



Нижегородский государственный университет
им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Лабораторная работа 4.

Параллельные методы сортировки



Гергель В.П., профессор, д.т.н.
Кафедра математического
обеспечения ЭВМ

Содержание

Упражнения:

- ☐ Постановка задачи
- ☐ Реализация последовательного алгоритма сортировки
- ☐ Разработка параллельного алгоритма сортировки
- ☐ Реализация параллельного алгоритма сортировки



Упражнение 1 – Постановка задачи

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

$$S = \{a_1, a_2, \dots, a_n\}$$

в порядке монотонного возрастания или убывания

$$S \sim S' = \{(a'_1, a'_2, \dots, a'_n) : a'_1 \leq a'_2 \leq \dots \leq a'_n\}$$



Операция "сравнить и переставить"

Базовая операция – "сравнить и переставить" (*compare-exchange*)

```
// базовая операция сортировки
if ( A[i] > A[j] ) {
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
```

- ❑ Последовательное применение данной операции позволяет упорядочить данные
- ❑ В способах выбора пар значений для сравнения проявляется различие алгоритмов сортировки



Пузырьковая сортировка: последовательный алгоритм

```
// Последовательный алгоритм пузырьковой сортировки
BubbleSort(double A[], int n) {
    for (i=0; i<n-1; i++)
        for (j=0; j<n-i; j++)
            compare_exchange(A[j], A[j+1]);
}
```

- ❑ Трудоемкость вычислений имеет порядок $O(n^2)$,
- ❑ В прямом виде сложен для распараллеливания



Упражнение 2 – Последовательный алгоритм сортировки: *этапы разработки*

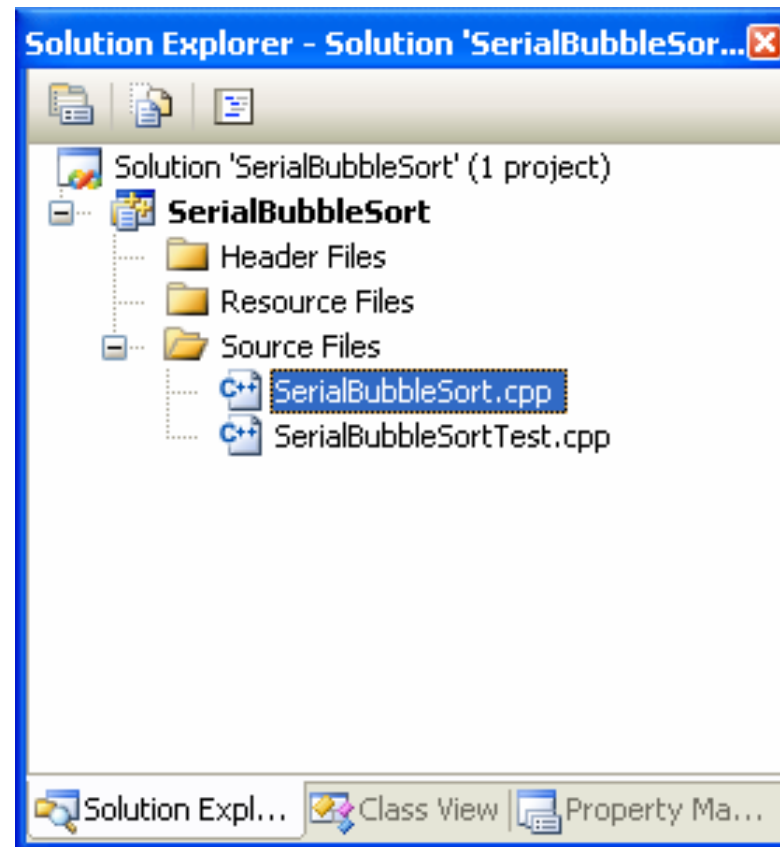
Поэтапная разработка последовательного алгоритма:

- ❑ Задание 1 – Открытие проекта SerialBubbleSort
- ❑ Задание 2 – Ввод сортируемых данных
- ❑ Задание 3 – Завершение процесса вычислений
- ❑ Задание 4 – Реализация алгоритма пузырьковой сортировки
- ❑ Задание 5 – Проведение вычислительных экспериментов



Задание 1 – Открытие проекта SerialBubbleSort

- ❑ Откройте проект **SerialBubbleSort**
- ❑ Откройте файл **SerialBubbleSort.cpp**



Задание 2 – Ввод сортируемых данных...

