ЗАДАНИЯ
ДЛЯ ПРАКТИЧЕСКИХ РАБОТ
и
МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ИХ ВЫПОЛНЕНИЮ

по дисциплине

"Распределенные информационные системы"

ОГЛАВЛЕНИЕ

1. Технология программирования OpenMP 5
Задание 1. Создание проекта в среде MS Visual Studio с поддержкой OpenMP 5
Задание 2. Многопоточная программа «Hello World!»..5
Задание 3. Программа «I am!»...5
Задание 4. Общие и частные переменные в OpenMP: программа «Скрытая ошибка» 6
Задание 5. Общие и частные переменные в OpenMP: параметр reduction 6
Задание 6. Распараллеливание циклов в OpenMP: программа «Сумма чисел» 7
Задание 7. Распараллеливание циклов в OpenMP: параметр schedule 7
Задание 8. Распараллеливание циклов в OpenMP: программа «Число π»..........................8
Задание 9. Распараллеливание циклов в OpenMP: программа «Матрица» 8
Задание 10. Параллельные секции в OpenMP: программа «I'm here» 9
Задание 11. Гонка потоков в OpenMP: программа «Сумма чисел» с atomic 10
Задание 12. Гонка потоков в OpenMP: программа «Число π» с critical 10
Задание 13. Исследование масштабируемости OpenMP-программ 10
2. Технология программирования MPI...12
Задание 14. Создание проекта в среде MS Visual Studio с поддержкой MPI 12
Задание 15. Программа «I am!»...12
Задание 16. Программа «На первый-второй рассчитайся!»..12
Задание 17. Коммуникации «точка-точка»: простые блокирующие обмены...................12
Задание 18. Коммуникации "точка-точка": схема «эстафетная палочка» 13
Задание 19. Коммуникации «точка-точка»: схема «мастер-рабочие» 13
Задание 20. Коммуникации «точка-точка»: простые неблокирующие обмены...............14
Задание 21. Коммуникации «точка-точка»: схема «сдвиг по кольцу»..............................14
Задание 22. Коммуникации «точка-точка»: схема «каждый каждому»...........................15
Задание 23. Коллективные коммуникации: широковещательная рассылка данных 16
Задание 24. Коллективные коммуникации: операции редукции 17
Задание 25. Коллективные коммуникации: функции распределения и сбора данных 17
Задание 26. Группы и коммуникаторы...18
Задание 27*. MPI-2: динамическое создание процессов 18
Задание 28*. MPI-2: односторонние коммуникации..19
Задание 29. Исследование масштабируемости MPI-программ..19
3. Технология программирования MPI+OpenMP 20
Задание 30. Проект в среде Visual Studio 2010 с поддержкой MPI и OpenMP.................20
Задание 31. Программа «I am» 20
Задание 32. Программа «Число π»..21
4. Методические указания 22
1. Технология программирования OpenMP 22

Указания к заданию 1. Создание проекта в среде MS Visual Studio с поддержкой OpenMP 22
Указания к заданию 2. Многопоточная программа «Hello World!» 25
Указания к заданию 3. Программа «I am!» 26
Указания к заданию 4. Общие и частные переменные в OpenMP: программа «Скрытая ошибка» 27
Указания к заданию 5. Общие и частные переменные в OpenMP: параметр reduction 27
Указания к заданию 6. Распараллеливание циклов в OpenMP: программа «Сумма чисел» 28
Указания к заданию 7. Распараллеливание циклов в OpenMP: параметр schedule 29
Указания к заданию 8. Распараллеливание циклов в OpenMP: программа «Число π» 30
Указания к заданию 9. Распараллеливание циклов в OpenMP: программа «Матрица» 30
Указания к заданию 10. Параллельные секции в OpenMP: программа «I'm here» 31
Указания к заданию 11. Гонка потоков в OpenMP: программа «Сумма чисел» с atomic 32
Указания к заданию 12. Гонка потоков в OpenMP: программа «Число π» с critical ... 32
Указания к заданию 13. Исследование масштабируемости OpenMP-программ 33
2. Технология программирования MPI 37
Указания к заданию 14. Создание проекта в среде MS Visual Studio с поддержкой MPI 37
Указания к заданию 15. Программа «I am!» 41
Указания к заданию 16. Программа «На первый-второй рассчитайся!» 43
Указания к заданию 17. Коммуникации «точка-точка»: простые блокирующие обмены 43
Указания к заданию 18. Коммуникации "точка-точка": схема «эстафетная палочка». 45
Указания к заданию 19. Коммуникации «точка-точка»: схема «мастер-рабочие» 46
Указания к заданию 20. Коммуникации «точка-точка»: простые неблокирующие обмены 46
Указания к заданию 21. Коммуникации «точка-точка»: схема «сдвиг по кольцу» 47
Указания к заданию 22. Коммуникации «точка-точка»: схема «каждый каждому» 48
Указания к заданию 23. Коллективные коммуникации: широковещательная рассылка данных 49
Указания к заданию 24. Коллективные коммуникации: операции редукции 50
Указания к заданию 25. Коллективные коммуникации: функции распределения и сбора данных 51
Указания к заданию 26. Группы и коммуникаторы 52
Указания к заданию 27. MPI-2: динамическое создание процессов 55
Указания к заданию 28. MPI-2: односторонние коммуникации 55
Указания к заданию 29. Исследование масштабируемости MPI-программ 57
3. Технология программирования MPI+OpenMP 57

Указания к заданию 30. Проект в среде Visual Studio 2010 с поддержкой MPI и OpenMP 57
Указания к заданию 31. Программа «I am» 57
Указания к заданию 31. Программа «I am» 58

1. Технология программирования OpenMP

Задание 1. Создание проекта в среде MS Visual Studio с поддержкой OpenMP

Создайте проект в среде MS Visual Studio 2010 с поддержкой OpenMP.

Задание 2. Многопоточная программа «Hello World!»

Напишите OpenMP-программу, в которой создается 4 нити и каждая нить выводит на экран строку «`Hello World!`».

Входные данные: нет.

Выходные данные: 4-е строки «`Hello World!`».

Пример входных и выходных данных

Входные данные	Выходные данные
	`Hello World!` `Hello World!` `Hello World!` `Hello World!`

Задание 3. Программа «I am!»

1. Напишите программу, в которой создается k нитей, и каждая нить выводит на экран свой номер и общее количество нитей в параллельной области в формате:

`I am <Номер нити> thread from <Количество нитей> threads!`

Входные данные: k – количество нитей в параллельной области.

Выходные данные: k строк вида «`I am <Номер нити> thread from <Количество нитей> threads!`».

Пример входных и выходных данных

Входные данные	Выходные данные

3	`I am 0 thread from 3 threads!`
	`I am 1 thread from 3 threads!`
	`I am 2 thread from 3 threads!`

2. Модифицируйте программу таким образом, чтобы строку `I am <Номер нити> thread from <Количество нитей> threads!` выводили только нити с четным номером.

Пример входных и выходных данных

Входные данные	Выходные данные
3	`I am 0 thread from 3 threads!` `I am 2 thread from 3 threads!`

Задание 4. Общие и частные переменные в OpenMP: программа «Скрытая ошибка»

Изучите конструкции для управления работой с данными shared и private. Напишите программу, в которой создается k нитей, и каждая нить выводит на экран свой номер через переменную rank следующим образом:

```
rank = omp_get_thread_num();
printf("I am %d thread.\n", rank);
```

Экспериментами определите, общей или частной должна быть переменная rank.

Входные данные: целое число k – количество нитей в параллельной области.

Выходные данные: k строк вида «I am <Номер нити>.».

Пример входных и выходных данных

Входные данные	Выходные данные
3	I am 0 thread. I am 1 thread. I am 2 thread.

Задание 5. Общие и частные переменные в OpenMP: параметр reduction

1. Напишите программу, в которой две нити параллельно вычисляют сумму чисел от 1 до N. Распределите работу по нитям с помощью оператора if языка C. Для сложения результатов вычисления нитей воспользуйтесь OpenMP-параметром reduction.

Входные данные: целое число N – количество чисел.

Выходные данные: каждая нить выводит свою частичную сумму в формате «[Номер_нити]: Sum = <частичная_сумма>», один раз выводится общая сумма в формате «Sum = <сумма>».

Пример входных и выходных данных

Входные данные	Выходные данные
4	[0]: Sum = 3 [1]: Sum = 7 Sum = 10

2*. Модифицируйте программу таким образом, чтобы она работала для k нитей.

Входные данные: целое число k – количество нитей, целое число N – количество чисел.

Выходные данные: каждая нить выводит свою частичную сумму в формате «[Номер_нити]: Sum = <частичная_сумма>», один раз выводится общая сумма в формате «Sum = <сумма>».

Пример входных и выходных данных

Входные данные	Выходные данные
2 4	[0]: Sum = 3 [1]: Sum = 7 Sum = 10
2 2	[0]: Sum = 1 [1]: Sum = 2 Sum = 3
3 2	[0]: Sum = 1 [1]: Sum = 2 [2]: Sum = 0 Sum = 3

Задание 6. Распараллеливание циклов в OpenMP: программа «Сумма чисел»

Изучите OpenMP-директиву параллельного выполнения цикла for. Напишите программу, в которой k нитей параллельно вычисляют сумму чисел от 1 до N. Распределите работу по нитям с помощью OpenMP-директивы for.

Входные данные: целое число k – количество нитей, целое число N – количество чисел.

Выходные данные: каждая нить выводит свою частичную сумму в формате «[Номер_нити]: Sum = <частичная_сумма>», один раз выводится общая сумма в формате «Sum = <сумма>».

Пример входных и выходных данных

Входные данные	Выходные данные
2 4	[0]: Sum = 3 [1]: Sum = 7 Sum = 10
2 2	[0]: Sum = 1 [1]: Sum = 2 Sum = 3
3 2	[0]: Sum = 1 [1]: Sum = 2 [2]: Sum = 0 Sum = 3

Задание 7. Распараллеливание циклов в OpenMP: параметр schedule

Изучите параметр schedule директивы for. Модифицируйте программу «Сумма чисел» из задания 6 таким образом, чтобы дополнительно выводилось на экран сообщение о том, какая нить, какую итерацию цикла выполняет:

[<Номер нити>]: calculation of the iteration number <Номер итерации>.

Задайте k = 4, N = 10. Заполните следующую таблицу распределения итераций цикла по нитям в зависимости от параметра schedule:

Номер итерации	Значение параметра schedule						
	static	static, 1	static, 2	dynamic	dynamic, 2	guided	guided, 2
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

Задание 8. Распараллеливание циклов в OpenMP: программа «Число π»

1. Напишите OpenMP-программу, которая вычисляет число π с точностью до N знаков после запятой. Используйте следующую формулу:

$$\pi = \left(\frac{4}{1+x_0^2} + \frac{4}{1+x_1^2} + \ldots + \frac{4}{1+x_{N-1}^2} \right) \times \frac{1}{N}, \text{ где } x_i = (i+0.5) \times \frac{1}{N}, \ i = \overline{0, N-1}$$

Распределите работу по нитям с помощью OpenMP-директивы for.

Входные данные: одно целое число N (точность вычисления).

Выходные данные: одно вещественное число pi.

Пример входных и выходных данных

Входные данные	Выходные данные
1000000000	3.14159265

Задание 9. Распараллеливание циклов в OpenMP: программа «Матрица»

Напишите OpenMP-программу, которая вычисляет произведение двух квадратных матриц $A \times B = C$ размера $n \times n$. Используйте следующую формулу:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ . & . & . & \ldots & . \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \ldots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \ldots & b_{2n} \\ . & . & . & \ldots & . \\ b_{n1} & b_{n2} & b_{n3} & \ldots & b_{nn} \end{pmatrix}$$

$$c_{im} = \sum_{j=1}^{n} a_{ij} \cdot b_{jm}; i = 1,2, \ldots, n; \ m = 1,2, \ldots, n$$

$$C = \begin{pmatrix} \sum_{j=1}^{n} a_{1j} \cdot b_{j1} & \sum_{j=1}^{n} a_{1j} \cdot b_{j2} & \sum_{j=1}^{n} a_{1j} \cdot b_{j3} & \dots & \sum_{j=1}^{n} a_{1j} \cdot b_{jn} \\ \sum_{j=1}^{n} a_{2j} \cdot b_{j1} & \sum_{j=1}^{n} a_{2j} \cdot b_{j2} & \sum_{j=1}^{n} a_{2j} \cdot b_{j3} & \dots & \sum_{j=1}^{n} a_{2j} \cdot b_{jn} \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{j=1}^{n} a_{nj} \cdot b_{j1} & \sum_{j=1}^{n} a_{nj} \cdot b_{j2} & \sum_{j=1}^{n} a_{nj} \cdot b_{j3} & \dots & \sum_{j=1}^{n} a_{nj} \cdot b_{jn} \end{pmatrix}$$

Входные данные: целое число n, $1 \le n \le 10$, n^2 вещественных элементов матрицы A и n^2 вещественных элементов матрицы B.

Выходные данные: n^2 вещественных элементов матрицы C.

Пример входных и выходных данных

Входные данные	Выходные данные
2 1 3 4 8 5 4 3 0	14 4 44 16

Задание 10. Параллельные секции в OpenMP: программа «I'm here»

Изучите OpenMP-директивы создания параллельных секций sections и section. Напишите программу, содержащую 3 параллельные секции, внутри каждой из которых должно выводиться сообщение:

```
[<Номер нити>]: came in section <Номер секции>
```

Вне секций внутри параллельной области должно выводиться следующее сообщение:

```
[<Номер нити>]: parallel region
```

Запустите приложение на 2-х, 3-х, 4-х нитях. Проследите, как нити распределяются по параллельным секциям.

Входные данные: k – количество нитей в параллельной области.

Выходные данные: k-строк вида «`[<Номер нити>]: came in section <Номер секции>`», k-строк вида «`[<Номер нити>]: parallel region`».

Пример входных и выходных данных

Входные данные	Выходные данные
3	[0]: came in section 1 [1]: came in section 2 [2]: came in section 3 [0]: parallel region [1]: parallel region [2]: parallel region

Задание 11. Гонка потоков[1] в OpenMP: программа «Сумма чисел» с atomic

Перепишите программу, в которой параллельно вычисляется сумма чисел от 1 до N (см. задание 6), без использования параметра reduction. Вместо параметра reduction используйте директиву atomic.

Задание 12. Гонка потоков в OpenMP: программа «Число π» с critical

Перепишите параллельную программу вычисления числа π (см. задание 8) без использования параметра reduction. Вместо параметра reduction используйте директиву critical.

Задание 13. Исследование масштабируемости OpenMP-программ

1. Проведите серию экспериментов на персональном компьютере по исследованию масштабируемости OpenMP-программ. Заполните следующую таблицу:

№ п/п	Программа	Параметр N	Количество нитей	Время выполнения (сек.)
1	«Матрица» (см. задание 9)	100	1	
2		1 000 000	1	
3		100	2	
4		1 000 000	2	
5		100	4	
6		1 000 000	4	
7		100	6	
8		1 000 000	6	
9		100	8	
10		1 000 000	8	
11		100	10	
12		1 000 000	10	
13		100	12	
14		1 000 000	12	

На основании данных таблицы постройте график масштабируемости для каждого значения параметра N. Определите для каждого графика, при каком количестве нитей достигается максимальное ускорение.

2. Проведите серию экспериментов на суперкомпьютере по исследованию масштабируемости OpenMP-программ. Заполните следующую таблицу:

---

[1] ***Гонка потоков (race conditions)*** – ситуация когда результат вычислений зависит от темпа выполнения программы нитями. Для исключения гонки необходимо обеспечить, чтобы изменение значений общих переменных осуществлялось в каждый момент времени только одним единственным потоком. В OpenMP это может быть организовано следующими основными механизмами:

- неделимые (atomic) операции,
- механизм критических секций (critical sections).

