МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТЕТ» (ДГТУ)

Кафедра «Информационные технологии»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «СПЕЦКУРС ДЛЯ ОБЪЕКТОВ ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ»

Ростов-на-Дону
ДГТУ
2023

УДК 372.8:004
Составители: М. В. Привалов

Методические указания для выполнения лабораторных работ по дисциплине «Спецкурс для объектов профессиональной деятельности». - Ростов-на-Дону: Донской гос. техн. ун-т, 2023. – 17 с.

Методические указания освещают вопросы построения алгоритмов параллельной обработки данных с использованием различных технологий распараллеливания: стандарта OpenMP, интерфейса MPI и технологии вычислений общего назначения на GPU - OpenCL.

Предназначены для студентов направления 09.03.03 «Прикладная информатика» очной и очно-заочной форм обучения.

УДК 372.8:004

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Информационные технологии»,
д-р техн. наук, профессор Б.В. Соболь

В печать ___.___. 20___г.
Формат 60×84/16. Объем___усл.п.л.
Тираж ___экз. Заказ №___.

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1

ОГЛАВЛЕНИЕ

Лабораторная работа №1.
Создание простейшей программы с использованием OpenMP

Цель работы: изучить способы создания параллельных программ для симметричных мультипроцессоров с использованием стандарта OpenMP, освоить особенности конкурентного выполнения кода в нескольких потоках.

Теоретические сведения

OpenMP – это стандартное API для написания параллельных приложений, которое основано на абстракции разделяемого адресного пространства (C, C++, Fortran)

Состоит из:

- директив компилятора
- функций времени выполнения
- переменных окружения

Спецификация стандарта поддерживается организацией OpenMP Architecture Review Board: http://www.openmp.org Актуальная спецификация стандарта – OpenMP 4.0.

- Для использования OpenMP требуется, чтобы компилятор языка программирования
  - поддерживал директивы OpenMP
  - содержал набор run-time библиотек
- Проект Visual C++, который использует OpenMP, должен компилироваться с ключом **/openmp :**
  - (Свойства конфигурации->C/C++->Язык)
- Компилятору GCC, который используется в Unix-подобных системах (или, допустим, в Code Blocks) требуется указать ключ **–fopenmp**
  - ***Примечание: при использовании GCC в окружении MinGW под Windows может оказаться, что компилятор собран без этой поддержки. Выходом из такой ситуации является только замена компилятора***

OpenMP использует модель исполнения fork-join. Из главного потока (нити) порождаются потоки-рабочие, которые по окончании сливаются снова с главным.

Для использования технологии OpenMP в коде C/C++ помимо установки флагов компилятора необходимо подключить библиотеку omp:

#include <omp.h>

после чего можно использовать директивы и библиотеку функций.

Синтаксис директив OpenMP

#pragma omp directive_name [clause[clause ...]]
<структурный блок >

Действия, соответствующие директиве, применяются непосредственно к структурному блоку, расположенному за директивой. Структурным блоком может быть любой оператор, имеющий единственный вход и единственный выход.

Количество нитей, выполняющих работу, определяется переменной окружения OMP_NUM_THREADS или вызовом функции omp_set_num_threads().

Определение нитью «своих» координат в вычислительном пространстве:

- свой номер: **omp_get_thread_num()**;
- общее число нитей: **omp_get_num_threads()**.

Как уже отмечалось выше, мастер-нить имеет номер 0, у остальных же нитей номера больше 0 и идут по порядку. При этом планировщик нитей автоматически распределяет их по доступным вычислительным ресурсам (назначает процессорам и ядрам).

Данные функции, несмотря на их простоту, очень важны, так как индивидуальный номер нити и их общее число – это параметры, на основе которых определяется, какая именно работа будет выполняться нитями.

Основная директива распараллеливания, которая задаёт параллельный блок – это
#pragma omp parallel [clause ...] newline <структурный блок >

Возможные параметры clause приведены ниже:

– **if (scalar_expression)** – нити для выполнения блока создаются только если условие в if выполняется;

– **private (list)** - определяет список переменных, которые будут локальными для каждого потока; переменные создаются в момент формирования потоков параллельной области; начальное значение переменных является неопределенным;

– **shared (list)** - определяет список переменных, которые будут общими для всех потоков параллельной области; правильность использования таких переменных должна обеспечиваться программистом;

– **reduction (operator: list)** - определяет оператор и переменную для редуцирования;

– **num_threads(integer-expression)** - определяет количество потоков, которые будут выполнять код; по умолчанию – все доступные на этом устройстве.