Функция *ProcessInitialization*:

- ❑ инициализация переменных
- ❑ ввод количества сортируемых данных
- ❑ выделение памяти для сортируемых данных
- ❑ заполнение памяти начальными значениями

```
// Function for allocating the memory and setting  
the initial values
```

```
void ProcessInitialization(double *&pData,  
    int& DataSize);
```

Выходные параметры:

- pData - созданный массив исходных данных
- DataSize - считанное с клавиатуры количество
 исходных данных



Задание 2 – Ввод сортируемых данных...

Функция *DummyDataInitialization* – инициализация данных

- Простое правило формирования данных:
последовательные числа, начиная от количества данных
и до единицы:

(10,9,8,7,6,5,4,3,2,1)

```
// Function for simple setting the initial data  
void DummyDataInitialization(double* pData,  
    int DataSize);
```

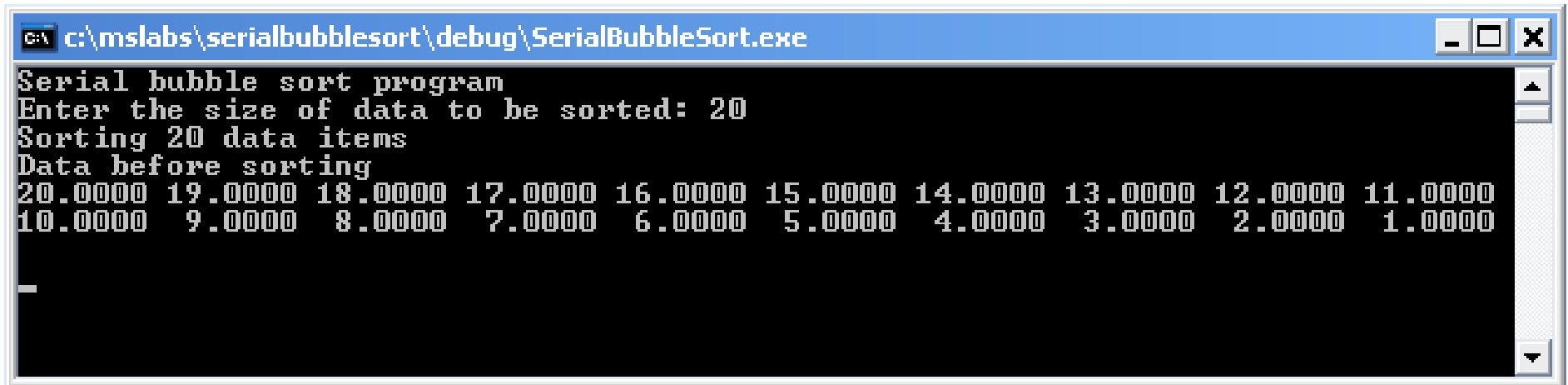
Функция *PrintData* – отладочная печать

```
// Function for formatted data output  
void PrintData(double *pData, int DataSize);
```



Задание 2 – Ввод сортируемых данных

- ❑ Реализуйте функции *ProcessInitialization*, *DummyDataInitialization* и *PrintData*
- ❑ Добавьте вызов функции *ProcessInitialization* в функцию *main*
- ❑ Добавьте отладочную печать в функцию *main*
- ❑ Протестируйте работоспособность приложения



```
C:\mslabs\serialbubblesort\debug\SerialBubbleSort.exe
Serial bubble sort program
Enter the size of data to be sorted: 20
Sorting 20 data items
Data before sorting
20.0000 19.0000 18.0000 17.0000 16.0000 15.0000 14.0000 13.0000 12.0000 11.0000
10.0000 9.0000 8.0000 7.0000 6.0000 5.0000 4.0000 3.0000 2.0000 1.0000
```



Задание 3 – Завершение процесса вычислений...

Функция *ProcessTermination* – освобождение памяти, выделенной динамически в процессе выполнения программы

```
// Function for computational process termination  
void ProcessTermination(double *pData);
```

Входные параметры:

- pData – массив данных для уничтожения



Задание 3 – Завершение процесса вычислений

- ❑ Реализуйте функцию *ProcessTermination*
- ❑ Добавьте вызов функции *ProcessTermination* в функцию *main*
- ❑ Протестируйте работоспособность приложения



Задание 4 – Реализация алгоритма пузырьковой сортировки...