№ п/п	Программа	Параметр N	Количество нитей	Время выполнения (сек.)
15	«Число π» (см. задание 8)	100	1	
16		10 000 000	1	
17		4 000 000 000	1	
18		100	2	
19		10 000 000	2	
20		4 000 000 000	2	
21		100	4	
22		10 000 000	4	
23		4 000 000 000	4	
24		100	6	
25		10 000 000	6	
26		4 000 000 000	6	
27		100	8	
28		10 000 000	8	
29		4 000 000 000	8	
30		100	10	
31		10 000 000	10	
32		4 000 000 000	10	
33		100	12	
34		10 000 000	12	
35		4 000 000 000	12	
36	«Число π» с critical (см. задание 12)	100	1	
37		10 000 000	1	
38		4 000 000 000	1	
39		100	2	
40		10 000 000	2	
41		4 000 000 000	2	
42		100	4	
43		10 000 000	4	
44		4 000 000 000	4	
45		100	6	
46		10 000 000	6	
47		4 000 000 000	6	
48		100	8	
49		10 000 000	8	
50		4 000 000 000	8	
51		100	10	
52		10 000 000	10	
53		4 000 000 000	10	
54		100	12	
55		10 000 000	12	
56		4 000 000 000	12	

На основании данных таблицы постройте график масштабируемости для каждого значения параметра N. Определите для каждого графика, при каком количестве нитей достигается максимальное ускорение.

2. Технология программирования MPI

Задание 14. Создание проекта в среде MS Visual Studio с поддержкой MPI

Создайте проект в среде Visual Studio 2010 с поддержкой MPI.

Задание 15. Программа «I am!»

Напишите программу, в которой каждый процесс выводит на экран свой номер и общее количество процессов в приложении в формате:

```
I am <Номер процесса> process from <Количество процессов>
processes!
```

Входные данные: нет.

Выходные данные: строки в формате «I am <Номер процесса> process from <Количество процессов> processes!».

Пример входных и выходных данных

Входные данные	Выходные данные
3	I am 0 process from 3 processes! I am 1 process from 3 processes! I am 2 process from 3 processes!

Задание 16. Программа «На первый-второй рассчитайся!»

Напишите программу, в которой каждый процесс с четным номером выводит на экран строку «I am <Номер процесса>: FIRST!», а каждый процесс с нечетным номером – «I am <Номер процесса>: SECOND!». Процесс с номером 0 должен вывести на экран общее количество процессов в приложении в формате «<Количество процессов> processes.».

Входные данные: нет.

Выходные данные: строки в формате «I am <Номер процесса>: FIRST!» или «I am <Номер процесса>: SECOND!» или «<Количество процессов> processes.».

Пример входных и выходных данных

Входные данные	Выходные данные
4	4 processes. I am 1 process: FIRST! I am 2 process: SECOND! I am 3 process: FIRST!

Задание 17. Коммуникации «точка-точка»: простые блокирующие обмены

Изучите основные MPI-функции блокирующей передачи сообщений точка-точка MPI_Send и MPI_Recv. Напишите MPI-программу, в которой с помощью данных функций процесс с номером 0 отправляет сообщение процессу с номером 1. Процесс 1 выводит полученное сообщение на экран.

Входные данные: нет.

Выходные данные: «`receive message '<сообщение>'`».

Пример входных и выходных данных

Входные данные	Выходные данные
	`receive message '45'`

Задание 18. Коммуникации «точка-точка»: схема «эстафетная палочка»

Напишите MPI-программу, реализующую при помощи блокирующих функций посылки сообщений типа точка-точка схему коммуникации процессов «эстафетная палочка», в которой каждый процесс дожидается сообщения от предыдущего и потом посылает следующему (см. рис. 1). В качестве передаваемого сообщения используйте на процессе 0 его номер, на остальных процессах – инкрементированное полученное сообщение.

Рис. 1. Схема коммуникации процессов «эстафетная палочка»

Входные данные: нет.

Выходные данные: «`[<номер_процесса>]: receive message '<сообщение>'`».

Пример входных и выходных данных для 4-х процессов

Входные данные	Выходные данные

	`[0]: receive message '3'`
	`[1]: receive message '0'`
	`[2]: receive message '1'`
	`[3]: receive message '2'`

Задание 19. Коммуникации «точка-точка»: схема «мастер-рабочие»

Напишите MPI-программу, реализующую при помощи блокирующих функций посылки сообщений типа точка-точка схему коммуникации процессов «master-slave», в которой один процесс, называемый master[2], принимает

[2] Master-процессом может быть любой произвольный процесс, обычно это процесс с номером 0.

сообщение от остальных процессов, называемых slave (см. рис. 2). В качестве передаваемого сообщения используйте номер процесса. Master-процесс должен вывести на экран все полученные сообщения.

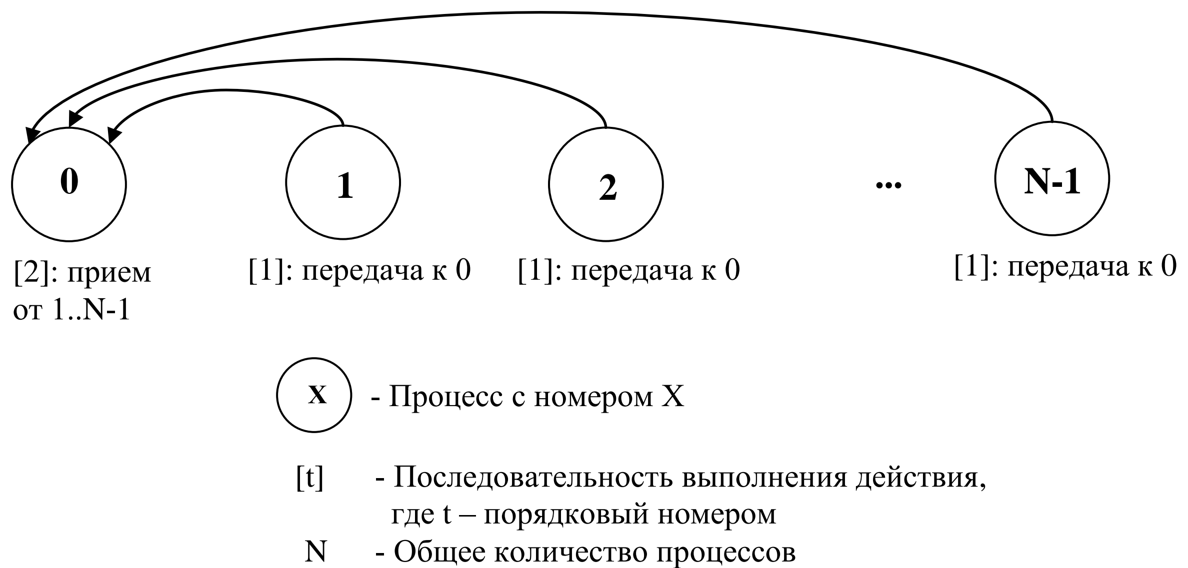


Рис. 2. Схема коммуникации процессов «master-slave»

Входные данные: нет.

Выходные данные: «`receive message '<сообщение>' from <номер_процесса>`».

Пример входных и выходных данных для 4-х процессов

Входные данные	Выходные данные

	`receive message '1'`
	`receive message '2'`
	`receive message '3'`

Задание 20. Коммуникации «точка-точка»: простые неблокирующие обмены

Изучите основные MPI-функции неблокирующей передачи сообщений точка-точка MPI_Isend, MPI_Irecv, MPI_Wait. Напишите MPI-программу, в которой с помощью данных функций процесс с номером 0 отправляет сообщение процессу с номером 1. Процесс 1 выводит полученное сообщение на экран.

Входные данные: нет.

Выходные данные: «`receive message '<сообщение>'`».

Пример входных и выходных данных

Входные данные	Выходные данные
	`receive message '45'`

Задание 21. Коммуникации «точка-точка»: схема «сдвиг по кольцу»

Напишите MPI-программу, реализующую при помощи блокирующих

функций посылки сообщений типа точка-точка схему коммуникации процессов «сдвиг по кольцу», в которой осуществляются одновременные посылка и прием сообщений всеми процессами (см. рис. 3). В качестве передаваемого сообщения используйте номер процесса.

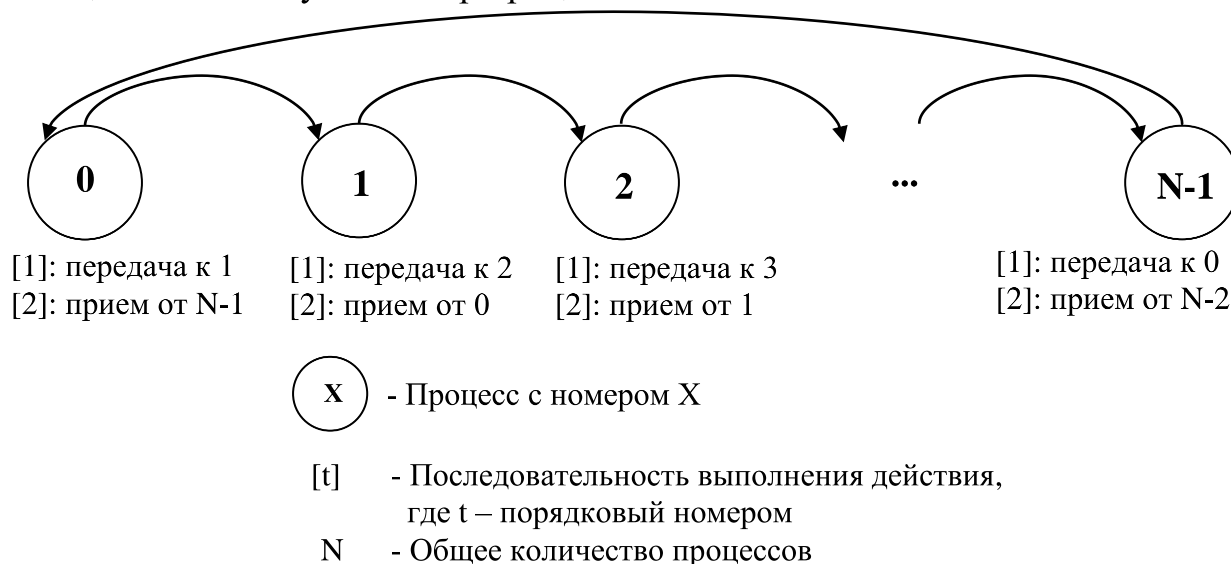


Рис. 3. Схема коммуникации процессов «сдвиг по кольцу»

Входные данные: нет.

Выходные данные: «`[<номер_процесса>]: receive message '<сообщение>'`».

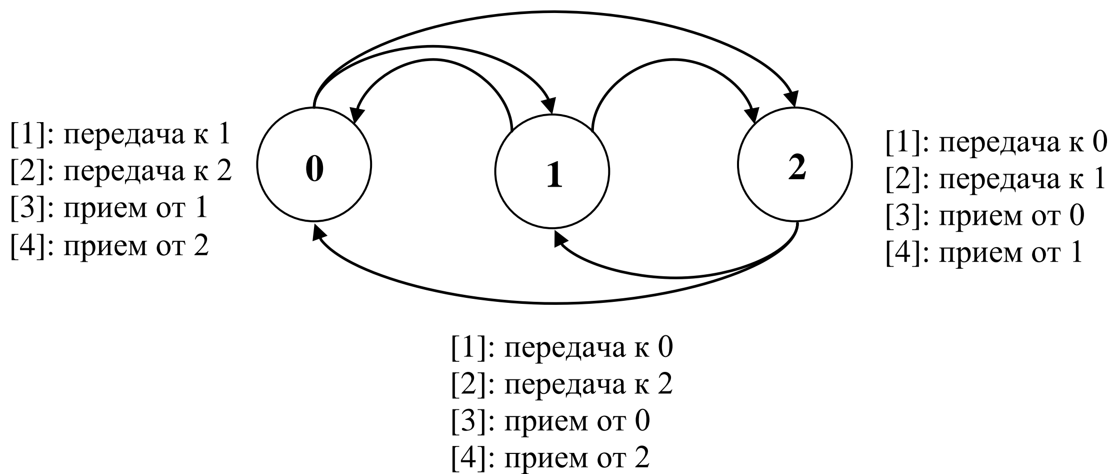
Пример входных и выходных данных для 4-х процессов

Входные данные	Выходные данные

	`[0]: receive message '3'`
	`[1]: receive message '0'`
	`[2]: receive message '1'`
	`[3]: receive message '2'`

Задание 22. Коммуникации «точка-точка»: схема «каждый каждому»

Напишите MPI-программу, реализующую при помощи блокирующих функций посылки сообщений типа точка-точка схему коммуникации процессов «каждый каждому», в которой осуществляется пересылка сообщения от каждого процесса каждому (см. рис. 4). В качестве передаваемого сообщения используйте номер процесса. Каждый процесс должен вывести на экран все полученные сообщения.



X - Процесс с номером X

[t] - Последовательность выполнения действия, где t – порядковый номером

Рис. 4. Схема коммуникации процессов «каждый каждому» на примере 3-х процессов

Входные данные: нет.

Выходные данные: каждый процесс выводит сообщение «`[<номер_процесса>]: receive message '<сообщение>' from <номер_процесса>`».

Пример входных и выходных данных для 3-х процессов

Входные данные	Выходные данные

	`[0]: receive message '1' from 1`
	`[0]: receive message '2' from 2`
	`[1]: receive message '0' from 0`
	`[1]: receive message '2' from 2`
	`[2]: receive message '0' from 0`
	`[2]: receive message '1' from 1`

Задание 23. Коллективные коммуникации: широковещательная рассылка данных

1. Изучите MPI-функцию широковещательной рассылки данных MPI_Bcast. Напишите MPI-программу, которая в строке длины n определяет количество вхождений символов. Ввод данных должен осуществляться процессом с номером 0. Для рассылки строки поиска и ее длины по процессам используйте функцию MPI_Bcast.

2*. Перепишите программу, используя вместо функции MPI_Bcast функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: целое число n ($1 \leq n \leq 100$), строка из n символов (каждый символ в строке может представлять собой только строчную букву английского алфавита).

Выходные данные: количество вхождений всех символов, имеющихся в строке в формате «`<буква> =<значение>`».

Пример входных и выходных данных

Входные данные	Выходные данные
9 aaaaaaaaa	a = 9
3 rvtabc	a = 1 b = 1 c = 1 r = 1 t = 1 v = 1

Задание 24. Коллективные коммуникации: операции редукции

1. Изучите MPI-функцию для выполнения операций редукции над данными, расположенными в адресных пространствах различных процессов, MPI_Reduce. Реализуйте программу вычисления числа π (см. задание 8), используйте функцию MPI_Reduce для суммирования результатов, вычисленных каждым процессом.

2*. Перепишите программу, используя вместо функции MPI_Reduce функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: одно целое число N (точность вычисления).

Выходные данные: одно вещественное число pi.

Пример входных и выходных данных

Входные данные	Выходные данные
1000000000	3.14159265

Задание 25. Коллективные коммуникации: функции распределения и сбора данных

1. Изучите MPI-функции распределения и сбора блоков данных по процессам MPI_Scatter и MPI_Gather. Напишите программу, которая вычисляет произведение двух квадратных матриц $A \times B = \mathrm{C}$ размера $n \times n$. Используйте формулу, приведенную в задании 9. Ввод данных и вывод результата должны осуществляться процессом с номером 0. Для распределения матриц A и B и сбора матрицы C используйте функций MPI_Scatter и MPI_Gather.

2*. Перепишите программу, используя вместо функций MPI_Scatter и MPI_Gather функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: целое число *n,* $1 \leq n \leq 10$, n^2 вещественных элементов матрицы *A* и n^2 вещественных элементов матрицы *B*.

Выходные данные: n^2 вещественных элементов матрицы *C*.