Задание на лабораторную работу

Без директив параллельных циклов и без sections/section в рамках параллельного блока решите задания, описанные ниже.

Часть 1.

1. Создайте новый проект C/C++.
2. Включите использование OpenMP в проекте.
3. Напишите простую программу, которая в 5 нитей выведет на экран сообщение:
   «Привет! Я – поток #»
   * вместо # должен быть номер нити, выполняющей вывод.

Запустите программу несколько раз подряд. Поясните результаты.

Часть 2.

1. Модифицируйте программу таким образом, чтобы чётные нити продолжили выводить на экран сообщение из **Части 1**, а нечётные заполнили значениями массив размером N*10, где N – номер варианта (номер студента по списку группы). При этом обеспечьте, чтобы нечётные потоки были как можно более равномерно загружены работой (разница в количестве заполняемых элементов массива, порученных разным нитям, не должна превышать 1 элемент).

Контрольные вопросы

1. Что такое OpenMP?
2. Для каких систем пригоден OpenMP?
3. Каким образом обеспечивается параллелизм?
   Сколько нитей(потоков) будет запущено для параллельного блока? Можно ли это

изменить и как?

4. Каковы потенциальные проблемы параллельной обработки данных в OpenMP?

Лабораторная работа №2
Распараллеливание алгоритмов с использованием OpenMP

Цель работы: изучить способы создания параллельных программ для симметричных мультипроцессоров с использованием стандарта OpenMP.

Теоретические сведения

Директива распараллеливания циклов. Для того, чтобы распределить выполнение цикла по имеющимся ядрам, используется следующая директива:

#pragma omp for [clause ...] newline for loop

Распределение итераций в директиве for регулируется параметром (clause) schedule:

- static – итерации делятся на блоки по chunk итераций и статически разделяются между потоками; если параметр chunk не определен, итерации делятся между потоками равномерно и непрерывно;
- dynamic – распределение итерационных блоков осуществляется динамически (по умолчанию chunk=1);
- guided – размер итерационного блока уменьшается по экспоненциальному закону при каждом распределении;
- chunk определяет минимальный размер блока (по умолчанию chunk=1);
- runtime – правило распределения определяется переменной OMP_SCHEDULE (при использовании runtime параметр chunk задаваться не должен).

Директива параллельных секций. Для выделения раздельных фрагментов кода в параллельные области используется директива **sections**. Отметим ряд свойств этой директивы:

- каждый фрагмент выполняется однократно (директива **section**);
- разные фрагменты выполняются разными потоками;
- завершение директивы по умолчанию синхронизируется;
- директивы **sections** должны использоваться только в статическом контексте.

Директива непараллельных секций. Чтобы выполнить блок только один раз, используется директива single

#pragma omp single [name]
<структурный блок >

Для запрета конкурентного выполнения блока кода используется директива critical:

#pragma omp critical [name]
<структурный блок >

Отличия этих директив друг от друга состоит в том, что блок single выполняется только единожды каким-либо потоком, а блок critical – всеми потоками, но только одним одновременно.

Задание на лабораторную работу

Распараллельте заданный алгоритм с использованием OpenMP. Оцените ускорение многопоточного выполнения. При создании параллельной программы требуется использовать имеющиеся инструменты OpenMP: параллельный цикл и параллельную секцию.

1. Распараллельте алгоритм с использованием параллельного цикла OpenMP. Оцените время выполнения
2. Распараллельте этот же алгоритм с использованием механизма параллельных секций OpenMP. Тоже оцените время выполнения.
3. Оцените время выполнения последовательного варианта.

Сравните быстродействие во всех трёх случаях и поясните результаты. Варианты заданий приведены в табл. 1.

Таблица 1.

Варианты заданий к лабораторной работе №2

Вариант	**Операции с векторами**
1	C = A + B
2	C = A – B*x
3	a = (B*C)
4	a = MAX(B)*C
5	b = MIN(A+C)
6	A = B*MIN(C)
7	A = B + C – D*e
8	C = A - B + D
9	d = MAX(A + B + C)
10	d = ((A + B)* C)
11	d = (A * (B + C))
12	d = (A*B) + (C*B)
13	d = (B*C) - (A*B) +(C*B)
14	E = A + B + C – D*e
15	E = A + C – B*e + D
16	e = ((A +B)*(C + D))

Пояснения: A, B, C, D – векторы некоторого размера (одинакового для всех в рамках одного варианта); e, x – некоторые скалярные величины, задаваемые пользователем.