№ шага	Данные
1	<u>13</u> 88 59 55 43
2	13 <u>88</u> ↑ 59↓ 55 43
3	13 59 <u>88</u> ↑ 55↓ 43
4	13 59 55 <u>88</u> ↑ 43↓
5	<u>13</u> 59 55 43 88
...	...
16	13 43 55 59 88

Функция *SerialBubble* – пузырьковая сортировка

```
// Serial bubble sort algorithm  
void SerialBubble(double *pData, int DataSize);
```

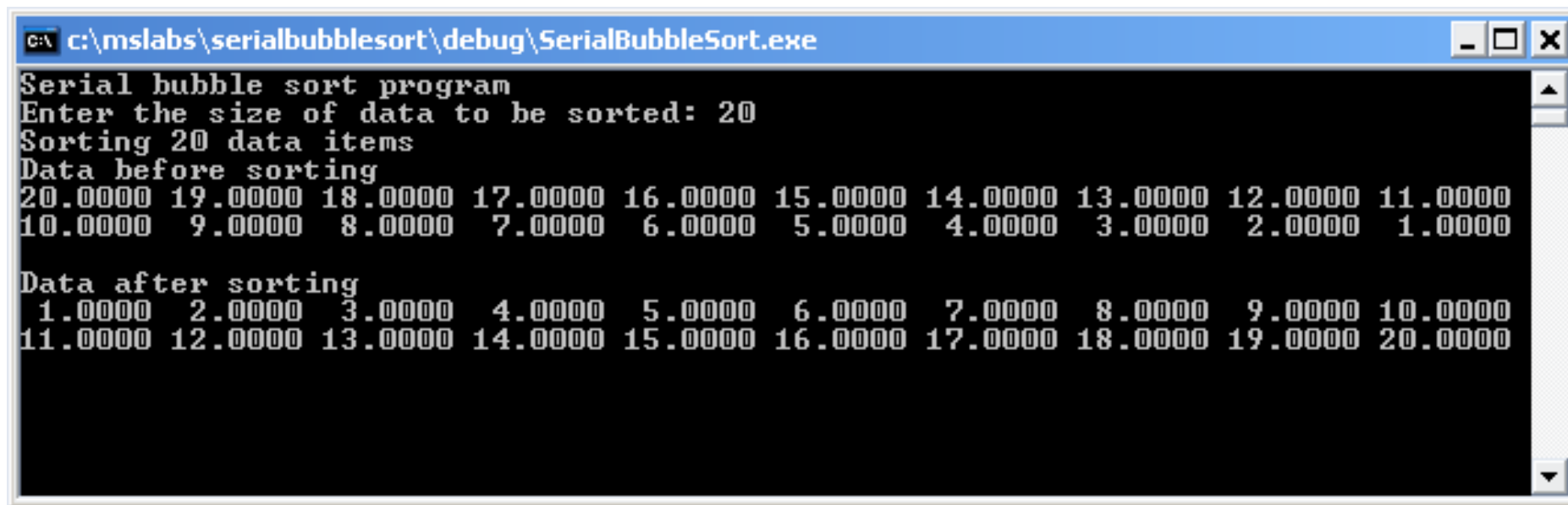
Входные параметры:

- pData – массив данных для сортировки
- DataSize – количество сортируемых данных



Задание 4 – Реализация алгоритма пузырьковой сортировки

- ❑ Реализуйте функцию *SerialBubble*
- ❑ Добавьте отладочную печать в функцию *main*
- ❑ Протестируйте работоспособность приложения



```
c:\mslabs\serialbubblesort\debug\SerialBubbleSort.exe
Serial bubble sort program
Enter the size of data to be sorted: 20
Sorting 20 data items
Data before sorting
20.0000 19.0000 18.0000 17.0000 16.0000 15.0000 14.0000 13.0000 12.0000 11.0000
10.0000 9.0000 8.0000 7.0000 6.0000 5.0000 4.0000 3.0000 2.0000 1.0000
Data after sorting
1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000
11.0000 12.0000 13.0000 14.0000 15.0000 16.0000 17.0000 18.0000 19.0000 20.0000
```



Задание 5 – Проведение вычислительных экспериментов...