Пример входных и выходных данных

Входные данные	Выходные данные
2 1 3 4 8 5 4 3 0	14 4 44 16

Задание 26. Группы и коммуникаторы

Напишите программу, в которой производится широковещательная рассылка сообщения с помощью функции MPI_Bcast, но только по процессам с четным номером. Для рассылки сообщения создайте новый коммуникатор. Каждый процесс приложения должен выводить на экран «MPI_COMM_WORLD: <номер процесса в коммуникаторе MPI_COMM_WORLD> from <количество процессов в коммуникаторе MPI_COMM_WORLD>. New comm: <номер процесса в новом коммуникаторе> from <количество процессов в новом коммуникаторе>. Message = <сообщение>»

Входные данные: message – строка с сообщением, считываемым только процессом 0, количество символов в message от 1 до 10.

Выходные данные: строки вида «MPI_COMM_WORLD: <номер процесса в коммуникаторе MPI_COMM_WORLD> from <количество процессов в коммуникаторе MPI_COMM_WORLD>. New comm: <номер процесса в новом коммуникаторе> from <количество процессов в новом коммуникаторе>. Message = <сообщение>».

Пример входных и выходных данных для 3 процессов

Входные данные	Выходные данные
A	MPI_COMM_WORLD: 0 from 3. New comm: 0 from 2. Message = A MPI_COMM_WORLD: 1 from 3. New comm: no from no. Message = no MPI_COMM_WORLD: 2 from 3. New comm: 1 from 2. Message = A

Задание 27. MPI-2: динамическое создание процессов*

Напишите программу, в которой нулевым процессом динамически запускается еще n процессов. Каждый процесс в программе выводит сообще-

ние в формате «`I am <Номер процесса> process from <Количество процессов> processes! My parent is <Номер процесса родителя>`».

Входные данные: целое число n – количество процессов, которые должны быть запущены динамически.

Выходные данные: строки вида «`I am <Номер процесса> process from <Количество процессов> processes! My parent is <Номер процесса родителя>`».

Пример входных и выходных данных для 2-х процессов

Входные данные	Выходные данные

`3`	`I am 0 process from 2 processes!`
	`My parent is none.`
	`I am 1 process from 2 processes!`
	`My parent is none.`
	`I am 0 process from 3 processes!`
	`My parent is 0.`
	`I am 1 process from 3 processes!`
	`My parent is 0.`
	`I am 2 process from 3 processes!`
	`My parent is 0.`

Задание 28. MPI-2: односторонние коммуникации*

Реализуйте программу вычисления числа π (см. задание 24), используйте функции односторонней коммуникации для обмена данными между процессами.

Входные данные: одно целое число N (точность вычисления).

Выходные данные: одно вещественное число pi.

Пример входных и выходных данных

Входные данные	Выходные данные
`1000000000`	`3.14159265`

Задание 29. Исследование масштабируемости MPI-программ

Проведите серию экспериментов на суперкомпьютере по исследованию масштабируемости OpenMP-программ. Заполните следующую таблицу:

№ п/п	Программа	Параметр N	Количество нитей	Время выполнения (сек.)
57	«Число π» (см. задание 24)	100	1	
58		10 000 000	1	
59		4 000 000 000	1	
60		100	2	
61		10 000 000	2	
62		4 000 000 000	2	
63		100	4	
64		10 000 000	4	
65		4 000 000 000	4	

66		100	6	
67		10 000 000	6	
68		4 000 000 000	6	
69		100	8	
70		10 000 000	8	
71		4 000 000 000	8	
72		100	10	
73		10 000 000	10	
74		4 000 000 000	10	
75		100	12	
76		10 000 000	12	
77		4 000 000 000	12	

На основании данных таблицы постройте график масштабируемости для каждого значения параметра N. Определите для каждого графика, при каком количестве нитей достигается максимальное ускорение.

3. Технология программирования MPI+OpenMP

Задание 30. Проект в среде Visual Studio 2010 с поддержкой MPI и OpenMP

Создайте проект в среде Visual Studio 2010 с минимальным кодом с поддержкой MPI и OpenMP.

Задание 31. Программа «I am»

Напишите программу, в которой в каждом процессе создается n нитей. Каждая нить должна выводить на экран свой номер, номер процесса-родителя и общее количество нитей во всех процессах в следующем формате: I am <Номер нити> thread from <Номер родительского процесса> process. Number of hybrid threads = <Количество нитей * Количество процессов>.

Входные данные: целое число n – количество нитей, которые должны быть запущены.

Выходные данные: строки вида «I am <Номер нити> thread from <Номер родительского процесса> process. Number of hybrid threads = <Количество нитей * Количество процессов>».

Пример входных и выходных данных для 2-х процессов

Входные данные	Выходные данные
3	I am 0 thread from 0 process. Number of hybrid threads = 6 I am 1 thread from 0 process. Number of hybrid threads = 6 I am 2 thread from 0 process. Number of hybrid threads = 6 I am 0 thread from 1 process. Number of hybrid threads = 6

	I am 1 thread from 1 process. Number of hybrid threads = 6 I am 2 thread from 1 process. Number of hybrid threads = 6

Задание 32. Программа «Число π»

Реализуйте программу вычисления числа π (см. задание 8) с использованием MPI+OpenMP.

Входные данные: одно целое число N (точность вычисления).

Выходные данные: одно вещественное число pi.

Пример входных и выходных данных для 2-х процессов

Входные данные	Выходные данные
1000000000	3.14159265

4. Методические указания

1. Технология программирования OpenMP

Указания к заданию 1. Создание проекта в среде MS Visual Studio с поддержкой OpenMP

1. Создайте на рабочем столе папку с вашей фамилией.
2. Запустите Microsoft Visual Studio 2010. **Внимание!** При первом запуске Visual Studio выберите интерфейс по умолчанию «Параметры разработки Visual C++».
3. ***Создание проекта***. Для этого выберите пункт в меню *File -> New -> Project*, или нажмите Ctrl+Shift+N
4. В окне *New Project* в раскрывающемся списке *Visual C++* выбирите *Win32*. В подокне в середине выберите *Win32 Console Application*. Внизу введите имя проекта *Name* (например, example1), и место расположения проекта *Location* (укажите папку с вашей фамилией на рабочем столе), и нажмите кнопку *OK*. См. рис. 3.1.

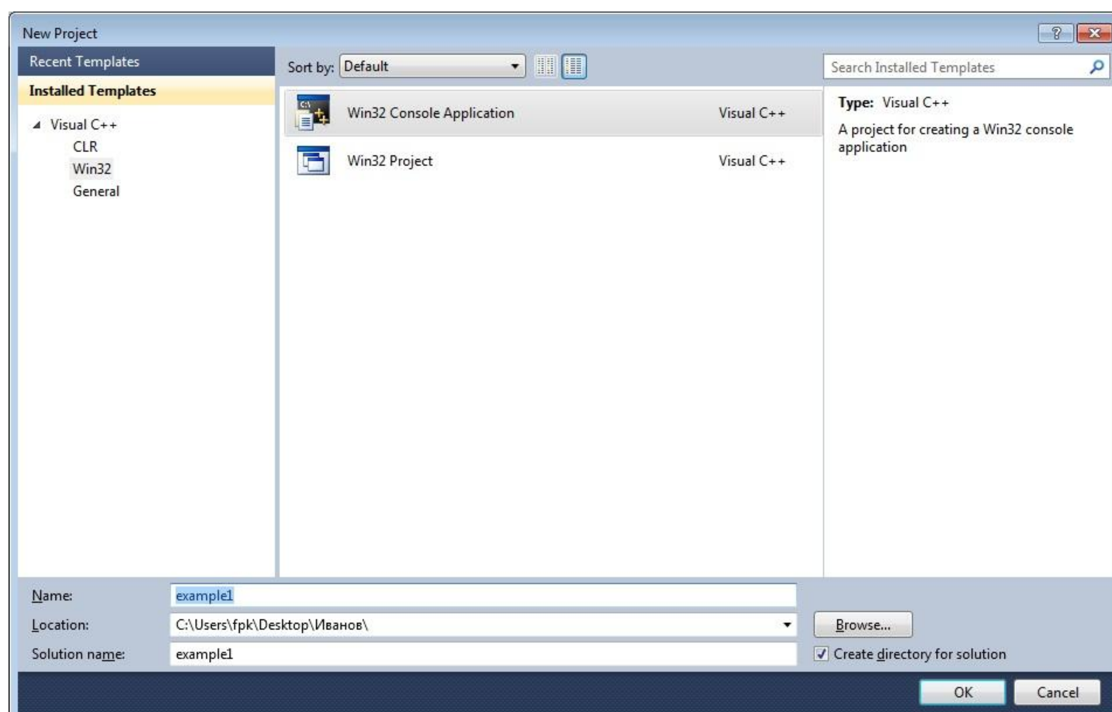


Рис. 3.1

5. В открывшемся окне *Win32 Application Wizard - example1* нажмите кнопку *Next*, и затем в *Additional options* поставьте галочку напротив *Empty project*. Нажмите кнопку *Finish*. См. рис. 3.2.

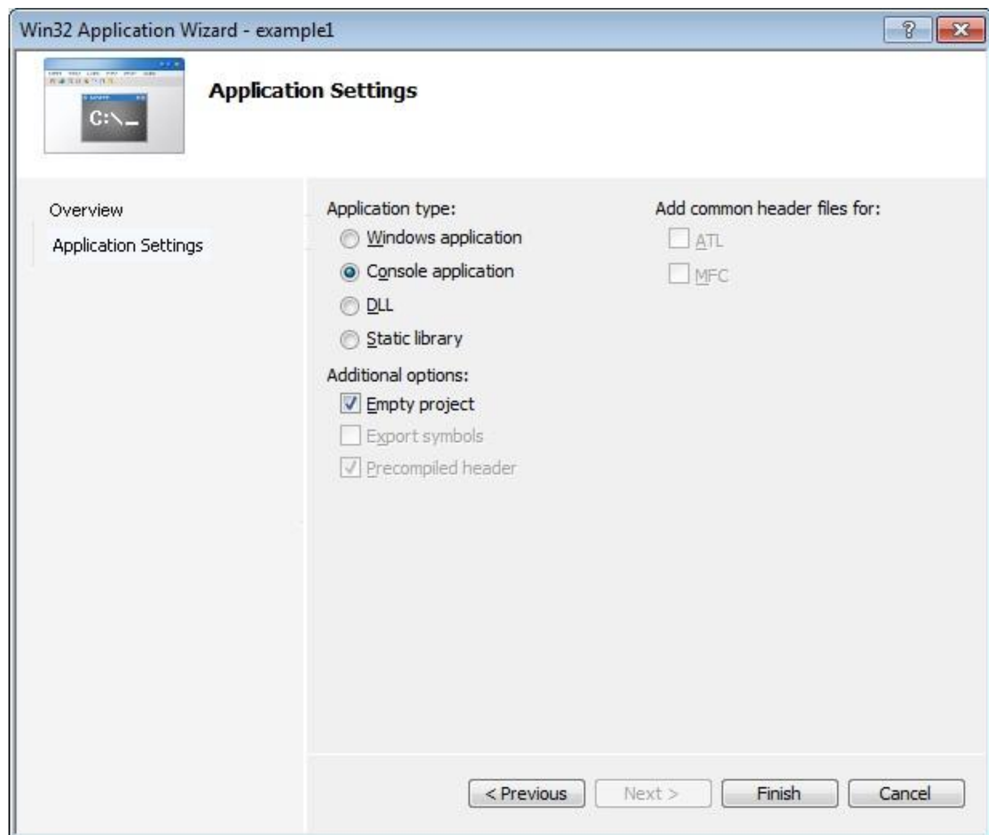


Рис. 3.2

6. Теперь ***создадим файл с кодом приложения***. Выберите пункт в меню *Project*
   -> *Add New Item*, или нажмите Ctrl+Shift+A. В категории *Visual C++* выберите подкатегорию *Code*. В подокне в середине установите *C++ File (.cpp)*. Введите имя файла, например, source, и нажмите кнопку *Add*. См. рис. 3.3.

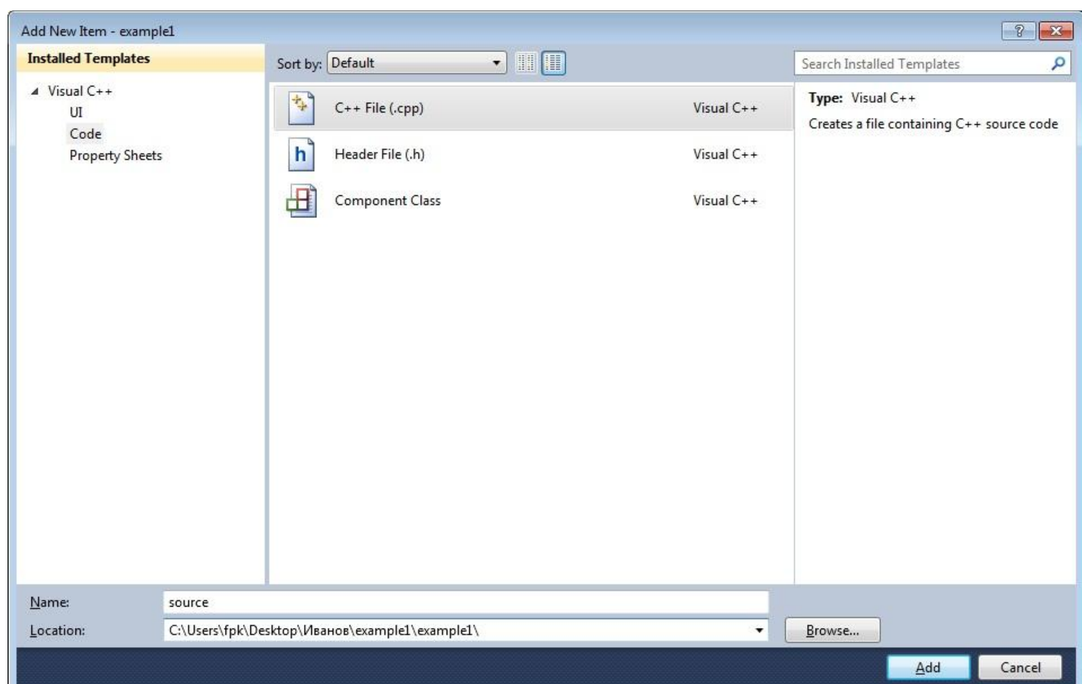


Рис. 3.3

7. В открывшемся окне *source.cpp* введите следующий код на языке С:

```
int main() {
  return 0;
}
```

   Сохраните файл, выбрав пункт меню *File -> Save source.cpp*, или нажав Ctrl+S.

8. ***Для компиляции*** приложения выберите пункт меню *Debug -> Build Solution*, или нажмите F7.
9. ***Для запуска*** приложения выберите пункт меню *Debug -> Start Without Debugging*, или нажмите Ctrl+F5. См. рис. 3.4.