Контрольные вопросы

1. Какими директивами выполняется распределение работы в OpenMP?
2. Как выполняется распределение работы вручную в OpenMP?
3. Как разделяется работа в цикле for в OpenMP?
4. Какие варианты расписания существуют для распределения работы цикла?
5. Зачем нужна редукция?

Лабораторная работа №3
Блокирующий и неблокирующий обмен. Дедлоки.

Цель работы: получить навыки работы в MPI с блокирующими и неблокирующими обменами данных.

Теоретические сведения

Попарная коммуникация (от точки к точке), в отличие от коллективной коммуникации (содержащей группу задач), включает передачу сообщения между одной парой процессов.

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

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

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

В дополнение к выбору способа коммуникации программист должен решить, будут ли вызовы отправки и получения блокирующими или неблокирующими. Блокирующая или неблокирующая отправка может быть спарена с блокирующим или неблокирующим получением. Для функции приема MPI_Recv не существует различных режимов работы.

Неблокирующие операции передачи сообщений. В отличие от блокирующих операций, возврат управления из неблокирующих операций производится немедленно. Поэтому необходимо следить за тем, чтобы не изменить данные до того, как они отправились, либо за тем, чтобы не начать обработку еще не пришедших данных. Для того, чтобы узнать, пришло (отправилось) ли полностью сообщение, используется специальная функция. Ниже следует описание необходимых функций.

int MPI_Isend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request),

int MPI_Issend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request),

int MPI_Ibsend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request),

int MPI_Irsend(void *buf, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm, MPI_Request *request),

int MPI_Irecv(void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm, MPI_Request *request).

Вызов неблокирующей функции приводит к инициации запрошенной операции передачи, после чего выполнение функции завершается и процесс может продолжить свои действия. Перед своим завершением неблокирующая функция определяет переменную request, которая далее может использоваться для проверки завершения инициированной операции обмена.

Проверка состояния выполняемой неблокирующей операции передачи данных производится при помощи функции:

int MPI_Test(MPI_Request *request, int *flag, MPI_status *status), где

- request — дескриптор операции, определенный при вызове неблокирующей функции;
- flag — результат проверки (true, если операция завершена);
- status — результат выполнения операции обмена (только для завершенной операции).

Операция проверки является неблокирующей, т.е. процесс может проверить состояние неблокирующей операции обмена и продолжить далее свои вычисления, если по результатам проверки окажется, что операция все еще не завершена. Возможная схема совмещения вычислений и выполнения неблокирующей операции обмена может состоять в следующем:

```
MPI_Isend(buf, count, type, dest, tag, comm, &request);
.....
do {
    ....
    MPI_Test(&request, &flag, &status);
} while (!flag);
```

Если при выполнении неблокирующей операции окажется, что продолжение вычислений невозможно без получения передаваемых данных, то может быть использована блокирующая операция ожидания завершения операции:

int MPI_Wait(MPI_Request *request, MPI_status *status),

где

- request — дескриптор операции, определенный при вызове неблокирующей функции;
- status — результат выполнения операции обмена (только для завершенной операции).

Кроме рассмотренных, MPI содержит ряд дополнительных функций проверки и ожидания неблокирующих операций обмена:

- MPI_Testall — проверка завершения всех перечисленных операций обмена;
- MPI_Waitall — ожидание завершения всех операций обмена;
- MPI_Testany — проверка завершения хотя бы одной из перечисленных операций обмена;
- MPI_Waitany — ожидание завершения любой из перечисленных операций обмена;
- MPI_Testsome — проверка завершения каждой из перечисленных операций обмена;
- MPI_Waitsome — ожидание завершения хотя бы одной из перечисленных операций обмена и оценка состояния по всем операциям.

Ход работы:

Составить программу с использованием блокирующих и неблокирующих операций, которая ускоряет вычисление формулы, согласно варианту, приведенному в табл. 2.

Таблица 2.

Варианты заданий к лабораторной работе №3.