Функция *RandomDataInitialization* – задание исходных данных при помощи датчика случайных чисел

```
// Function for initializing the data by the  
random generator
```

```
void RandomDataInitialization(double* pData,  
    int DataSize);
```

Входные параметры:

- pData – массив данных для инициализации
- DataSize – количество данных для инициализации



Задание 5 – Проведение вычислительных экспериментов

- ❑ Реализуйте функцию *RandomDataInitialization*
- ❑ Замените вызов функции *DummyDataInitialization* на *RandomDataInitialization* в функции *ProcessInitialization*
- ❑ Добавьте вычисление и вывод времени выполнения сортировки
- ❑ Протестируйте работоспособность приложения
- ❑ Измерьте времена работы пузырьковой сортировки при различных количествах исходных данных
- ❑ Измерьте времена работы сортировки из стандартной библиотеки при различных количествах исходных данных
- ❑ Вычислите время выполнения операции "*сравнить и переставить*" и рассчитайте теоретическое время работы
- ❑ Заполните таблицу результатов вычислений



Упражнение 3 – Разработка параллельного алгоритма сортировки...

Параллельное обобщение базовой операции при $p = n$
(каждый процессор содержит 1 элемент данных)

- ❑ Выполнить взаимообмен имеющихся на процессорах p_i и p_j значений (с сохранением на этих процессорах исходных элементов),
- ❑ Сравнить на каждом процессоре p_i и p_j получившиеся одинаковые пары значений (a_i, a_j) и по результатам сравнения разделить данные между процессорами – на одном процессоре (например, p_i) оставить меньший элемент, на другом процессоре (т.е. p_j) запомнить большее значение пары

$$a'_i = \min(a_i, a_j)$$

$$a'_j = \max(a_i, a_j)$$



Упражнение 3 – Разработка параллельного алгоритма сортировки...

Параллельное обобщение базовой операции при $p < n$

(каждый процессор содержит блок данных размера n / p)

- ❑ Упорядочить блок на каждом процессоре в начале сортировки,
- ❑ Выполнить взаимообмен блоков между процессорами p_i и p_{i+1} ,
- ❑ Объединить блоки A_i и A_{i+1} на каждом процессоре в один отсортированный блок с помощью операции слияния,
- ❑ Разделить полученный двойной блок на две равные части и оставить одну из этих частей (например, с меньшими значениями данных) на процессоре p_i , а другую часть (с большими значениями соответственно) – на процессоре p_{i+1}

$$[A_i \cup A_{i+1}]_{\text{сорт}} = A'_i \cup A'_{i+1} : \forall a'_i \in A'_i, \forall a'_j \in A'_{i+1} \Rightarrow a'_i \leq a'_j$$

Рассмотренная процедура обычно именуется в литературе как операция "*сравнить и разделить*" (*compare-split*)



Упражнение 3 – Разработка параллельного алгоритма сортировки...

Результат выполнения параллельного алгоритма

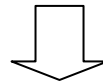
- Имеющиеся на процессорах данные упорядочены
- Порядок распределения данных по процессорам соответствует линейному порядку нумерации (т.е значение последнего элемента на процессоре p_i меньше или равно значения первого элемента на процессоре p_{i+1} , $0 \leq i < p-1$)



Пузырьковая сортировка: алгоритм чет-нечетной перестановки

```
// Последовательный алгоритм чет-нечетной перестановки
OddEvenSort(double A[], int n) {
    for (i=1; i<n; i++) {
        if ( i%2==1) // нечетная итерация
            for (j=0; j<n/2-1; j++)
                compare_exchange(A[2j+1],A[2j+2]);
        if (i%2==0) // четная итерация
            for (j=1; j<n/2-1; j++)
                compare_exchange(A[2j],A[2j+1]);
    }
}
```