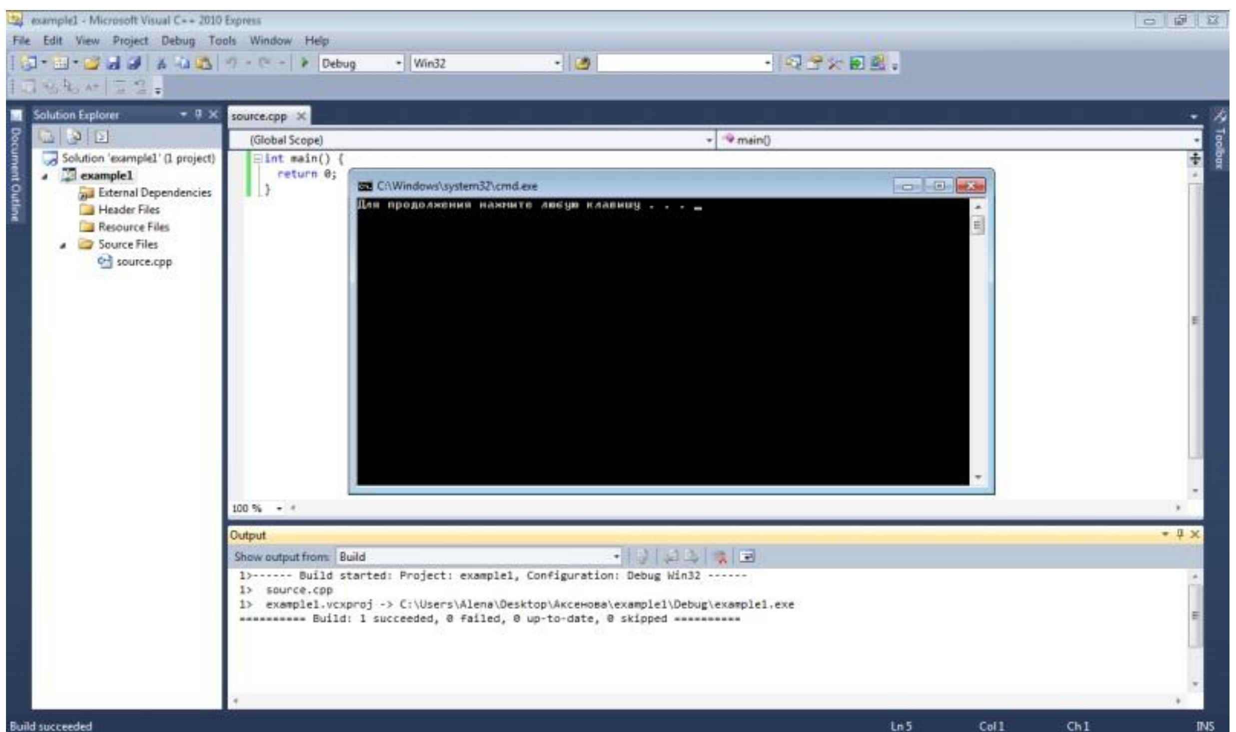


Рис. 3.4

10. ***Для включения поддержки OpenMP*** установите дополнительные параметры компиляции проекта:

    - В главном меню выберите *Project-> Имя_проекта Properties*
    - В открывшемся окне выберите *Configuration Properties / C/C++ / Language*. Установите для опции *OpenMP Support* значение *Yes (/openmp)*. Нажмите кнопку *OK*. См. рис. 3.5.

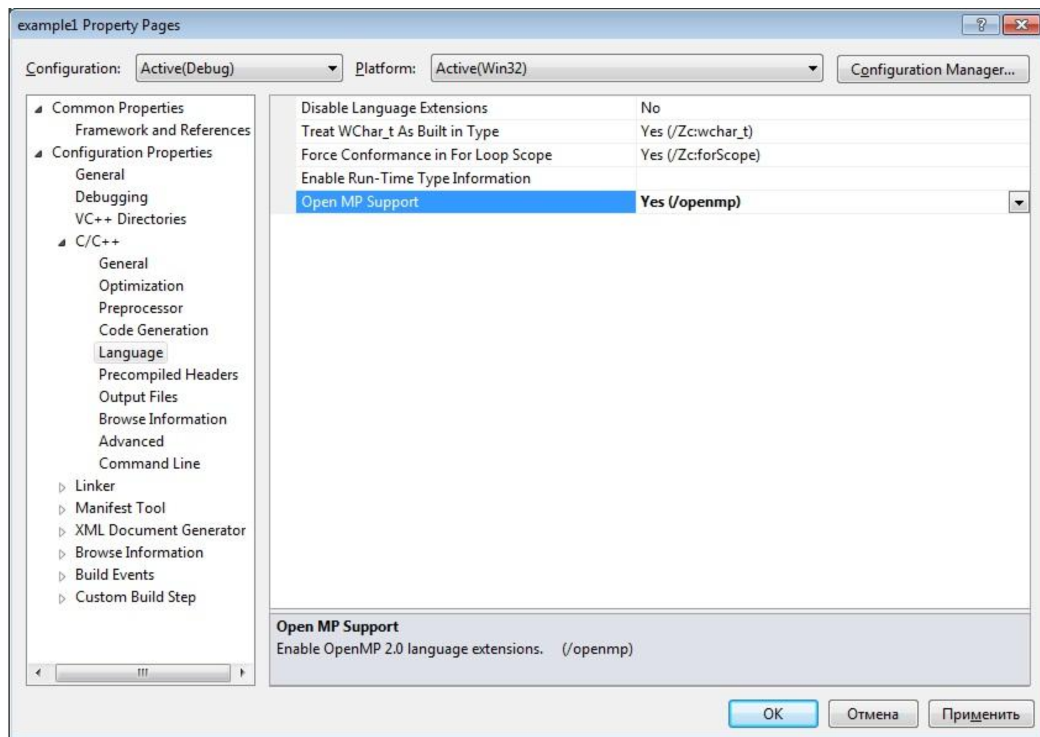


Рис. 3.5

11. ***Для компиляции*** приложения нажмите F7.
12. ***Для запуска*** приложения нажмите Ctrl+F5.

Указания к заданию 2. Многопоточная программа «Hello World!»

1. Создайте проект omp_intro в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. Напишите программу, печатающую на экран строку «Hello World!».
3. Подключите заголовочный файл omp.h с функциями и переменными OpenMP. Строка подключения заголовочного файла:

```
#include <omp.h>
```

4. В функции main создайте параллельную область с помощью OpenMP-директивы parallel. **Обратите внимание**, что открывающаяся фигурная скобка и название директивы должны находиться в разных строках! Поместите команду вывода строки «Hello World!» внутрь параллельной области.

```
#pragma omp parallel
{
  printf("Hello World!\n");
}
```

5. Задайте количество нитей в параллельной области одним из следующих способов:

   Способ 1. Вызовите функцию omp_set_num_threads() перед началом па-

раллельной области. В качестве параметра укажите одно целое число – количество нитей в параллельной области:

```
omp_set_num_threads(4);
#pragma omp parallel
{
   printf("Hello World!\n");
}
```

Способ 2. Добавьте к директиве parallel параметр num_threads(). В качестве параметра укажите одно целое число – количество нитей в параллельной области:

```
#pragma omp parallel num_threads(4)
{
   printf("Hello World!\n");
}
```

6. Скомпилируйте и запустите ваше приложение. Убедитесь, что строка «Hello World!» выводится на экран столько раз, сколько нитей вы задали в параллельной области.

Указания к заданию 3. Программа «I am!»

1. Откройте проект omp_intro в Microsoft Visual Studio 2010 (см. указания к заданию 2).
2. Определите параметр k. В параллельной области функции main задайте k нитей.
3. Для получения номера нити внутри параллельной области необходимо вызвать OpenMP-функцию `omp_get_thread_num()`. Для получения значения количества нитей внутри параллельной области необходимо вызвать OpenMP-функцию `omp_get_num_threads()`.

   В параллельную область вставьте следующий код для вывода на экран строки:

```
printf("I am %d thread from %d threads!\n",
   omp_get_thread_num(), //Номер нити в параллельной области
   omp_get_num_threads() //Количество нитей в параллельной об-
   ласти
);
```

4. Скомпилируйте и запустите ваше приложение. Убедитесь, что результат верный.
5. В параллельной области с помощью оператора if определите четный ли номер нити и выводите строку «`I am <Номер нити> thread from <Количество нитей> threads!`» только в случае четного номера.

6. Скомпилируйте и запустите ваше приложение. Убедитесь, что результат верный.

Указания к заданию 4. Общие и частные переменные в OpenMP: программа «Скрытая ошибка»

1. Создайте проект omp_hide_error в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. В функции main создайте параллельную область с k нитями. Вставьте следующий код:

```
rank = omp_get_thread_num();
printf("I am %d thread.\n", rank);
```

3. Определите переменную rank как общую. Для этого достаточно объявить переменную rank до начала параллельной области. Скомпилируйте и запустите ваше приложение. Верный ли результат выдает программа?
4. Добавьте в параллельную область код, имитирующий длительные вычисления, следующим образом:

```
rank = omp_get_thread_num();
Sleep(100); // Имитация длительных вычислений
printf("I am %d thread.\n", rank);
```

Справка: функция Sleep приостановит выполнение программы на указанный интервал времени, заданный во входном параметре в миллисекундах. Для использования данной функции в вашей программе необходимо подключить заголовочный файл Windows.h.

5. Скомпилируйте и запустите ваше приложение. Верный ли результат выдает программа?
6. Переопределите переменную rank как частную. Для этого добавьте к директиве parallel параметр private(), в круглые скобки поместите переменную rank:

```
#pragma omp parallel private(rank)
```

7. Скомпилируйте и запустите ваше приложение. Объясните все полученные выше результаты.

Указания к заданию 5. Общие и частные переменные в OpenMP: параметр reduction

1. Создайте проект omp_reduction в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. В функции main создайте параллельную область с 2-я нитями.
3. Для распределения вычислений по нитям в параллельной области напишите следующую конструкцию:

```
if (omp_get_thread_num() == 0) {
```

```
    // вычисления для нити с номером 0
}
else {
    // вычисления для нити с номером 1
}
```

Напишите для нити с номером 0 цикл, вычисляющий сумму чисел от 1 до N/2, а для нити с номером 1 – от N/2 до N. Частичные суммы запишите в переменную sum.

4. *Общей или частной должна быть переменная sum?* Переменная sum должна быть с одной стороны частной, чтобы избежать ошибки потери слагаемого при одновременной записи в нее двумя нитями, а с другой стороны – должна быть общей, чтобы иметь возможность сложить частичные суммы, подсчитанные нитями. Для таких случаев удобно использовать OpenMP-параметр reduction.
5. Вставьте в директиве parallel параметр reduction:

```
#pragma omp parallel reduction(+:sum)
```

Синтаксис параметра reduction:

reduction(*операция***:***список***)**, где

операция – это одна из операций +, *, -, &, |, ^, &&, || (для языка C);

список – это список общих переменных, для каждой из которых создаются локальные копии в каждой нити. Над локальными копиями переменных после выполнения всех операторов параллельной области выполняется *операция*. Локальные копии инициализируются соответственно типу *операции* (для аддитивных операций – 0 или его аналоги, для мультипликативных операций – 1 или ее аналоги).

6. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.

Указания к заданию 6. Распараллеливание циклов в OpenMP: программа «Сумма чисел»

1. Откройте проект omp_reduction в Microsoft Visual Studio 2010 (см. указания к заданию 5).
2. Определите параметр k. В параллельной области функции main задайте k нитей.
3. В параллельную область вставьте директиву for, которая самостоятельно будет производить распределение итераций по нитям:

```
#pragma omp for
```

После директивы for должен идти только оператор for языка С. Счетчик цикла должен принимать все (!) значения от 1 до N:

```
#pragma omp for
   for(i=1; i<=N; i++) {
       // вычисление суммы чисел от 1 до N }
```

4. В результате выполнения такого цикла каждая из k нитей будет выполнять k-ю часть всех имеющихся итераций цикла. При этом на вид параллельных циклов накладывается ограничение: программа не должна зависеть от того, какая именно нить, какую итерацию параллельного цикла выполнит, т.е. **итерации цикла должны быть не зависимы**!

   В данной задаче это условие выполняется, т.к. очередность сложения слагаемых не важна.

5. Для сложения частичных сумм используйте параметр reduction.
6. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.

Указания к заданию 7. Распараллеливание циклов в OpenMP: параметр schedule

1. Откройте проект omp_reduction в Microsoft Visual Studio 2010 (см. указания к заданию 6).

1. В параллельную область вставьте вывод сообщения «`[<Номер нити>]: calculation of the iteration number <Номер итерации>.`».
2. Добавьте для директивы for параметр schedule, который задает, каким образом итерации цикла распределяются между нитями. Присваивая ему поочередно значения из таблицы, компилируйте и запускайте ваше приложение. Результаты выполнения запишите в таблицу, данную в задании.

   Синтаксис параметра schedule:

   `schedule(type[, chunk])`,

   где type задает тип распределения итераций; основные значения следующие:

   `static` – блочно-циклическое распределение итераций цикла; размер блока – chunk. Первый блок из chunk итераций выполняет нулевая нить, второй блок — следующая и т.д. до последней нити, затем распределение снова начинается с нулевой нити. Если значение chunk не указано, то все множество итераций делится на непрерывные куски примерно одинакового размера (конкретный способ зависит от реализации), и полученные порции итераций распределяются между нитями.

   `dynamic` – динамическое распределение итераций с фиксированным

размером блока: сначала каждая нить получает chunk итераций (по умолчанию chunk=1), та нить, которая заканчивает выполнение своей порции итераций, получает первую свободную порцию из chunk итераций. Освободившиеся нити получают новые порции итераций до тех пор, пока все порции не будут исчерпаны. Последняя порция может содержать меньше итераций, чем все остальные.

`guided` – динамическое распределение итераций, при котором размер порции уменьшается с некоторого начального значения до величины chunk (по умолчанию chunk=1) пропорционально количеству еще не распределенных итераций, деленному на количество нитей, выполняющих цикл. Размер первоначально выделяемого блока зависит от реализации. В ряде случаев такое распределение позволяет аккуратнее разделить работу и сбалансировать загрузку нитей. Количество итераций в последней порции может оказаться меньше значения chunk.

Указания к заданию 8. Распараллеливание циклов в OpenMP: программа «Число π»

1. Создайте проект omp_pi в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. Напишите последовательную программу, вычисляющую число π с точностью до N знаков после запятой по приведенной в задании формуле.
3. Определите код, который можно распараллелить, и задайте параллельную область с помощью директивы parallel.
4. Для распределения работы по нитям используйте OpenMP-директиву for. Убедитесь, что все итерации распараллеливаемого цикла независимы!
5. Изучите все переменные в параллельной области. Определите, общие переменные, частные и reduction.
6. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.

Указания к заданию 9. Распараллеливание циклов в OpenMP: программа «Матрица»

1. Создайте проект omp_matrix в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. Напишите последовательную программу, вычисляющую произведение матриц по приведенной в задании формуле.
3. Определите код, который можно распараллелить, и задайте параллельную область с помощью директивы parallel.
4. Определите цикл for, подходящий для распараллеливания. Используйте для него OpenMP-директиву for.

5. Изучите все переменные в параллельной области. Определите, общие переменные, частные и reduction.
6. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.

Указания к заданию 10. Параллельные секции в OpenMP: программа «I'm here»

1. Создайте проект omp_sections в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. В функции main создайте параллельную область.
3. В параллельной области вставьте директиву sections, которая определяет набор независимых секций кода, каждая из которых выполняется своей нитью:

```
#pragma omp sections
{
   // Определение секций
}
```

4. Внутри директивы sections определите три участка кода для выполнения одной нитью с помощью директивы section:

```
#pragma omp sections
{
   // Определение секций
   #pragma omp section
   {
       // Участок кода для выполнения одной нитью
   }
   #pragma omp section
   {
       // Участок кода для выполнения одной нитью
   }
   #pragma omp section
   {
       // Участок кода для выполнения одной нитью
   }
}
```

5. Вставьте в каждую секцию вывод на экран сообщения:

```
[<Номер нити>]: came in section <Номер секции>
```

`<Номер секции>` определите самостоятельно по порядку числами 1, 2, 3.

6. Вставьте вне секций внутрь параллельной области вывод на экран сообщения:

```
[<Номер нити>]: parallel region
```

7. Скомпилируйте и запустите ваше приложение на 2-х, 3-х, 4-х нитях. Проследите, как нити распределяются по параллельным секциям.

   Примечание: Какие именно нити будут задействованы, для выполнения какой секции, не специфицируется стандартом OpenMP. Если количество нитей больше количества секций, то часть нитей для выполнения данного блока секций не будет задействована. Если количество нитей меньше количества секций, то некоторым (или всем) нитям достанется более одной секции.