Вариант	**Операции с векторами**
1	A = SORT(B) + SORT(C)
2	C = A + B
3	C = A – B*x
4	C = A + SORT(B)
5	C = SORT(A) - SORT(B)
6	a = (B*C)
7	b = (A*SORT(C))
8	a = MAX(B)
9	b = MIN(A+C)
10	A = B*MIN(C)
11	c = MAX(A)*(A*B)
12	A = B + C – D*e
13	C = A - B + D
14	SORT(A + B) - C
15	d = MAX(A + B + C)
16	d = ((A + B)* C)
17	d = (A * (B + C))
18	d = (A*B) + (C*B)
19	d = MAX(B - C) + MIN(A + B)
20	D = MIN(A + B) * (B – C + X)
21	D = SORT(A) + SORT(B) - SORT(C)
22	d = (B*C) - (A*B) +(C*B)

Вариант	Операции с векторами
23	E = A + B + C – D*e
24	E = A + C – B*e + D
25	e = ((A +B)*(C + D))

Обеспечить выполнение операций в нескольких процессах, при этом предусмотреть случаи, когда размер задачи не делится нацело на количество запущенных процессов. Раздача исходных данных должна выполняться с использованием неблокирующих операций, а сбор результатов в «главный» процесс – с помощью блокирующих. Оцените ускорение.

Объясните, что такое «мёртвая блокировка» (дэдлок), при каких условиях она может появиться в вашей программе и как таких ситуаций избегать.

Условные обозначения

a - скалярная величина
A - вектор размерности N
MA - матрица размерности NxN
a*B - произведение вектора на скаляр
a*MB - произведение матрицы на скаляр
(A*B) - скалярное произведение векторов A и B
(MA*MB) - произведение матриц MA и MB
(MA*B) - произведение матрицы на вектор
SORT(A) - сортировка вектора A по возрастанию
MIN(A) - поиск минимального элемента вектора
MAX(A) - поиск максимального элемента вектора
TRANS(MA) - транспонирование матрицы MA
MAX(MA) - поиск максимального элемента матрицы
MIN(MA) - поиск минимального элемента матрицы
SORT(MA) - сортировка строк матрицы по убыванию

Контрольные вопросы

1. Для каких систем предназначен MPI?
2. В чём отличие MPI от OpenMP?
3. За счёт чего может возникнуть мёртвая блокировка в программе, использующей MPI?
4. Чем отличаются блокирующие и неблокирующие двухточечные операции MPI?
5. Как проверить статус неблокирующей коммуникации?

Лабораторная работа №4.
Коллективные операции передачи данных

Цель работы: освоить способы эффективной организации параллелизма в MPI с применением коллективных операций пересылки данных.

Теоретические положения

Функции MPI_Send и MPI_Recv, рассмотренные ранее, обеспечивают возможность выполнения парных операций передачи данных между двумя процессами параллельной программы. Для выполнения коммуникационных коллективных операций, в которых принимают участие все процессы коммуникатора, в MPI предусмотрен специальный набор функций (табл.3).

Таблица 3.

Коллективные операции передачи данных

Вид коллективной операции	**Функции MPI**
Передача от одного процесса всем процессам (широковещательная рассылка)	MPI_Bcast
Сбор и обработка данных на одном процессе от всех процессов (редукция данных)	MPI_Reduce
- то же с рассылкой результатов всем процессам	MPI_Allreduce MPI_Reduce_scatter
- то же с получением частичных результатов обработки	MPI_Scan
Обобщенная передача от одного процесса всем процессам (распределение данных)	MPI_Scatter MPI_Scatterv
Обобщенная передача от всех процессов одному процессу (сбор данных)	MPI_Gather MPI_Gatherv
- то же с рассылкой результатов всем процессам	MPI_Allgather MPI_Allgatherv
Обобщенная передача данных от всех процессов всем процессам	MPI_Alltoall MPI_Alltoallv

Операция редукции. Операцией редукции данных называется операция обработки данных, полученных на одном процессе от всех процессов. Для наилучшего выполнения действий, связанных с редукцией данных, в MPI предусмотрена функция:

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

где op — операция, которая должна быть выполнена над данными;

В качестве операций редукции данных могут быть использованы предопределенные в MPI операции – см. табл. 4 и 5.

Таблица 4.