Разные правила для выполнения четных и нечетных итераций



Возможности распараллеливания



Пузырьковая сортировка: *параллельный алгоритм чет-нечетной перестановки...*

```
// Параллельный алгоритм чет-нечетной перестановки
ParallelOddEvenSort ( double A[], int n ) {
    int id = GetProcId(); // номер процесса
    int np = GetProcNum(); // количество процессов
    for ( int i=0; i<np; i++ ) {
        if ( i%2 == 1 ) { // нечетная итерация
            if ( id%2 == 1 ) // нечетный номер процесса
                compare_split_min(id+1); // сравнение-обмен справа
            else compare_split_max(id-1); // сравнение-обмен слева
        }
        if ( i%2 == 0 ) { // четная итерация
            if ( id%2 == 0 ) // четный номер процесса
                compare_split_min(id+1); // сравнение-обмен справа
            else compare_split_max(id-1); // сравнение-обмен слева
        }
    }
}
```



Упражнение 4 – Параллельный алгоритм сортировки: *этапы разработки...*

Поэтапная разработка параллельного алгоритма:

- ❑ Задание 1 – Открытие проекта ParallelBubbleSort
- ❑ Задание 2 – Инициализация и завершение параллельной программы
- ❑ Задание 3 – Ввод исходных данных
- ❑ Задание 4 – Завершение процесса вычислений
- ❑ Задание 5 – Распределение данных между процессами
- ❑ Задание 6 – Локальная сортировка данных
- ❑ Задание 7 – Обмен отсортированными данными
- ❑ Задание 8 – Слияние и разделение данных



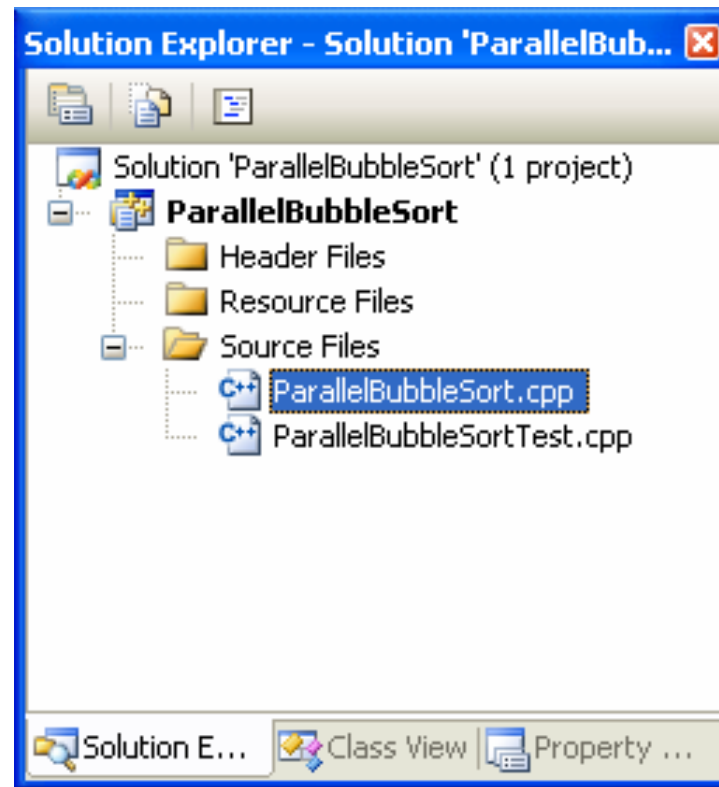
Упражнение 4 – Параллельный алгоритм сортировки: *этапы разработки*

- ❑ Задание 9 – Выполнение итераций параллельной чет-нечетной сортировки
- ❑ Задание 10 – Сбор отсортированных данных
- ❑ Задание 11 – Проверка правильности работы программы
- ❑ Задание 12 – Реализация сортировки для любого количества сортируемых данных
- ❑ Задание 13 – Проведение вычислительных экспериментов



