементов матрицы B2 получаются из матрицы A1 по формуле $a2_i = \frac{1}{a1_i^3}$, а матрица A2 представляет собой транспонированную матрицу B1.

Размеры массивов A1, B1 n и m задавать с клавиатуры, значения массивов заполнять случайными числами со знаком.

5. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны k-массивов $A_0, A_1, \dots A_{k-1}$ и k-массивов $B_0, B_1, \dots B_{k-1}$ с размерами n и m соответственно и данными типа float. Для каждой пары массивов A и B вычислить значения функции:

- если x < y рассчитать $t = \frac{y * a_{max}}{b_{max} + b_{min}}$;

- если x => y рассчитать $T = \frac{x * a_{min}}{b_{max} + b_{min}}$,

Где $x = \sum_{i=0}^{N-1} a_i$, $y = \sum_{i=0}^{M-1} b_i$

Количество k и размеры массивов N и M задавать с клавиатуры, значения массивов A и B заполнять случайными числами со знаком.

6. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны k-массивов X размером m. Для каждого массива вычислить функцию $z = \sum_{i=0}^{m-1} y_i$, где для каждого элемента массива X рассчитывается y по формуле:

$$y = \frac{e^{sinx} + \sqrt[3]{sinx}}{1 - \ln(\sqrt{x^2 - a} - 0.82)} + 0.25 * \frac{(x^2 - a) * sinx}{\cos(x^2 - a) + sinx}$$

В программе расчета предусмотреть проверку возможной ошибки деления на ноль. Если знаменатель равен нулю, знаменателю присвоить ничтожно малое число 10^{-24}.

Количество k и размер m массивов X задавать с клавиатуры, значения массивов X заполнять случайными числами со знаком.

7. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны две матрицы A и B с данными типа float и размером mxm, где m-кратно 3. Матрицы организованы в виде одномерного массива. Разбить эти матрицы на подматрицы (блоки) размером 3x3 и вычислить определители 3-го порядка для всех блоков и поместить их в массивы da и db размером (m*m)/9. Далее определить значения элементов матрицы C размером mxm по следующему алгоритму:

Если $da[i] > db[i]$ выполнить:

$C[i] = A[i] * (da[i])^2 + A[i+1] * da[i]$

$C[i+1] = A[i] * (da[i])^2 - A[i+1] * da[i]$

$C[i+2] = A[i+2] * (da[i])^2 + A[i+3] * da[i]$

$C[i+3] = A[i+2] * (da[i])^2 - A[i+3] * da[i]$

Иначе, выполнить:

$$C[i] = B[i] * (db[i])^2 + B[i+1] * db[i]$$

$$C[i+1] = B[i] * (db[i])^2 - B[i+1] * db[i]$$

$$C[i+2] = B[i+2] * (db[i])^2 + B[i+3] * db[i]$$

$$C[i+3] = B[i+2] * (db[i])^2 - B[i+3] * db[i]$$

Размер m задавать с клавиатуры, значения матриц A и B заполнять случайными числами со знаком

8. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны k-массивов X размером n. Для каждого массива вычислить следующие функции:

$u = sint$, если $t > 0$;

$v = \cos(t + \frac{\pi}{4})$, если < 0;

$w = \sqrt{|\frac{x_1 + x_{N-1}}{2}|}$, если $t = 0$;

Где $t = \frac{1}{100} * \sum_{i=0}^{N-1} x_i$

Количество k и размер n массивов X задавать с клавиатуры, значения массивов X заполнять случайными числами со знаком.

9. **Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.**

Даны k-массивов $A_0, A_1, \dots A_{k-1}$ и k-массивов $B_0, B_1, \dots B_{k-1}$ размерами m и данными типа float. Для каждой i-ой пары массивов A ,B вычислить значение

$$Z = \frac{\left(\sum_{i=0}^{M-1} \frac{\sqrt{a_i}}{a_{min}^2} + \sqrt{\sum_{i=0}^{M-1} \frac{|b_i|}{b_{min}^2}}\right)}{2}$$ где x_{max}, x_{min} и y_{max}, y_{min} – максимальные и минимальные значения массивов X и Y.