Базовые (предопределенные) типы операций MPI для функций редукции данных

Операции	**Описание**
MPI_MAX	Определение максимального значения
MPI_MIN	Определение минимального значения
MPI_SUM	Определение суммы значений
MPI_PROD	Определение произведения значений
MPI_LAND	Выполнение логической операции "И" над значениями сообщений
MPI_BAND	Выполнение битовой операции "И" над значениями сообщений
MPI_LOR	Выполнение логической операции "ИЛИ" над значениями сообщений
MPI_BOR	Выполнение битовой операции "ИЛИ" над значениями сообщений
MPI_LXOR	Выполнение логической операции исключающего "ИЛИ" над значениями сообщений
MPI_BXOR	Выполнение битовой операции исключающего "ИЛИ" над значениями сообщений
MPI_MAXLOC	Определение максимальных значений и их индексов
MPI_MINLOC	Определение минимальных значений и их индексов

Помимо данного стандартного набора операций могут быть определены и новые дополнительные операции непосредственно самим пользователем библиотеки MPI.

Таблица 5.

Разрешенные сочетания операции типа операнда в операции редукции

Операции	**Допустимый тип операндов для языка С**
MPI_MAX,MPI_MIN,MPI_SUM,MPI_PROD	Целый, вещественный
MPI_LAND, MPI_LOR, MPI_LXOR	Целый
MPI_BAND, MPI_BOR, MPI_BXOR	Целый, байтовый
MPI_MINLOC, MPI_MAXLOC	Целый, вещественный

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

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

Отметим также, что функция MPI_Scatter передает всем процессам сообщения одинакового размера. Выполнение более общего варианта операции распределения данных, когда размеры сообщений для процессов могут быть различны, обеспечивается при помощи функции MPI_Scatterv.

Операция обобщенной передачи данных от всех процессов одному процессу (сбор данных) является двойственной к процедуре распределения данных. Для неё используются функции MPI_Gather и MPI_Gatherv. Всё сказанное выше о функциях распределения справедливо и для функций сбора данных.

Задание на лабораторную работу

Используя коллективные операции, решите поставленную задачу. Рекомендуется использовать операции из набора: широковещательная рассылка, редукция, распределение и сборка данных. Считать, что количество данных и количество доступных вычислительных узлов (запускаемых процессов) может быть произвольным. То есть считать, что в общем случае количество строк (столбцов, элементов и т.п.) может не делиться нацело на количество рабочих процессов, за исключением случаев, где это специально оговорено заданием. Обеспечить корректную работу программы при любых разумных размерностях данных и количествах процессов.

Варианты заданий

1. Дана матрица MxN, сгенерированная случайным образом, и вектор из N элементов. Считая каждую строку матрицы вектором, найти такую, которая бы имела наименьшее скалярное произведение на заданный вектор.
2. В матрице MxN определить сумму скалярных произведений всех возможных пар соседних строк.
3. Имеется вещественная матрица размера MxN. Определить номер строки, в которой необходимо произвести максимальное количество перестановок для её сортировки по возрастанию.
4. Дана матрица размером MxN, где M - чётное. Определить максимальную сумму элементов чётных строк.
5. Дан вектор натуральных чисел. Получить новый вектор, в котором элемент равен 1, если соответствующий элемент исходного вектора – простое число, и 0 в противном случае.
6. Дана матрица размером MxN. Определить строку, имеющую минимальную сумму элементов.
7. Реализовать алгоритм поиска простых чисел с равномерным распределением проверяемых чисел между имеющимися вычислительными узлами.

8. Дан вектор из N элементов. Найти его сумму с использованием модифицированной каскадной схемы и групповых операций. N задать самостоятельно.
9. Дан вектор, в котором каждая группа из 4х элементов представляет собой координаты прямоугольника на плоскости. Найти прямоугольник с наибольшим периметром.
10. Считая строки матрицы размером MxN векторами N-мерного пространства, определить, какая из них ближе всего к вектору, введенному пользователем. Используйте метрику Евклида.
11. Дан вектор размером N, где N кратно 6. Каждая группа из 6 элементов может быть рассмотрена как координаты вершин треугольника на плоскости. Среди групп, которые формируют треугольники, найти таковой с минимальной площадью.
12. Дан вектор размером N элементов, где N кратно 100. Распределить вектор блоками по 100 элементов и найти блок с минимальным значением оценки локальной дисперсии.
13. Дана матрица размером MxN. Используя коллективные операции рассчитать сумму элементов столбца, номер которого задан пользователем.
14. В матрице размером MxN, хранящей значения бит, сформировать N+1й столбец проверки на чётность.
15. К матрице размером MxN добавить N+1й столбец, содержащий суммы элементов строк. Найти строку с максимальной суммой.
16. Имеется матрица MxN, хранящая значения яркости пикселей изображения (полутоновое, 8 бит на пиксель). Выполнить линейную фильтрацию изображения фильтром, имеющим следующее ядро:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