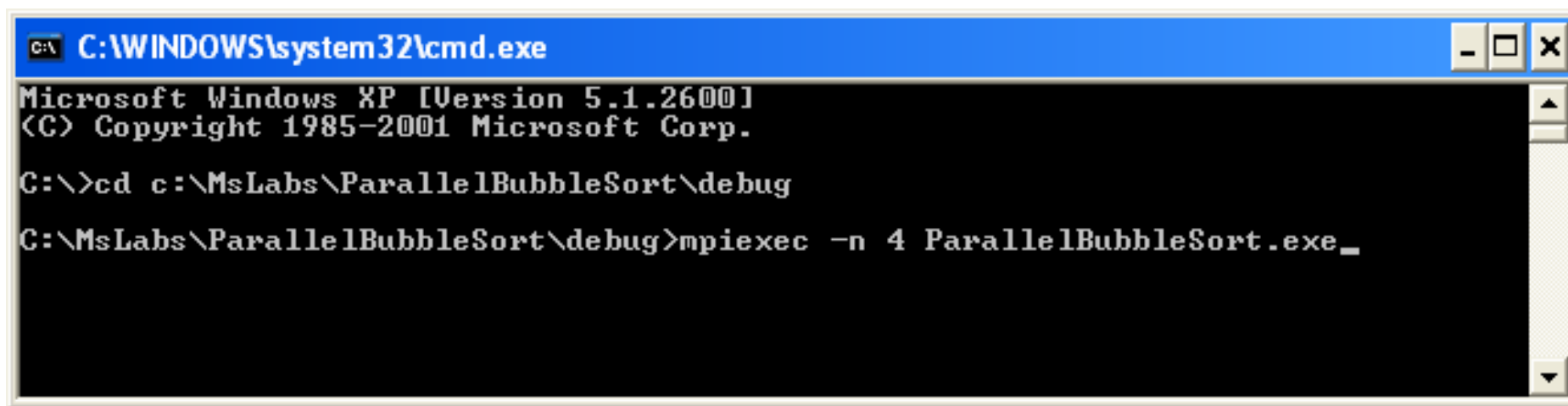
Задание 1 – Открытие проекта ParallelBubbleSort

- ❑ Откройте проект **ParallelBubbleSort**
- ❑ Откройте файл **ParallelBubbleSort.cpp**



Задание 2 – Инициализация и завершение параллельной программы

- ❑ Добавьте инициализацию среды выполнения MPI-программ
- ❑ Протестируйте работоспособность параллельной программы



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>cd c:\MsLabs\ParallelBubbleSort\debug
C:\MsLabs\ParallelBubbleSort\debug>mpiexec -n 4 ParallelBubbleSort.exe_
```



Задание 3 – Ввод исходных данных...

Функция *ProcessInitialization*:

- ☐ инициализация переменных
- ☐ ввод количества сортируемых данных *ведущим процессом*
- ☐ распределение количества данных между процессами
- ☐ выделение памяти для сортируемых данных на ведущем процессе
- ☐ выделение памяти для блоков данных на всех процессах



Задание 3 – Ввод исходных данных...

```
// Function for allocating the memory and setting  
the initial values  
void ProcessInitialization(double *&pData,  
int& DataSize, double *&pProcData, int& BlockSize);
```

Выходные параметры:

- pData – созданный массив исходных данных на ведущем процессе
- DataSize – считанное с клавиатуры количество исходных данных
- pProcData – блок данных процесса
- BlockSize – размер блока данных процесса



Задание 3 – Ввод исходных данных

- ❑ Реализуйте функцию *ProcessInitialization*
- ❑ Добавьте вызов функции *ProcessInitialization* в функцию *main*
- ❑ Протестируйте работоспособность приложения



Задание 4 – Завершение процесса вычислений...

Функция ***ProcessTermination*** – освобождение памяти, выделенной динамически в процессе выполнения программы

```
// Function for computational process termination  
void ProcessTermination(double *pData,  
    double *pProcData);
```

Входные параметры:

- pData – массив данных в динамической памяти
- pProcData – блок данных процесса



Задание 4 – Завершение процесса вычислений

- ❑ Реализуйте функцию *ProcessTermination*
- ❑ Добавьте вызов функции *ProcessTermination* в функцию *main*
- ❑ Протестируйте работоспособность приложения



Задание 5 – Распределение данных между процессами...

Функция ***DataDistribution*** – распределение данных между процессами

```
// Data distribution among the processes  
void DataDistribution(double *pData,  
    int DataSize, double *pProcData,  
    int BlockSize);
```

Входные параметры:

- pData – массив данных для распределения
- DataSize – количество данных для распределения
- BlockSize – размер блока данных процесса

Выходные параметры:

- pProcData – заполненный блок данных процесса



Задание 5 – Распределение данных между процессами...

Функция ***TestDistribution*** – проверка правильности выполнения распределения данных