Указания к заданию 11. Гонка потоков в OpenMP: программа «Сумма чисел» с atomic

1. Создайте проект omp_atomic в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. Скопируйте в проект написанный вами ранее код программы, вычисляющей сумму чисел от 1 до N.
3. Переменную, которая была объявлена в параметре reduction, сделайте общей. Перед оператором, в котором происходит вычисление данной переменной, вставьте директиву atomic:

```
#pragma omp atomic
   // Один оператор
```

   Примечания: данная директива относится к идущему непосредственно за ней оператору присваивания (например, вида sum+=expr, где sum – общая переменная, expr – некоторое выражение), гарантируя корректную работу с общей переменной, стоящей в его левой части. На время выполнения оператора блокируется доступ к данной переменной всем запущенным в данный момент нитям, кроме нити, выполняющей операцию. Атомарной является только работа с переменной в левой части оператора присваивания, при этом вычисления в правой части не обязаны быть атомарными.

   Директива atomic может быть применена только для простых выражений, но является наиболее эффективным средством, поскольку многие из допустимых для директивы операций на самом деле выполняются как атомарные на аппаратном уровне.

4. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.

Указания к заданию 12. Гонка потоков в OpenMP: программа «Число π» с critical

1. Создайте проект omp_critical в Microsoft Visual Studio 2010 с поддержкой OpenMP (см. указания к заданию 1).
2. Скопируйте в проект написанный вами ранее код программы, вычисляющей число π.
3. Переменную, которая была объявлена в параметре reduction, сделайте общей. Оператор, в котором происходит вычисление данной переменной, поместите внутрь критической секции:

```
#pragma omp critical
```

```
{
   // Операторы
}
```

Синтаксис директивы critical:

```
#pragma omp critical [(имя)]
{
 // Операторы }
```

где `(имя)` – имя секции; может отсутствовать.

Примечание: в каждый момент времени в критической секции может находиться не более одной нити. Если критическая секция уже выполняется какой-либо нитью, то все другие нити, выполнившие директиву для секции с данным именем, будут заблокированы, пока вошедшая нить не закончит выполнение данной критической секции. Как только работавшая нить выйдет из критической секции, одна из заблокированных на входе нитей войдет в нее. Если на входе в критическую секцию стояло несколько нитей, то случайным образом выбирается одна из них, а остальные заблокированные нити продолжают ожидание.

Все неименованные критические секции условно ассоциируются с одним и тем же именем. Все критические секции, имеющие одно и то же имя, рассматриваются единой секцией, даже если находятся в разных параллельных областях. Побочные входы и выходы из критической секции запрещены.

4. С помощью функции omp_get_wtime() замерьте время работы программы. Скомпилируйте и запустите ваше приложение. Сравните время выполнения программы вычисления числа π с параметром reduction с программой с директивой critical.

Указания к заданию 13. Исследование масштабируемости OpenMP-программ

1. Откройте проект omp_matrix в Microsoft Visual Studio 2010 (см. указания к заданию 9).
2. Модифицируйте код следующим образом:

   - Для замера времени работы программы вызовите функцию omp_get_wtime() до и после параллельной области следующим образом:

```
double start, stop;
start = omp_get_wtime();
#pragma omp parallel
{
   // ...
}
stop = omp_get_wtime();
printf("Computation time: %f sec.\n", stop-start)
```

Функция omp_get_wtime() возвращает значение **в секундах**.

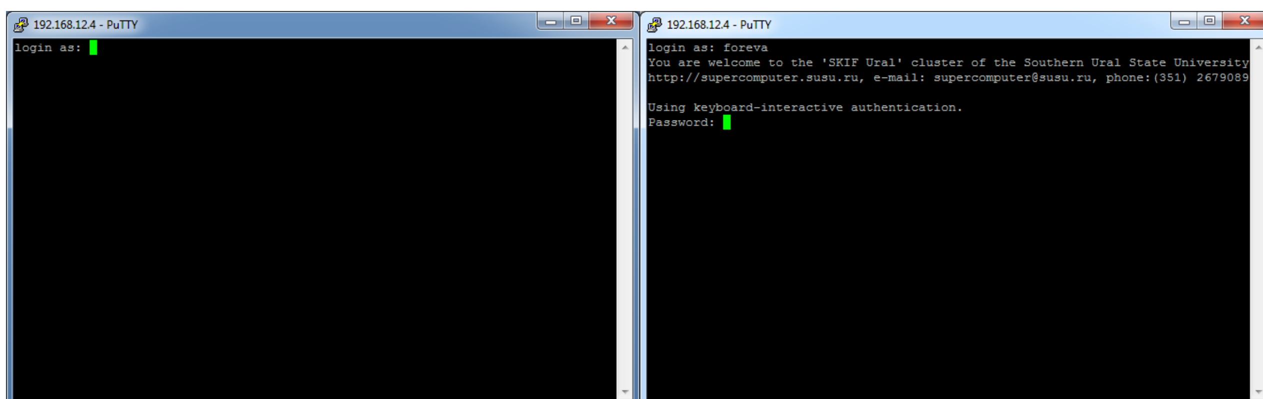
Внимание! Исключите из времени замера все операции ввода/вывода.

- Память под переменные с матрицами A, B, C должна выделяться динамически после ввода пользователем параметра n. Используйте для выделения памяти функцию языка C malloc или calloc.
- Для объявления параметра n используйте расширенный целый тип данных long.
- Уберите вывод на экран всех данных, кроме вычисленного времени работы программы.

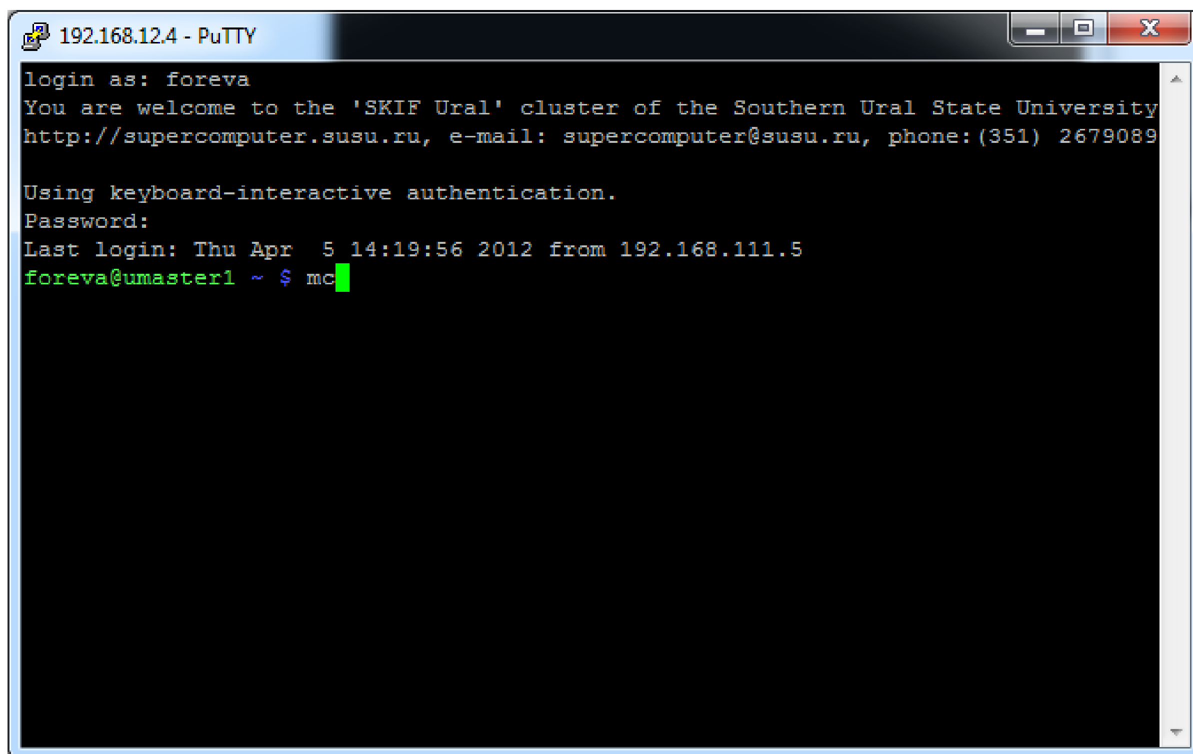
3. Проведите серию экспериментов согласно таблице в задании. Для получения более точных результатов каждый эксперимент необходимо выполнять несколько раз, после чего итоговым результатом считается среднее значение.
4. ***Вычислите ускорение***:
   1) Определить минимальную конфигурацию: k (где k – минимальное кол-во ядер, на которых запускается задача)
   2) Посчитать время для минимальной конфигурации (T_k)
   3) Вычислить время расчета на минимальной конфигурации: $k \times T_k = T_1$
   4) Ускорение $= \frac{T_1}{T_N}$, где $N -$ количество ядер очередного запуска
5. Постройте график масштабируемости для каждого значения параметра n. Определите для каждого графика, при каком количестве нитей достигается максимальное ускорение.
6. Откройте проект omp_pi в Microsoft Visual Studio 2010 (см. указания к заданию 8).
7. Модифицируйте код следующим образом:

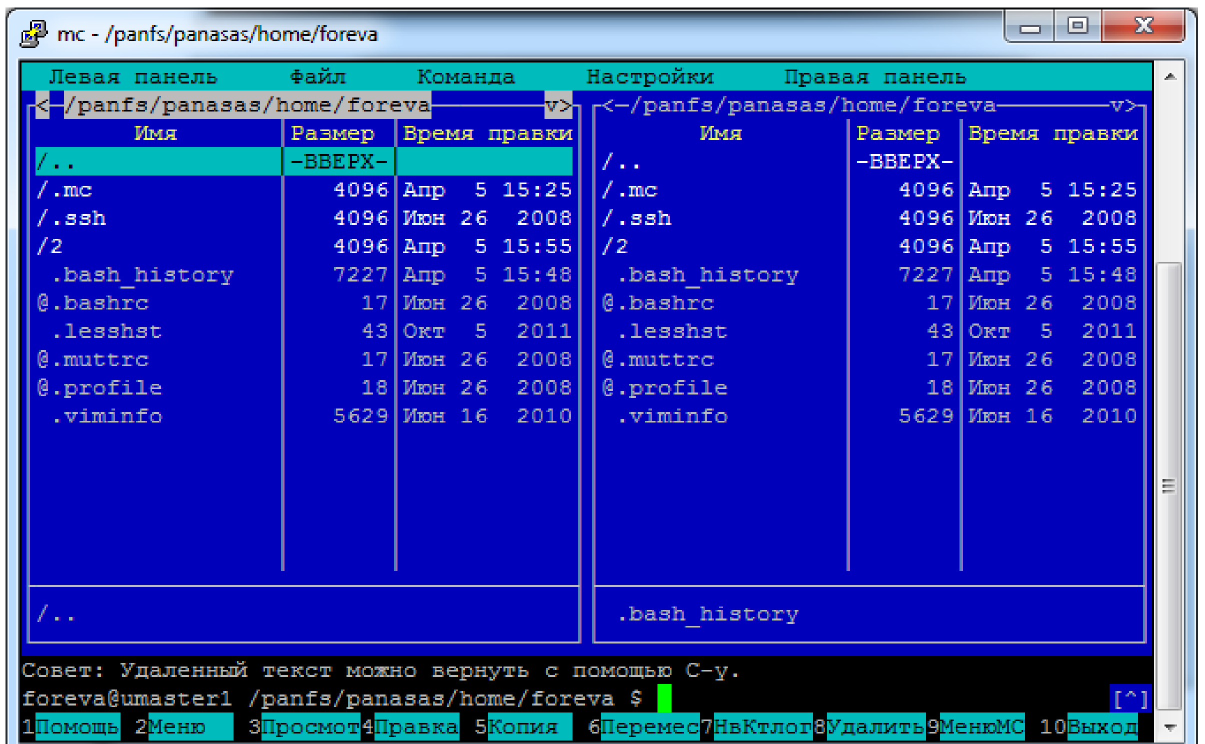
- Для замера времени работы программы используйте функцию omp_get_wtime().
- Для объявления параметра N используйте самый емкий целый тип данных unsigned long. При считывании с экрана данных такого типа функцией scanf используйте формат “%ul”.
- Для объявления переменной, которая будет хранить число π, используйте тип данных double. Для вывода на экран числа π функцией scanf используйте формат “%0.20g”, где два числа, разделенных точкой означают сколько знаков до и после запятой необходимо выводить.

5. ***Подключитесь удаленно к суперкомпьютеру***. Для этого, получите у преподавателя логин и пароль для доступа к суперкомпьютеру.
6. Запустите на персональном компьютере программу putty для подключения к суперкомпьютеру по протоколу SSH.
7. Произведите настройку программы putty согласно инструкции (http://supercomputer.susu.ac.ru/users/instructions/instr3.html) на сайте Лаборатории суперкомпьютерного моделирования.
8. После установки соединения с суперкомпьютером в окне необходимо ввести свой логин, а затем пароль, выданный Вам для доступа к суперкомпьютеру.



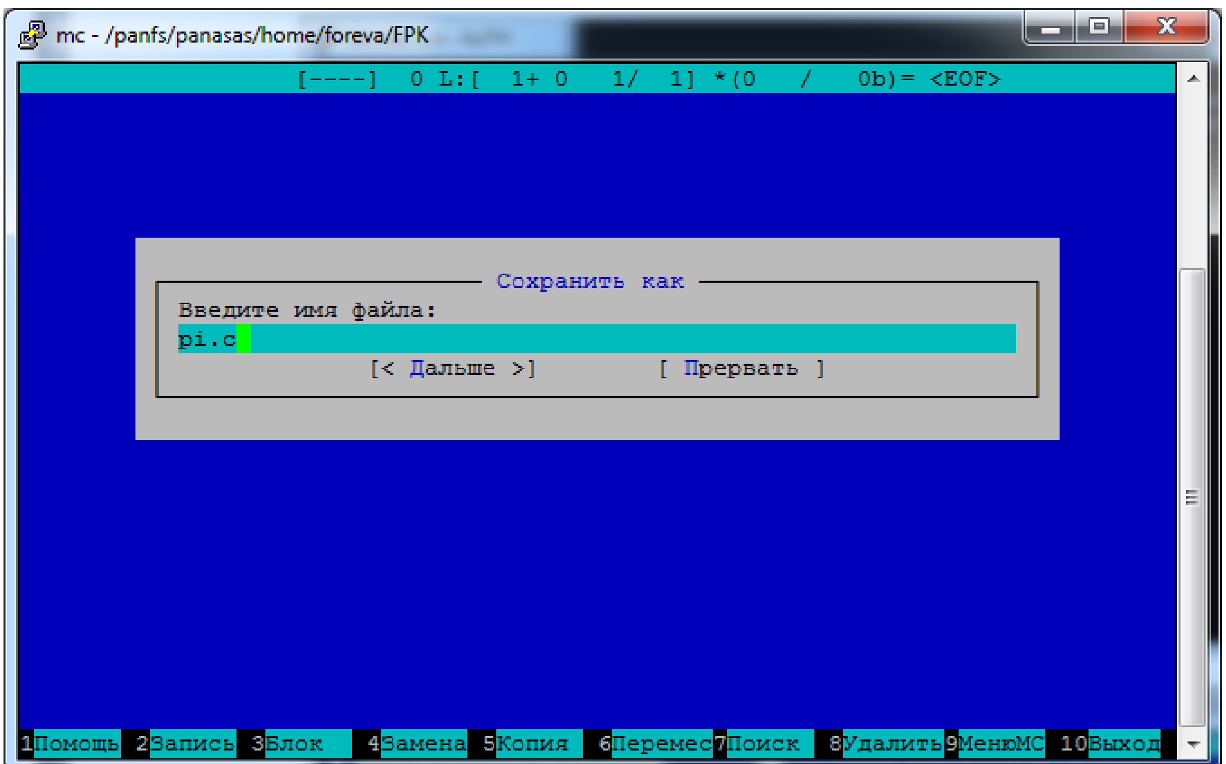
9. После успешного ввода логина и пароля появится приглашение для ввода команд. Введите команду mc для запуска файлового менеджера Midnight Commander.