17. Матрица размером MxN содержит пары строк. Каждая пара задаёт координаты множества точек на плоскости. Определите два множества, расстояние между центрами масс которых минимально.
18. Даны два множества точек на плоскости. Определить точку второго множества, расстояние от которой до центра масс первого минимально.
19. Дан фрагмент текста. Определить в тексте самое длинное слово.
20. В квадратной матрице выполнить циклический сдвиг элементов каждой строки таким образом, чтобы максимальный элемент строки оказался на главной диагонали.
21. Найдите в квадратной матрице размером MxM такое i, что скалярное произведение i-го столбца на i-ю строку минимально.
22. Рассматривая группы по 4 элемента вектора как координаты двух противоположных вершин прямоугольника, найти прямоугольник, центр которого расположен наиболее близко по отношению к началу координат.

Контрольные вопросы

1. Для чего нужны коллективные операции MPI?
2. Чем широковещание отличается от редукции?
3. Что такое обобщённая передача в MPI?
4. Чем отличаются функции MPI_Scatter и MPI_Scatterv?

Лабораторная работа №5.
Обработка векторных данных с использованием фреймворка OpenCL.

Цель работы: получить навыки создания простейших параллельных программ с использованием фреймворка OpenCL.

Теоретические положения

OpenCL – это фреймворк для создания параллельных программ, использующих гетерогенные вычисления. Фреймворк является открытым стандартом, основанным на C99 и позволяющим организовывать параллельные вычисления, в которых участвуют процессоры, графические ускорители (GPGPU вычисления, General Purpose GPU computing). В настоящее время существует несколько версий API, последняя из которых официально поддерживаемая имеющимися на рынке устройствами – это OpenCL 1.2. Подробности в различиях версий стандартов и списки поддерживаемых устройств можно найти на официальном сайте группы Khronos, являющейся разработчиком: http://www.khronos.org/

Для распараллеливания алгоритмов с использованием OpenCL понадобится что-либо из следующего списка аппаратных средств:

- процессор Intel или AMD, поддерживающий SIMD-инструкции SSE4.1
- процессор Intel не ниже Core 4го поколения с интегрированным ускорителем
- графический ускоритель nVidia с графическим ядром поколения не ниже GF100 (GeForce 400 Series или специализированный ускоритель из серии Tesla)
- графический ускоритель AMD/ATI поколения не ниже Radeon HD5xxx

Программную поддержку OpenCL, в случае графических ускорителей, обеспечит проприетарный драйвер от производителя видеокарт, а также соответствующий SDK.

- Для видеокарт nVidia рекомендуется CUDA SDK, включающий поддержку как архитектуры CUDA, так и OpenCL.
- Для видеокарт ATI рекомендуется AMD Software SDK
- Для использования с интегрированными ускорителями или процессорами Intel – Intel SDK for OpenCL Applications 2013.

Все SDK являются бесплатными и доступными на сайтах производителей соответствующего аппаратного обеспечения.

Программа, использующая фреймворк, состоит из двух основных частей, которые отличаются языком программирования и способом исполнения:

- host-программа;
- Программа OpenCL.

Host-программа – это обычная, или «основная» программа, написанная на одном из языков программирования и занимающаяся подготовкой данных и организацией вычислений. Эта программа исполняется, как обычно, на центральном процессоре.

OpenCL-программа – это программа, написанная на Си-подобном языке и которая исполняется средствами фреймворка на выбранном устройстве. Эта программа состоит из функций и функций-ядер.

Ядром (kernel) называют функцию, обеспечивающую точку доступа host-программе. Именно с ядра начинается каждый процесс исполнения параллельного кода, инициируемый host-программой.

Распараллеливание производится, как правило, по данным, так как архитектура OpenCL ориентирована на исполнение, которое по таксономии Флинна наиболее близко к SIMD. Механизм заключается в следующем. При исполнении программы команда, подающая ядро на

исполнение, создает коллекцию рабочих элементов, каждый из которых исполняет один и тот же код, поэтому поведение рабочего элемента зависит от ветвлений внутри кода или, чаще, от данных, выбранных по индексу globalID. Для более крупного деления рабочих элементов, они могут объединяться в рабочие группы. Каждой рабочей группе назначается её уникальный идентификатор в глобальном пространстве индексов, а рабочему элементу – локальный идентификатор внутри группы. Поэтому доступ к данным может осуществляться не только по глобальному индексу, но и по идентификаторам рабочей группы плюс локальному идентификатору элемента. При этом элементы внутри группы работают одновременно.