В программе расчета предусмотреть проверку возможной ошибки деления на ноль. Если знаменатель равен нулю, знаменателю присвоить ничтожно малое число 10^{-24}.

Количество k и размеры массивов X и Y задавать с клавиатуры, значения массивов X и Y заполнять случайными числами со знаком.

10. Определить ускорение параллельной программы по сравнению с последовательной для следующей задачи.

Даны 3 массива: C размером n, B размером m=(2/3)*n и массив A размером k=(1/2)*n и данными типа float. Вычислить значение функции:

$$Z = \frac{\sum_{i=0}^{k-1} cos a_i + \sum_{i=0}^{m-1} sin^2 b_i}{ln \sum_{i=0}^{n-1} |c_i|}$$

В программе расчета предусмотреть проверку возможной ошибки деления на ноль. Если знаменатель равен нулю, знаменателю присвоить ничтожно малое число 10^{-24}.

Размеры массивов A, B, C задавать с клавиатуры, значения массивов заполнять случайными числами со знаком.

В программе расчета предусмотреть проверку возможной ошибки деления на ноль. Если знаменатель равен нулю, знаменателю присвоить ничтожно малое число 10^{-24}.

3 Методические указания по самостоятельной работе

Для успешного освоения курса «Б.1.В.ДВ.2.1 Параллельное программирование» необходима самостоятельная работа. В настоящее время актуальными становятся требования к личным качествам современного студента – умению самостоятельно пополнять и обновлять знания, вести самостоятельный поиск необходимого материала, быть творческой личностью.

Самостоятельную работу по освоению дисциплины обучающимся следует начинать с изучения содержания рабочей учебной программы дисциплины, цели и задач, структуры и содержания курса, основной и дополнительной литературы, рекомендованной для самостоятельной работы.

Самостоятельная учебная деятельность является необходимым условием успешного обучения. Многие профессиональные навыки, способность мыслить и обобщать, делать выводы и строить суждения, выступать и слушать других, – все это развивается в процессе самостоятельной работы студентов.

Самостоятельная работа по освоению дисциплины включает:

- самостоятельное изучение разделов;
- самоподготовку (проработку и повторение лекционного материала и материала учебников и учебных пособий);
- подготовку к лабораторным работам;
- подготовку к рубежному и итоговому контролю.

Самостоятельная учебная работа – условие успешного окончания высшего учебного заведения. Она является равноправной формой учебных занятий, наряду с лекциями, семинарами, экзаменами и зачетами, но реализуемая во внеаудиторное время.

Эффективность аудиторных занятий во многом зависит от того, насколько умело студенты организуют в ходе них свою самостоятельную учебную познавательную

деятельность. Такая работа также способствует самообразованию и самовоспитанию, осуществляемому в интересах повышения профессиональных компетенций, общей эрудиции и формировании личностных качеств.

Самостоятельная работа реализуется:

1. непосредственно в процессе аудиторных занятий – на лекциях, лабораторных занятиях, при проведении рубежного контроля;
2. в контакте с преподавателем вне рамок расписания – на консультациях по учебным вопросам, при ликвидации задолженностей, при выполнении индивидуальных заданий;
3. в библиотеке, дома, в общежитии, на кафедре при выполнении студентом учебных задач.

В процессе проведения самостоятельной работы необходимо производить подбор литературных источников, научной периодической печати и т.д

4 Методические указания по итоговому контролю

Итоговый контроль знаний по дисциплине «Б.1.В.ДВ.2.1 Параллельное программирование» проводится в форме экзамена. Для подготовки к итоговому контролю знаний по дисциплине «Б.1.В.ДВ.2.1 Параллельное программирование» обучающиеся используют перечень вопросов, приведенный в фонде оценочных средств. Экзамен проводится в устной форме. В экзаменационный билет включен один теоретический вопрос. На подготовку студенту отводится 20-25 минут. На дифференцированном зачете ответы обучающегося оцениваются с учетом их полноты, правильности и аргументированности с учетом шкалы оценивания.

Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, использует в ответе профессиональные термины, правильно обосновывает принятое решение.

Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов, владеет необходимыми навыками и приемами их выполнения.

Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту за отсутствие знаний по дисциплине, представления по вопрос, непонимание материала по дисциплине, наличие коммуникативных «барьеров» в общении, отсутствие ответа на предложенный вопрос.

5 Учебно-методическое обеспечение дисциплины

5.1 Основная литература

1 . **Назаров С. В. Современные операционные системы: учебное пособие** [Электронный ресурс] / Назаров С. В., Широков А. И. - Интернет-Университет Информационных Технологий, 2011. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=233197

5.2 Дополнительная литература

1. **Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ** [Электронный ресурс] / Уильямс Э. - ДМК Пресс, 2012. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=232041

5.3 Периодические издания

1. Журнал «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ».
2. Журнал «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ».
3. Журнал «МИР ПК + DVD».
4. Журнал «ВЕСТНИК КОМПЬЮТЕРНЫХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ».
5. Журнал «ОТКРЫТЫЕ СИСТЕМЫ. СУБД».
6. Журнал «ЖУРНАЛ СЕТЕВЫХ РЕШЕНИЙ/ LAN».

5.4 Интернет-ресурсы

1. http://www.csm.ornl.gov/pvm/ - официальный сайт разработчиков параллельной виртуальной машины PVM.
2. http://www.dcc.ufrj.br/~gabriel/progpar/pvm-book2.pdf - документация и руководство по PVM.
3. http://rsusu1.rnd.runnet.ru/tutor/method/m2/page03.html - базовые функции MPI.
4. https://www.opennet.ru/docs/RUS/MPI_intro/- введение в технологию MPI.
5. http://www.openmp.org/ - официальный сайт разработчиков технологии OpenMP.

5.5 Программное обеспечение, профессиональные базы данных и информационные справочные системы современных информационных технологий

Тип программного обеспечения	Наименование	Схема лицензирования, режим доступа
Операционная система	CentOS Linux	Свободное ПО, https://www.centos.org/legal/
Текстовый редактор	nano	Свободное ПО, является компонентом операционных систем UNIX, Linux и т.п.
Набор средств разработки программного обеспечения	GNU Compiler Collection (gcc)	Свободное ПО, https://gcc.gnu.org/

Тип программного обеспечения	Наименование	Схема лицензирования, режим доступа
Программный пакет для реализации параллельных вычислений	Parallel Virtual Machine (pvm)	Свободное ПО, http://www.csm.ornl.gov/pvm/
	MPICH	Свободное ПО, http://git.mpich.org/mpich.git/blob/HEAD:/COPYRIGHT

6 Материально-техническое обеспечение дисциплины

Учебные аудитории для проведения занятий лекционного типа, семинарского типа, для проведения групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации. Для проведения лабораторных и практических работ используются компьютерный класс (ауд. № 4-113, 4-116, 4-117), оборудованный средствами оргтехники, программным обеспечением, персональными компьютерами, объединенными в сеть с выходом в Интернет.

Аудитории оснащены комплектами ученической мебели, техническими средствами обучения, служащими для представления учебной информации большой аудитории.

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой, подключенной к сети «Интернет», и обеспечением доступа в электронную информационно-образовательную среду Орского гуманитарно-технологического института (филиала) ОГУ (ауд. № 4-307).

Наименование помещения	Материальное-техническое обеспечение
Учебные аудитории: - для проведения занятий лекционного типа, семинарского типа, - для групповых и индивидуальных консультаций; - для текущего контроля и промежуточной аттестации	Учебная мебель, классная доска, мультимедийное оборудование (проектор, экран, ноутбук с выходом в сеть «Интернет»)
Компьютерные классы № 4-113, 4-116, 4-117	Учебная мебель, компьютеры (29) с выходом в сеть «Интернет», проектор, экран, лицензионное программное обеспечение
Помещение для самостоятельной работы обучающихся, для курсового проектирования (выполнения курсовых работ)	Учебная мебель, компьютеры (3) с выходом в сеть «Интернет» и обеспечением доступа в электронную информационно-образовательную среду Орского гуманитарно-технологического института (филиала) ОГУ, программное обеспечение

Для проведения занятий лекционного типа используются следующе наборы демонстрационного оборудования и учебно-наглядные пособия:

- презентации к курсу лекций.