10. Создайте новый каталог FPK в вашей домашней директории. Для этого нажмите клавишу F7 и введите имя FPK.

11. Перейдите в созданный каталог и создайте там файл pi.c. Для этого нажмите shift+F4. Засдастся и откроется пустой файл. Сохраните его: нажмите F2, кликните кнопку «Сохранить» и введите имя файла. Нажмите «Дальше».



12. Теперь скопируйте в файл текст программы. Для сохранения файла нажмите F2. Для выхода из режима редактирования нажмите два раза кнопку Esc.

13. Скомпилируйте Вашу программу. Для этого перейдите в режим командной строки, нажав Ctrl+O. Затем введите команду

```
icc -openmp ./pi.c -o pi
```

где
`pi.c` – файл с исходным кодом,
`-o` – параметр компилятора, позволяющий задать имя скомпилированного приложения,
`pi` – имя скомпилированного приложения,
`-openmp` – параметр компилятора, позволяющий скомпилировать OpenMP-приложение.

14. Запустите Ваше приложение следующей командой:

```
./pi
```

15. Проведите серию экспериментов согласно таблице в задании. Для получения более точных результатов каждый эксперимент необходимо выполнять несколько раз, после чего итоговым результатом считается среднее значение.

2. Технология программирования MPI

Указания к заданию 14. Создание проекта в среде MS Visual Studio с поддержкой MPI

1. Создайте проект mpi в Microsoft Visual Studio 2010 с минимальным кодом:

```
int main() {
  return 0;
}
```

2. ***Для включения поддержки MPI*** установите дополнительные параметры компиляции проекта:

   - В главном меню выберите *Project-> Имя_проекта Properties*
   - В открывшемся окне выберите *Configuration Properties / C/C++ / General*. Установите значение параметра *Additional Include Directories* в значение «**C:\Program Files\MPICH2\include**». См. Рис.1.

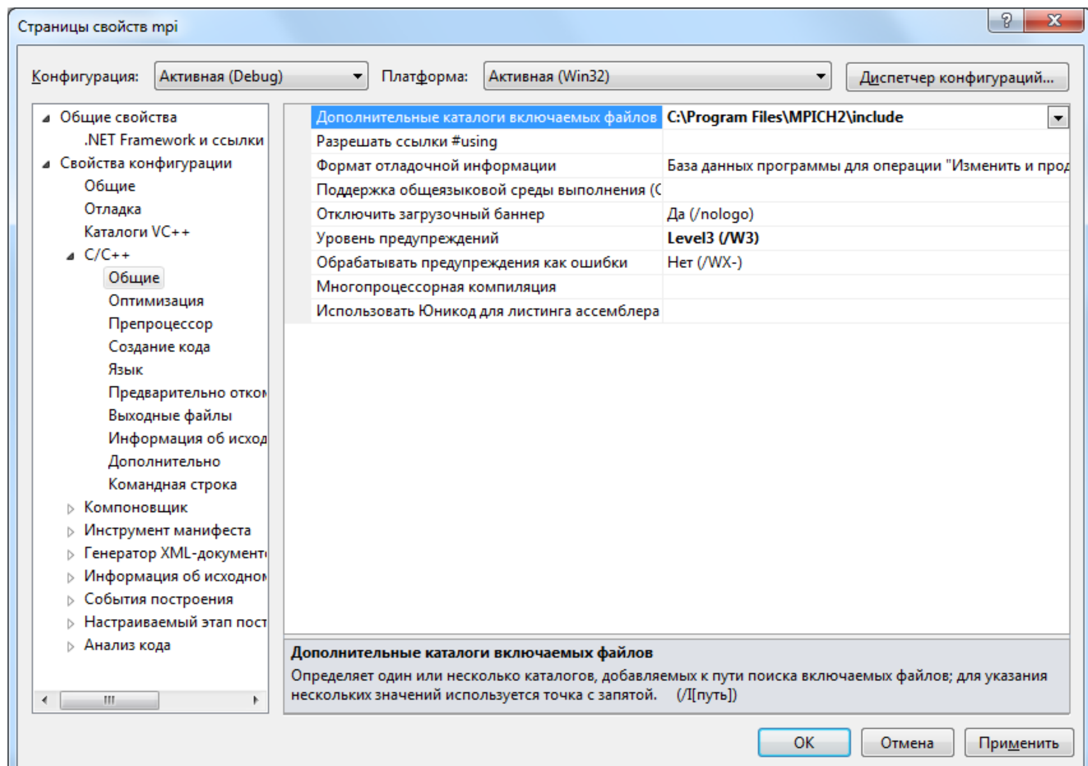


Рис. 1

- Раскройте *Configuration Properties / Linker / General*. Установите значение параметра *Additional Library Directories* в значение «**C:\Program Files\MPICH2\lib**». См. Рис.2.

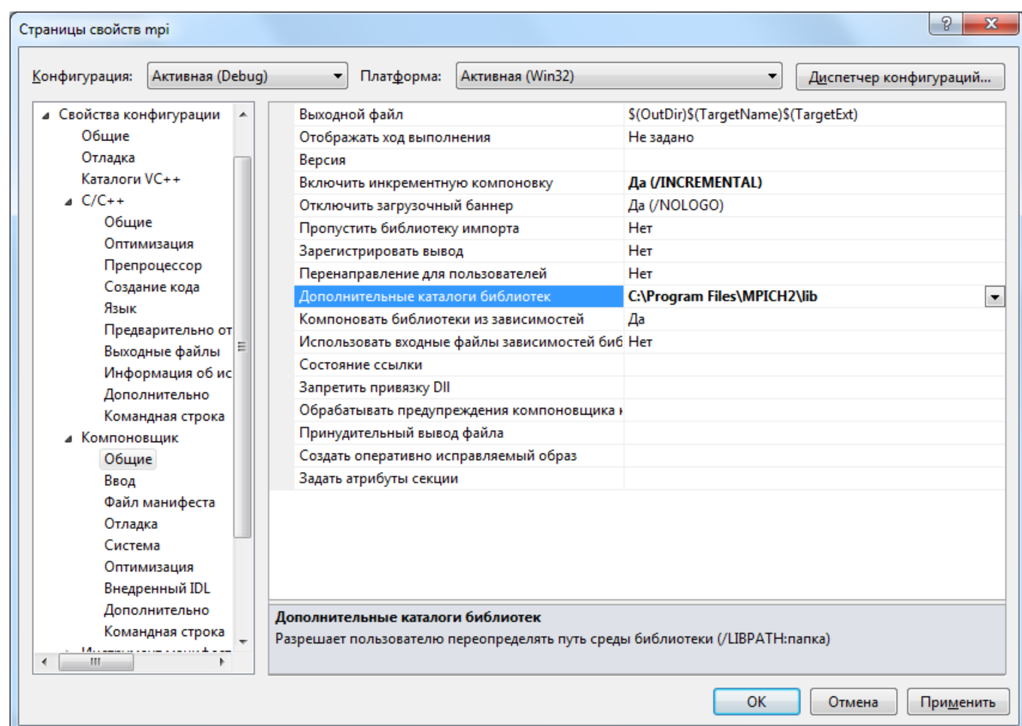


Рис. 2

- Раскройте вкладку *Configuration properties / Linker / Input*. Добавьте к значению параметра *Additional Dependencies*: «**mpi.lib**». См. Рис.3.

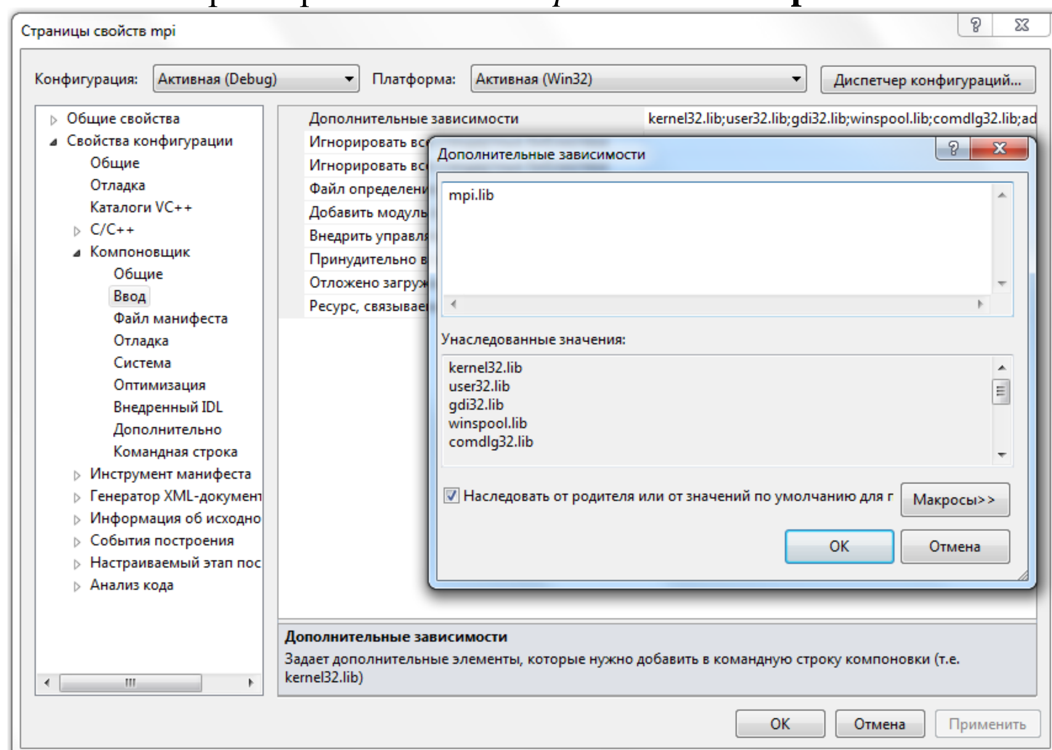


Рис. 3

- ***Для компиляции*** приложения нажмите F7.
- ***Для запуска*** приложения создайте файл run.bat в директории, в которой находится ваше скомпилированное приложение, следующего содержания:

```
"C:\Program Files\MPICH2\bin\mpiexec.exe" -np 2 -noprompt
mpi.exe
PAUSE
```

где
`-np` – параметр задающий количество процессов в приложении,
`-noprompt` – параметр для отмены запроса регистрации,
`mpi.exe` – имя вашего приложения.

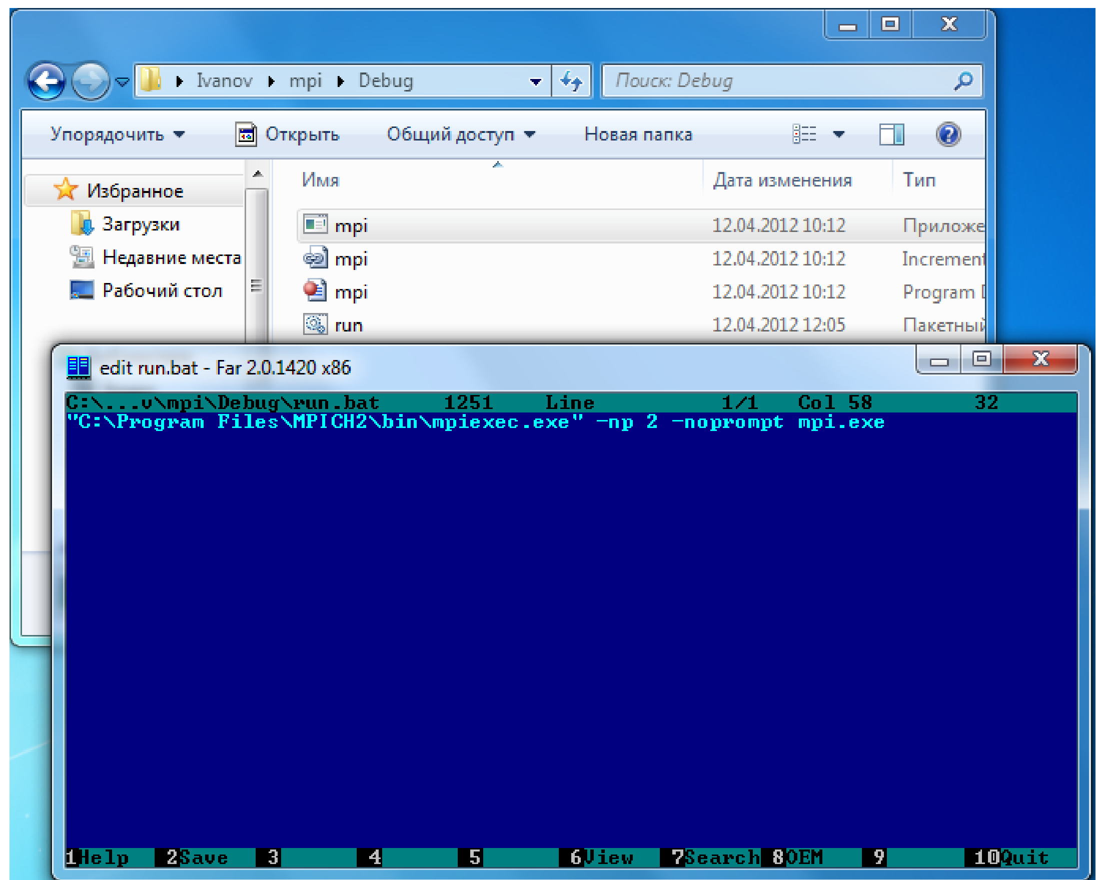


Рис. 4

- Пройдите авторизацию для запуска mpich. Для это перейдите в каталог C:\Program Files\MPICH2\bin\, запустите wmpiregister.exe. В открывшемся окне введите в поле Account ваш логин для входа на компьютеры в учебном классе:

  class\ваш_логин

  и в поле password ваш пароль. См. Рис. 5. Нажмите кнопку Register, убедитесь, что вывелось сообщение «Password encrypted into the Registry.» Нажмите кнопку OK.

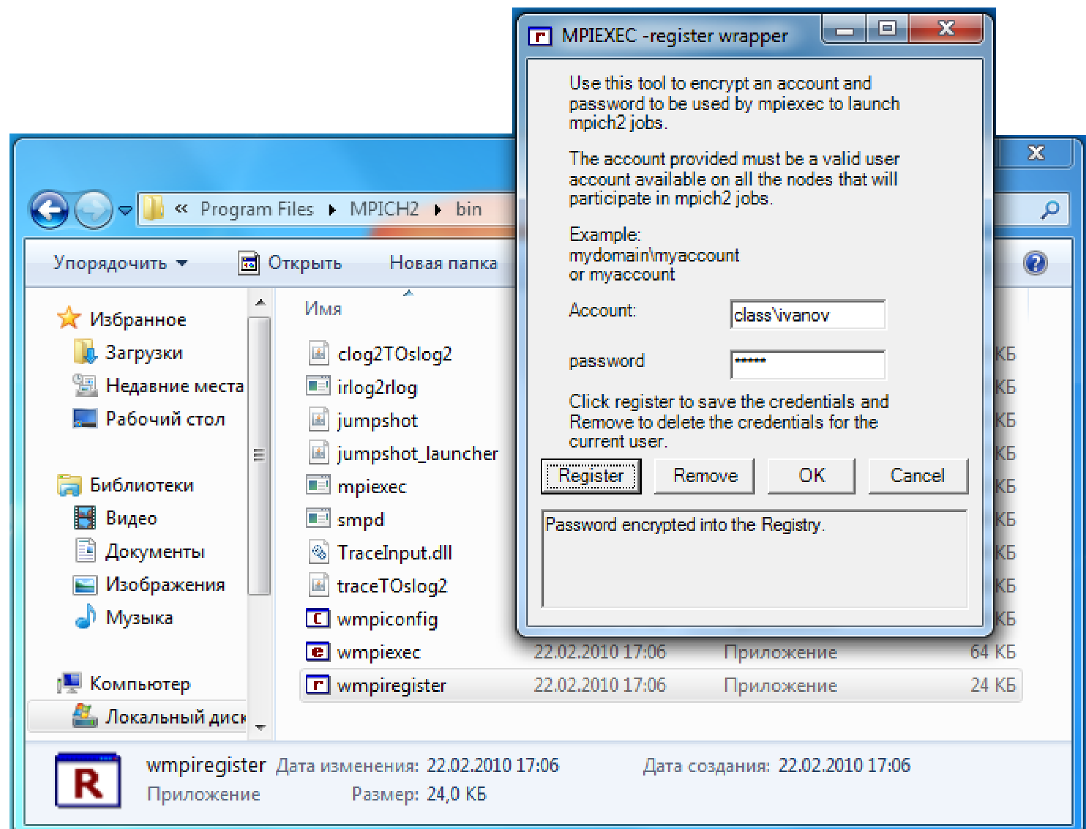


Рис. 5

- Теперь можно запускать созданный вами ранее bat-файл.