Пространство индексов охватывает N-мерное пространство, и поэтому называется NDRange. В настоящее время OpenCL поддерживает N, равное 1, 2 или 3. Таким образом, структура NDRange однозначно определяет адресацию с помощью индексов, см. рис.

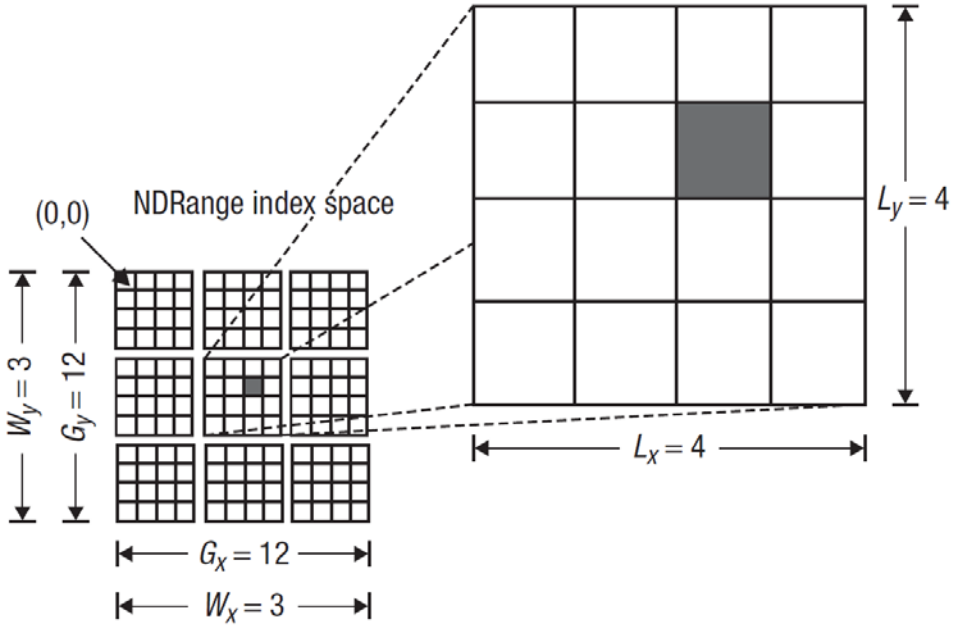

Рисунок. Пример того, как локальный, глобальный и групповой идентификаторы связаны с NDRange. Выделенный блок имеет глобальный идентификатор (gx, gy)=(6, 5), а групповой плюс локальный (wx, wy) = (1, 1), (lx, ly)=(2, 1).

Ход работы:

Составить программу с использованием OpenCL согласно варианту задания, указанному в лабораторной работе №3. Для организации вычислений организуйте 1-мерный NDrange.

Обеспечьте выполнение программы с ускорением на:

- CPU
- Графическом ускорителе

Оцените время исполнения в обоих случаях. Поясните результаты.

Контрольные вопросы

1. Что такое OpenCL и для чего он предназначен?
2. Какие устройства поддерживает OpenCL?
3. Что такое контекст и зачем он нужен?
4. Как называется основная единица исполнения программы OpenCL?
5. Какой функцией запускается программа на выполнении на устройстве?

Лабораторная работа №6.
Параллельные потоки OpenCL.

Цель работы: получить навыки создания и запуска параллельных программ OpenCL и организации двумерных NDRange.

Теоретические положения

Для исполнения параллельного кода ядра организуется N-мерное пространство индексов, которое называется NDRange, схема работы была показала в сведениях к работе №5 на рис. OpenCL поддерживает N, равное 1, 2 или 3. Таким образом, структура NDRange однозначно определяет адресацию с помощью индексов. Такая организация хорошо подходит для работы с многомерными данными.

Ход работы:

1. Составить программу с использованием OpenCL согласно варианту. При распараллеливании следует разбивать многомерные данные, используя NDRange необходимой размерности.
2. Рассчитать время выполнения OpenCL кода при разных размерах рабочей группы: максимальном, минимальном и среднем.