Указания к заданию 15. Программа «I am!»

1. Создайте проект mpi_i_am в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Подключите заголовочный файл mpi.h с функциями MPI. Строка подключения заголовочного файла:

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

3. Инициализируйте библиотеку MPI. Для этого в функции main вызовите MPI-функцию ***MPI_Init***:

```
MPI_Init(&argc, &argv);
```

Реальная инициализация для каждого приложения выполняется не более одного раза, а если MPI уже был инициализирован, то никакие действия не выполняются и происходит немедленный возврат из подпрограммы. **Все остальные MPI-функции могут быть вызваны только после вызова MPI_Init!**

В качестве параметров MPI_Init требует параметры командной строки, которые ваша программа получает через параметры функции main argc и argv. Добавьте параметры argc и argv в функцию main:

```
int main(int argc, char *argv[]){
  //...
}
```

4. Определите номер процесса в приложении с помощью функции ***MPI_Comm_rank***:

   ```
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   ```

   где

   первый параметр указывает коммуникатор, в котором определяется номер текущего процесса. `MPI_COMM_WORLD` – коммуникатор, объединяющий все процессы в MPI-приложении. Создается по умолчанию.

   `rank` – целочисленная переменная, в которой функция возвращает номер текущего процесса.

5. Определите количество процессов в приложении с помощью функции ***MPI_Comm_size***:

   ```
   int size;
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   ```

   где

   первый параметр указывает коммуникатор, в котором определяется количество процессов.

   `size` – целочисленная переменная, в которой функция возвращает количество процессов.

6. Вызовите команду вывода строки «`I am <Номер процесса> process from <Количество процессов> processes!`»:

   ```
   printf("I am %d process from %d processes!\n", rank, size);
   ```

7. В конце программы вызовите функцию MPI_Finalize:

   ```
   MPI_Finalize();
   ```

   MPI_Finalize – завершение параллельной части приложения. Все последующие обращения к любым MPI-процедурам, в том числе к MPI_Init, запрещены. К моменту вызова MPI_Finalize процессом все действия, требующие его участия в обмене сообщениями, должны быть завершены.

8. Скомпилируйте и запустите ваше приложение. Убедитесь, что на экран выводится верный результат.

Указания к заданию 16. Программа «На первый-второй рассчитайся!»

1. Создайте проект mpi_fist_second в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Подключите заголовочный файл mpi.h.
3. Инициализируйте библиотеку MPI с помощью функции MPI_Init.
4. Определите номер процесса в приложении с помощью функции MPI_Comm_rank.
5. С помощью оператора switch либо if определите три случая:
    - номер процесса равен 0, тогда в качестве операторов напишите определение количества процессов в приложении с помощью функции MPI_Comm_size и вывод на экран строки «<Количество процессов> processes.»
    - номер процесса является *четным* числом, т.е. оно не 0 и делится на два без остатка (rank%2==0) – выведите строку «I am <Номер процесса>: SECOND!».
    - номер процесса является *нечетным* числом, т.е. оно не делится на два без остатка (rank%2!=0) – выведите строку «I am <Номер процесса>: FIST!».
6. Завершите MPI-приложение функцией MPI_Finalize.
7. Скомпилируйте и запустите ваше приложение. Убедитесь, что на экран выводится верный результат.

Указания к заданию 17. Коммуникации «точка-точка»: простые блокирующие обмены

1. Создайте проект mpi_send в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Ознакомьтесь с основными функциями блокирующей передачи сообщений библиотеки MPI:

    int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm) – блокирующая посылка сообщения с идентификатором msgtag, состоящего из count элементов типа datatype, процессу с номером dest. Все элементы сообщения расположены подряд в буфере buf. Значение count может быть нулем. Тип передаваемых элементов datatype должен указываться с помощью предопределенных констант типа. Разрешается передавать сообщение самому себе.

    Блокировка гарантирует корректность повторного использования всех параметров после возврата из подпрограммы. Выбор способа осуществления этой гарантии: копирование в промежуточный буфер или непосредственная передача процессу dest, остается за MPI. Следует специаль-

но отметить, что возврат из подпрограммы MPI_Send не означает ни того, что сообщение уже передано процессу dest, ни того, что сообщение покинуло процессорный элемент, на котором выполняется процесс, выполнивший MPI_Send.

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Status *status) – прием сообщения с идентификатором msgtag от процесса source с блокировкой. Число элементов в принимаемом сообщении не должно превосходить значения count. Если число принятых элементов меньше значения count, то гарантируется, что в буфере buf изменятся только элементы, соответствующие элементам принятого сообщения. Если нужно узнать точное число элементов в сообщении, то можно воспользоваться подпрограммой MPI_Probe.

Блокировка гарантирует, что после возврата из подпрограммы все элементы сообщения приняты и расположены в буфере buf.

В качестве номера процесса-отправителя можно указать предопределенную константу MPI_ANY_SOURCE - признак того, что подходит сообщение от любого процесса. В качестве идентификатора принимаемого сообщения можно указать константу MPI_ANY_TAG - признак того, что подходит сообщение с любым идентификатором.

Если процесс посылает два сообщения другому процессу и оба эти сообщения соответствуют одному и тому же вызову MPI_Recv, то первым будет принято то сообщение, которое было отправлено раньше.

3. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
4. Создайте сообщение, объявив переменную buf (например, int buf). Присвойте этой переменной некоторое значение процессом 0.
5. Использую функцию MPI_Send, отправьте значение переменной buf от процесса 0 процессу 1. Для отправки переменной целого типа (например, int) в параметрах функции укажите тип MPI_INT из MPI_Datatype; для переменной символьного типа (char) – MPI_CHAR; для вещественных типов (float, double) - MPI_FLOAT, MPI_DOUBLE. При передаче массива данных указывается тип одного элемента и количество элементов в массиве в параметре count.
6. Использую функцию MPI_Recv, получите сообщение от процесса 0 в процессе 1. Выведите полученное значение на экран с помощью printf.
7. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 18. Коммуникации «точка-точка»: схема «эстафетная палочка»

1. Создайте проект mpi_baton в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
3. Запишите номер каждого процесса в переменную rank.
4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf.
6. В схеме коммуникации процессов «эстафетная палочка» для обеспечения последовательной от процесса к процессу передачи сообщения («эстафетной палочки»), процесс должен сначала дождаться получения сообщения, а затем пересылать его следующему процессу. Но все процессы не могут начать с вызова операции получения сообщения, т.к. в случае использования блокирующих операций MPI_Send и MPI_Recv возникнет ситуация ***тупика*** (***deadlock***), при которой все процессы будут простаивать и программа никогда не завершиться. Соответственно выделяют процесс, который инициализирует передачу сообщения, т.е. первым действием выполняет операцию отправки сообщения – это процесс с номером 0. Реализовать это можно следующим образом:

   - С помощью оператора if выделите в программе две секции кода: для процесса с номером 0 и для остальных процессов:

     ```
     if (rank == 0) {
     // Код, выполняемый процессом 0
     }
     else {
     // Код, выполняемый остальными процессами
     }
     ```

   - В секции для процесса 0 присвойте переменной buf значение 0. С помощью MPI_Send отправьте переменную buf процессу 1. Затем, вызвав функцию MPI_Recv, ожидайте сообщение от процесса с номером size-1.
   - В секции для остальных процессов, вызвав функцию MPI_Recv, ожидайте сообщение от процесса с номером rank-1. После получения сообщения увеличьте значение переменной buf на единицу и отправьте его следующему процессу: для процесса с номером size-1это будет 0, для остальных – rank+1.

7. Для всех процессов выведите значение переменной buf на экран с помощью printf.

8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 19. Коммуникации «точка-точка»: схема «мастер-рабочие»

1. Создайте проект mpi_masterslave в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
3. Запишите номер каждого процесса в переменную rank.
4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf.
6. С помощью оператора if выделите в программе две секции кода: для master-процесса и для остальных процессов:

```
if (rank == 0) {
// Код, выполняемый master-процессом
}
else {
// Код, выполняемый slave-процессами
}
```

7. В секции для slave-процессов присвойте переменной buf значение номера процесса. Отправьте buf master-процессу.
8. В секции для master-процесса с помощью оператора for создайте цикл со счетчиком src, изменяющимся от 1 до size-1. В теле цикла с помощью функцию MPI_Recv получите сообщение от процесса с номером src. Выведите полученное сообщение на экран с помощью printf.
9. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 20. Коммуникации «точка-точка»: простые неблокирующие обмены

1. Создайте проект mpi_isend в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Ознакомьтесь с основными функциями неблокирующей передачи сообщений библиотеки MPI:

   int MPI_Isend (void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request) – передача сообщения, аналогичная MPI_Send, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере buf. Это означает, что нельзя по-

вторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. тот момент, когда можно переиспользовать буфер buf без опасения испортить передаваемое сообщение) можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.
Сообщение, отправленное любой из процедур MPI_Send и MPI_Isend, может быть принято любой из процедур MPI_Recv и MPI_Irecv.

int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request) – прием сообщения, аналогичный MPI_Recv, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере buf. Окончание процесса приема можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.

int MPI_Wait (MPI_Request *request, MPI_Status *status) – ожидание завершения асинхронных процедур MPI_Isend или MPI_Irecv, ассоциированных с идентификатором request. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра status.

3. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
4. Создайте сообщение, объявив переменную buf (например, int buf). Присвойте этой переменной некоторое значение процессом 0.
5. Использую функцию MPI_Isend, инициализируйте отправку значение переменной buf от процесса 0 процессу 1.
6. Использую функцию MPI_Irecv, инициализируйте получение сообщения от процесса 0 в процессе 1.
7. С помощью функции MPI_Wait дождитесь завершения процессов получения и отправки сообщения. Выведите в процессе 1 полученное сообщение на экран с помощью printf.
8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 21. Коммуникации «точка-точка»: схема «сдвиг по кольцу»

1. Создайте проект mpi_ring в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
3. Запишите номер каждого процесса в переменную rank.

4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf, присвойте ей значение rank.
6. В схеме коммуникации процессов «сдвиг по кольцу» все процессы выполняют одни и те же действия: отправляют сообщение следующему процессу, затем получают сообщение от предыдущего процесса. Реализовать это можно следующим образом:

   - Определите номер процесса, которому будет отправляться сообщение. Вызовите неблокирующую MPI- функцию отправки сообщения.
   - Определите номер процесса, от которого будет приниматься сообщение. Вызовите неблокирующую MPI- функцию приема сообщения.
   - Дождитесь завершения операций обмена с помощью двух вызовов функции MPI_Wait, либо объединенной функции MPI_Waitall.

     int MPI_Waitall (int count, MPI_Request *array_of_requests, MPI_Status *array_of_statuses) блокирует работу, пока все операции обмена, связанные с активными дескрипторами в списке, не завершатся, и возвращает статус всех операций. Оба массива имеют одинаковое количество элементов. Элемент с номером i в array_of_statuses устанавливается в возвращаемый статус i-ой операции.

7. Для всех процессов выведите значение переменной buf на экран с помощью printf.
8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 22. Коммуникации «точка-точка»: схема «каждый каждому»

1. Создайте проект mpi_all в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
3. Запишите номер каждого процесса в переменную rank.
4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf, присвойте ей значение rank.
6. В схеме коммуникации процессов «каждый каждому» с n процессами каждый процесс отправляет всем другим процессам по одному сообщению

(всего n-1 сообщение), и принимает по одному сообщению от n-1процесса. Реализовать это можно следующим образом:

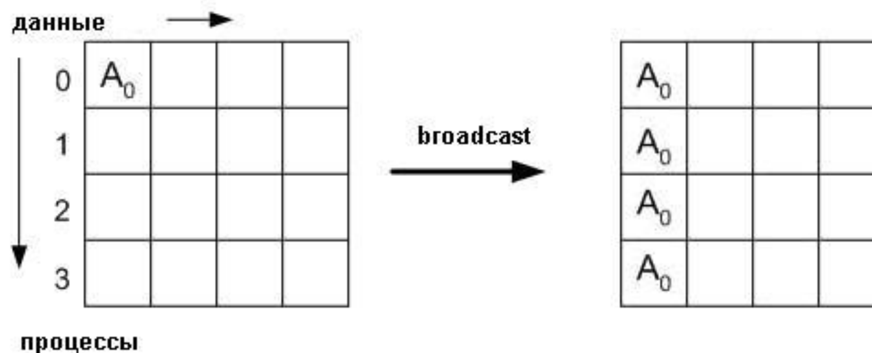
- Создайте цикл от 0 до n-1. Вызовите в теле этого цикла неблокирующую MPI- функцию отправки сообщения, в качестве номера процесса-получателя укажите счетчик цикла. С помощью оператора if исключите возможность отправки сообщения процессом самому себе.
- После завершения цикла. Вызовите функцию ожидания завершения асинхронных операций MPI_Waitall.
- Создайте еще один цикл от 0 до n-1. Вызовите в теле этого цикла неблокирующую MPI- функцию приема сообщения, в качестве номера процесса-получателя укажите счетчик цикла. С помощью оператора if исключите возможность приема сообщения процессом от самого себя.
- После завершения цикла. Вызовите функцию ожидания завершения асинхронных операций MPI_Waitall.

7. Для всех процессов выведите значение переменной buf на экран с помощью printf.
8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 23. Коллективные коммуникации: широковещательная рассылка данных

1. Создайте проект mpi_bcast в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Ознакомьтесь с функцией широковещательной рассылки данных MPI_Bcast:

   int MPI_Bcast (void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm) – процесс с номером root рассылает сообщение из своего буфера передачи всем процессам коммуникатора comm; count - число посылаемых элементов; datatype - тип посылаемых элементов.



3. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.