Варианты заданий

1. В матрице MxN, заполненной числами от 1 до 9 найти все возможные блоки 3x3 элемента, являющиеся магическими квадратами.
2. Дана матрица MxN, сгенерированная случайным образом, такая, что M и N кратно 8. Сформировать матрицу PxQ элементов, содержащую локальную дисперсию блоков
3. В матрице MxN определить сумму скалярных произведений пар соседних строк.
4. Имеется вещественная матрица размера MxN. Определить номер строки, в которой необходимо произвести максимальное количество перестановок для её сортировки по возрастанию.
5. Дана матрица размером MxN, где M - чётное. Определить максимальную сумму элементов чётных строк.
6. Дана матрица MxN натуральных чисел. Получить новую матрицу, в которой элемент равен 1, если соответствующий элемент исходной матрицы – простое число, и 0 в противном случае.
7. Дана матрица размером MxN. Определить строку, имеющую минимальную сумму элементов.
8. Дана матрица размером MxN, где N и M кратно 2. Каждая группа из 6 элементов может быть рассмотрена как координаты вершин треугольника. Если это так, найти треугольник с минимальной площадью. Группой считать 3 элемента i-ой строки и 3 элемента i+1-ой строки.
9. Дана матрица MxN, сгенерированная случайным образом и вектор из N элементов. Считая каждую строку матрицы вектором, найти такую, которая бы имела наименьшее скалярное произведение на вектор, заданный пользователем.
10. Дана матрица MxN, в которой каждая группа из 4x элементов представляет собой координаты прямоугольника на плоскости. Найти прямоугольник с наибольшим периметром. Группой считать 2 элемента i-ой строки и 2 элемента i+1-ой строки.
11. Считая строки матрицы размером MxN векторами N-мерного пространства, определить, какая из них расположена под наименьшим углом к вектору, введенному пользователем.

12. Дана матрица MxN, где где N кратно 6. Каждая группа из 6 элементов может быть рассмотрена как координаты вершин треугольника. Если это так, найти треугольник с минимальной площадью.
13. Дана матрица размером MxN элементов, где N кратно 10. Найти блок с минимальным средним значением.
14. Дана матрица размером MxN. Найдите блок матрицы размером 8x8, имеющий минимальную сумму элементов.
15. В матрице размером MxN, хранящей значения бит, сформировать N+1й столбец проверки на чётность.
16. К матрице размером MxN добавить N+1й столбец, содержащий суммы элементов строк. Найти строку с максимальной суммой.
17. Имеется матрица MxN, хранящая значения яркости пикселей изображения (полутоновое, 8 бит на пиксель). Выполнить линейную фильтрацию изображения фильтрами с ядрами

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \qquad \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

18. Матрица размером MxN содержит пары строк. Каждая пара задаёт координаты множества точек на плоскости. Определите два множества, расстояние между центрами масс которых минимально.
19. Даны два множества точек на плоскости. Определить точку второго множества, расстояние от которой до центра масс первого минимально.
20. Дана матрица вещественных значений MxN. Найти блок матрицы 4x4 с максимальным количеством отрицательных элементов.
21. Транспонировать квадратную матрицу.
22. Найдите в квадратной матрице такое i, что скалярное произведение i-го столбца на i-ю строку минимально.

Контрольные вопросы

1. Что такое «рабочая группа»? Как она определяется?
2. Что такое локальная и глобальная порция работы? Как они соотносятся?
3. Как определить набор данных, предназначенный для обработки внутри ядра OpenCL?
4. Какой бывает размерность NDrange и как её задать?
5. Каким образом можно записать данные на устройство и считать результаты?
6. Что такое «сэмплер»?

ЛИТЕРАТУРА

1. А. А. Малявко. Параллельное программирование на основе технологий openmp, cuda, opencl, mpi // ЮРАЙТ-Восток, 2021. – 136с.
2. В.П. Гергель, В. А. Фурсов. Лекции по параллельным вычислениям: учеб. пособие // Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2009. – 164с..
3. Р. В. Жалнин и др. Основы параллельного программирования с использованием технологий MPI и OpenMP: учебное пособие. // Саранск: Изд-во СВМО, 2013. – 78 с.
4. Р. Роби, Д. Замора. Параллельные и высокопроизводительные вычисления // ДМК Пресс, 2022. – 800 с.
5. Параллельные вычисления в УрО РАН. MPI для начинающих [Электронный ресурс] – Режим доступа: https://parallel.uran.ru/node/182 - Загл. с экрана.