4. Запишите номер каждого процесса в переменную rank.
5. Запишите количество параллельных процессов в программе в переменную size.
6. Объявите переменную buf в виде массива типа char длинной 100 элементов.
7. В процессе с номером 0 осуществите ввод числа n и строки длинны n в переменную buf.
8. Для передачи числа n в каждом процессе вызовите функцию MPI_Bcast, в качестве root укажите нулевой процесс, в качестве count – 1, в качестве datatype укажите тип данных переменной n.
9. Для передачи строки в каждом процессе вызовите функцию MPI_Bcast, в качестве root укажите нулевой процесс, в качестве count – число n, в качестве datatype укажите тип данных MPI_CHAR.
10. Определите, какой процесс, какие буквы английского алфавита будет искать в строке. Напишите цикл for, в котором каждый процесс перебирает все свои буквы. Например, следующим образом:

```
char a = 'a'; // Первая буква английского алфавита (строчная)
for (i=rank; i<26; i=i+size) {
  // Доступ к символу осуществляется через ASCII код:
  // (int)a + i
}
```

11. Затем внутри данного цикла напишите еще один цикл, который будет перебирать последовательно все символы входной последовательности длины n и сравнивать их с символом (char)((int)a + i). При совпадении необходимо увеличивать на единицу счетчик количества вхождений для каждой буквы.
12. Для всех процессов выведите значения счетчиков букв на экран с помощью printf.
13. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

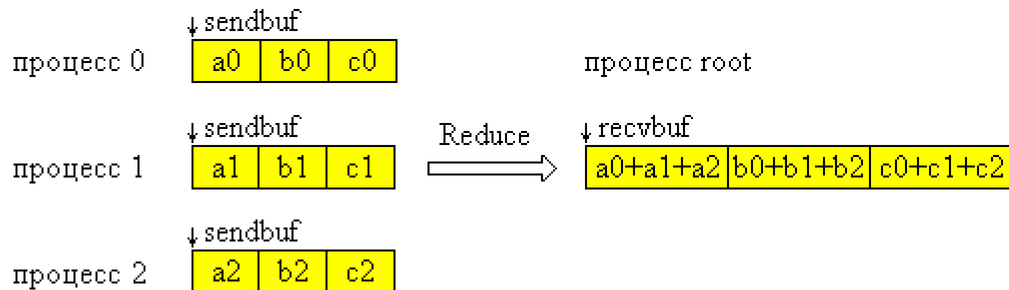
Указания к заданию 24. Коллективные коммуникации: операции редукции

1. Создайте проект mpi_reduce в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Ознакомьтесь с функцией, осуществляющей редукцию данных:

   int MPI_Reduce (void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm). Операция глобальной редукции, указанная параметром op, выполняется над первыми элементами входного буфера, и результат посылается в первый элемент буфера приема

процесса root. Затем то же самое делается для вторых элементов буфера и т.д.

MPI_Op определяет следующие основные операции: MPI_MAX (максимум), MPI_MIN (минимум), MPI_SUM (сумма), MPI_PROD (произведение).



3. Вставьте код программы, вычисляющей число π.
4. Распределите все итерации цикла по процессам.
5. Для рассылки параметра N используйте коллективную функцию MPI_Bcast.
6. Для сбора и суммирования на нулевом процессе всех частичных сумм, посчитанных каждым процессом, используйте коллективную функцию MPI_Reduce. Операцию op определите как MPI_SUM.
7. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

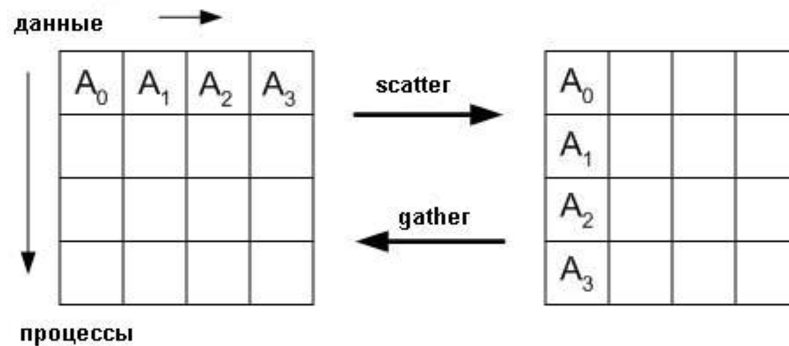
Указания к заданию 25. Коллективные коммуникации: функции распределения и сбора данных

1. Создайте проект mpi_scattergather в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Ознакомьтесь с функциями распределения и сбора блоков данных MPI_Scatter и MPI_Gather:

   int MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm) – разбивает сообщение из буфера посылки процесса root на равные части размером sendcount и посылает i-ю часть в буфер приема процесса с номером i (в том числе и самому себе). Процесс root использует оба буфера (посылки и приема), поэтому в вызываемой им подпрограмме все параметры являются существенными. Остальные процессы группы с коммуникатором comm являются только получателями, поэтому для них параметры, специфицирующие буфер посылки, не существенны.

   int MPI_Gather (void* sendbuf, int sendcount, MPI_Datatype sendtype,

void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm) – производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root. Длина блоков предполагается одинаковой. Объединение происходит в порядке увеличения номеров процессов-отправителей. То есть данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.



3. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.
4. Запишите номер каждого процесса в переменную rank.
5. Запишите количество параллельных процессов в программе в переменную size.
6. Разошлите параметр n всем процессам с помощью функции MPI_Bcast.
7. Распределите строки матрицы *A* равномерно по процессам с помощью функции MPI_Scatter.
8. Разошлите матрицу *B* всем процессам с помощью функции MPI_Bcast.
9. Произведите вычисления строки матрицы C на каждом процессе.
10. С помощью функции MPI_Gather соберите результат вычислений матрицы C со всех процессов на нулевом процессе.
11. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 26. Группы и коммуникаторы

1. Создайте проект mpi_comm в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Процессы параллельной программы объединяются в *группы*. В группу могут входить все процессы параллельной программы либо только часть имеющихся процессов. Один и тот же процесс может принадлежать нескольким группам. Управление группами процессов предпринимается для создания на их основе коммуникаторов.

Под *коммуникатором* в MPI понимается специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (*контекст*), используемых при выполнении операций передачи данных. Для работы с группами изучите следующие основные MPI-функции:

int MPI_Comm_group (MPI_Comm comm, MPI_Group *group) используется для получения группы, связанной с существующим коммуникатором. Далее, на основе существующих групп, могут быть созданы новые группы.

MPI_Group_incl (MPI_Group oldgroup, int n, int *ranks, MPI_Group *group) создает новую группу group из существующей группы oldgroup, которая будет включать в себя n процессов, ранги которых перечисляются в массиве ranks.

int MPI_Group_excl (MPI_Group oldgroup, int n, int *ranks, MPI_Group *group) создает новую группу group из группы oldgroup, которая будет включать в себя n процессов, ранги которых не совпадают с рангами, перечисленными в массиве ranks.

После завершения использования группа должна быть удалена с помощью функции **int MPI_Group_free (MPI_Group *group)**.

Для работы с коммуникаторами изучите следующие основные MPI-функции:

int MPI_Comm_create (MPI_Comm oldcom, MPI_Group group, MPI_Comm *newcomm) – создание нового коммуникатора из подмножества процессов существующего коммуникатора. Операция создания коммуникаторов должна выполняться всеми процессами исходного коммуникатора. На процессах, не принадлежащих group, newcomm примет значение нулевого коммуникатора MPI_COMM_NULL.

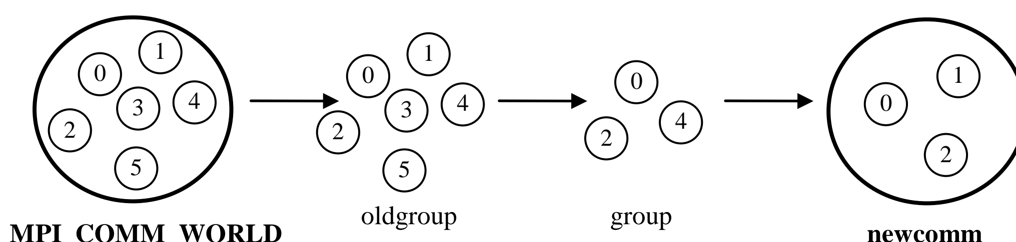
int MPI_Comm_free (MPI_Comm *comm) маркирует коммуникатор для удаления, дескриптор устанавливается в MPI_COMM_NULL. Любые ждущие операции, которые используют этот коммуникатор, будут завершаться нормально; объект фактически удаляется только в том случае, если не имеется никаких других активных ссылок на него.

3. Объявите переменные oldgroup и group типа MPI_Group для создания новой группы.
4. Объявите переменную newcomm типа MPI_Comm для создания нового коммуникатора.
5. Вставьте в код программы вызовы функций MPI_Init и MPI_Finalize.

6. Запишите номер каждого процесса в переменную rank.
7. Запишите количество параллельных процессов в программе в переменную size.
8. Создайте целочисленный массив ranks, поместите туда номера имеющихся в программе четных процессов:

   `ranks = {0,2,4,...,size}`, где size – количество процессов в программе.

9. Т.к. новые группы должны создаваться на основе существующих, получите с помощью MPI_Comm_group начальную группу процессов, из всех процессов программы, т.е. из входящих в коммуникатор MPI_COMM_WORLD. Группу запишите в переменную oldgroup.
10. Создайте группу процессов group с помощью функции MPI_Group_incl.
11. Для группы group создайте коммуникатор newcomm с помощью функции MPI_comm_create.



12. Определите в новом коммуникаторе newcomm количество процессов и номер каждого процесса с помощью функций MPI_Comm_size и MPI_Comm_rank. Запишите номер процесса в переменную newrank, а количество параллельных процессов в переменную newsize.
    Примечание: MPI_Comm_size и MPI_Comm_rank необходимо вызывать только в тех процессах, которые входят в коммуникатор newcomm. Процесс входит в коммуникатор newcomm при выполнении условия newcomm != MPI_COMM_NULL.
13. Получите только нулевым процессом коммуникатора newcomm строку message.
14. С помощью функции MPI_Bcast разошлите строку message всем процессам коммуникатора newcomm.
15. Для каждого процесса в программе выведите на экран сообщение:
    «`MPI_COMM_WORLD: < rank> from <size>. New comm: <newrank> from <newsize>. Message = <message>`»
16. Удалите группы group и oldgroup с помощью MPI_Group_free.

17. Удалите коммуникатор newcomm с помощью функции MPI_Comm_free. Функцию MPI_Comm_free нужно вызывать только в тех процессах, которые входят в удаляемый коммуникатор.
18. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 27*. MPI-2: динамическое создание процессов

1. Создайте проект mpi_dynamic в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Изучите основную функцию для динамического создания процессов:

   int MPI_Comm_spawn (char *command, char **argv, int max-procs,MPI_Info info, int root, MPI_Comm comm, MPI_Comm *intercomm, int *array_of_errcodes) - MPI_Comm_spawn пытается запустить maxprocs одинаковых копий программы MPI, определяемой command, устанавливая с ними соединение и возвращая интеркоммуникатор. Порожденные процессы называются *потомками*, а процессы, их породившие, *родителями*. Потомки имеют свой собственный MPI_COMM_WORLD, отдельный от родителей. Функция MPI_Comm_spawn является коллективной для comm, и не завершается, пока в потомках не вызовется MPI_Init. Подобным образом, MPI_Init в потомках не завершается, пока все родители не вызовут MPI_Comm_spawn. В этом смысле, MPI_Comm_spawn в родителях и MPI_Init в потомках формируют коллективную операцию над объединением родительских и дочерних процессов. Интеркоммуникатор, возвращаемый MPI_Comm_spawn, содержит родительские процессы в локальной группе и процессы-потомки в удаленной группе. Порядок процессов в локальной и удаленной группах такой же, как и порядок группы comm для родителей и MPI_COMM_WORLD для потомков. Этот интеркоммуникатор может быть получен в потомке через функцию MPI_Comm_get_parent.

3. С помощью функции MPI_Comm_spawn создайте в нулевом процессе дополнительные процессы. Распределите вычисления по этим процессам.
4. Результат вычислений соберите на нулевом процессе.
5. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 28*. MPI-2: односторонние коммуникации

1. Создайте проект mpi_comm в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 14).
2. Изучите основные функции для работы с «окном»:

   int MPI_Win_create (void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, MPI_Win *win) – возвращает оконный объект,

который может использоваться этими процессами для выполнения *RMA* операций. Каждый процесс определяет окно в существующей памяти, которое он предоставляет для дистанционного доступа процессам из группы коммуникационного взаимодействия comm. Окно состоит из size байт, начинающихся с адреса base. Процесс может и не предоставлять никакой памяти, при этом size=0. Для упрощения адресной арифметики в RMA операциях представляется аргумент, определяющий единицу смещения: значение аргумента смещения в RMA операции для процесса-адресата масштабируется с коэффициентом disp_unit, определенным адресатом при создании окна.

int MPI_Win_free (MPI_Win *win) - освобождает оконный объект и возвращает пустой дескриптор со значением MPI_WIN_NULL.

Функции работы с окном являются коллективными!

3. Создайте «окно» для возможности осуществлять односторонние коммуникации с помощью функции MPI_Win_create.
4. Изучите основные функции односторонней коммуникации:

   int MPI_Put (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win) - передает origin_count следующих друг за другом записей типа, определяемого origin_datatype, начиная с адреса origin_addr на узле инициатора, узлу адресата, определяемому парой win и target_rank. Данные записываются в буфер адресата по адресу target_addr = window_base + target_disp * disp_unit, где window_base и disp_unit базовый адрес и единица смещения окна, определенные при инициализации оконного обьекта процессом-получателем. Буфер получатель определяется аргументами target_count и target_datatype.

   int MPI_Get (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win) - MPI_Get похожа на MPI_Put, за исключением того, что передача данных происходит в обратном направлении. Данные копируются из памяти адресата в память инициатора. origin_datatype не может определять перекрывающиеся записи в буфере инициатора. Буфер адресата должен находиться в пределах окна адресата, и копируемые данные должны помещаться без округлений в буфер адресата.

5. С помощью функций односторонней коммуникации передайте всем процессам параметр N.
6. Произведите вычисления на каждом процессе.
7. Соберите результат на нулевом процессе с помощью функций односторонней коммуникации.

8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

Указания к заданию 29. Исследование масштабируемости MPI-программ

1. Откройте проект mpi_reduce в Microsoft Visual Studio 2010 (см. указания к заданию 24).
2. Подключитесь удаленно к суперкомпьютеру.
3. Проведите серию экспериментов согласно таблице в задании. Для получения более точных результатов каждый эксперимент необходимо выполнять несколько раз, после чего итоговым результатом считается среднее значение.
4. Для компиляции и запуска вашего приложения изучите документ «Инструкция по компиляции и запуску приложения на СКИФ Урал» (http://supercomputer.susu.ac.ru/users/instructions/instr_compile.html).

   Внимание! Никогда не запускайте свои программы без использования очереди задач, это может повлечь сбой вычислений других пользователей.

3. Технология программирования MPI+OpenMP

Указания к заданию 30. Проект в среде Visual Studio 2010 с поддержкой MPI и OpenMP

1. Создайте консольное приложение в среде Visual Studio с поддержкой OpenMP (см. указания к заданию 1).
2. Подключите к проекту MPICH (см. указания к заданию 14).
3. Скомпилируйте и запустите приложение.

Указания к заданию 31. Программа «I am»

1. Создайте консольное приложение в среде Visual Studio с поддержкой OpenMP и MPI (см. указания к заданию 30).
2. Получите номер процесса и количество процессов в программе в переменные rank_mpi и size_mpi соответственно.
3. Между функциями MPI_Inint и MPI_Finalize создайте с помощью OpenMP-директивы parallel параллельную область:

```
MPI_Init(...);
// Операторы
#pragma omp parallel
{
// Операторы
}
// Операторы
MPI_Finalize();
```

4. Получите номер нити и количество нитей в программе в переменные rank_omp и size_omp соответственно.
5. Внутри параллельной области сформируйте сообщение для вывода на экран:

```
printf("I am  %d thread from  %d> process. Number of hybrid
threads =  %d.\n",  rank_omp,
rank_mpi, size_omp*size_mpi);
```

6. Скомпилируйте и запустите ваше приложение.

Указания к заданию 31. Программа «I am»

1. Создайте консольное приложение в среде Visual Studio с поддержкой OpenMP и MPI (см. указания к заданию 30)
2. Напишите программу, вычисляющую число π. Основная вычислительная часть программы – это цикл for. Распределите все итерации цикла равными долями по процессам, например, следующим образом:

```
for (i=rank; i<N; i=i+size_mpi)
{
// Вычисления
}
```

   где size_mpi – количество MPI-процессов в программе.

3. Внутри процесса распределите итерации по нитям с помощью OpenMP директивы for.
4. Скомпилируйте и запустите ваше приложение. Убедитесь, что выдается верный результат.