```
// Function for testing the data distribution  
void TestDistribution(double *pData,  
    int DataSize, double *pProcData,  
    int BlockSize);
```

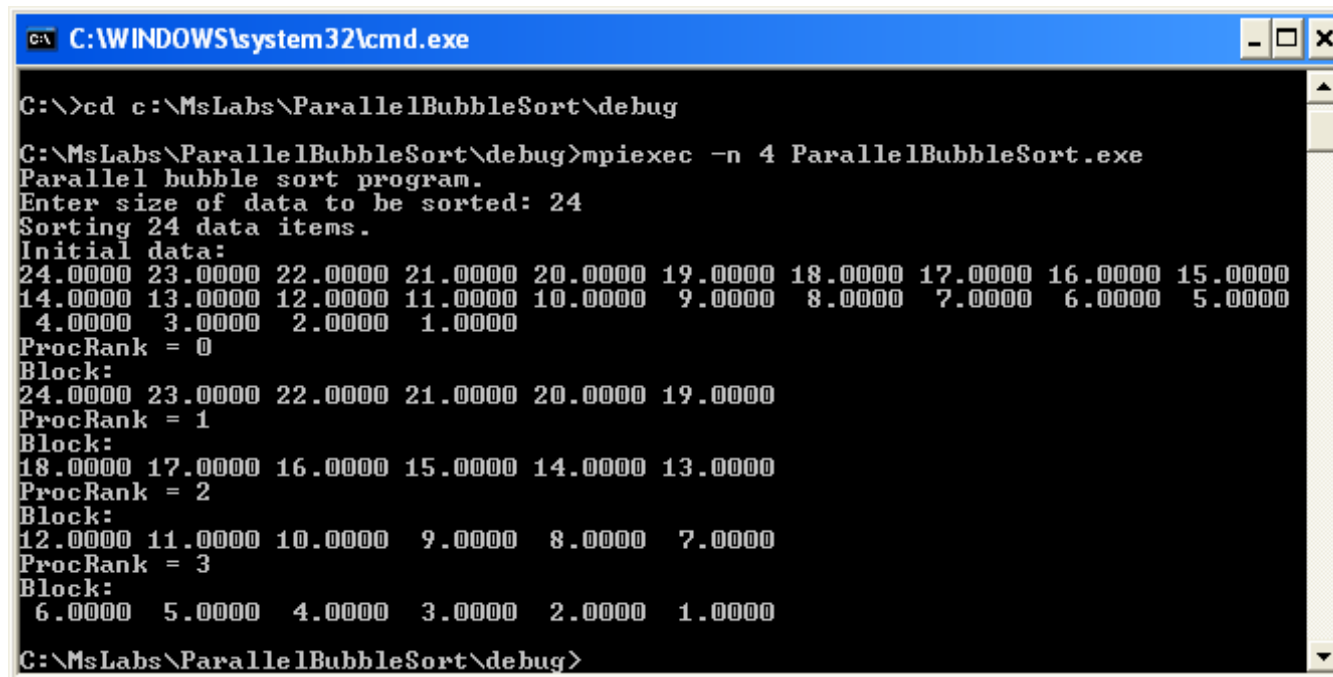
Входные параметры:

- pData – массив исходных данных на ведущем процессе
- DataSize – количество исходных данных
- pProcData – блок данных процесса
- BlockSize – размер блока данных процесса



Задание 5 – Распределение данных между процессами

- ❑ Реализуйте функции *DataDistribution* и *TestDistribution*
- ❑ Добавьте вызовы функций *DataDistribution* и *TestDistribution* в функцию *main*
- ❑ Протестируйте работоспособность приложения



```
C:\WINDOWS\system32\cmd.exe

C:\>cd c:\MsLabs\ParallelBubbleSort\debug

C:\MsLabs\ParallelBubbleSort\debug>mpiexec -n 4 ParallelBubbleSort.exe
Parallel bubble sort program.
Enter size of data to be sorted: 24
Sorting 24 data items.
Initial data:
24.0000 23.0000 22.0000 21.0000 20.0000 19.0000 18.0000 17.0000 16.0000 15.0000
14.0000 13.0000 12.0000 11.0000 10.0000 9.0000 8.0000 7.0000 6.0000 5.0000
4.0000 3.0000 2.0000 1.0000
ProcRank = 0
Block:
24.0000 23.0000 22.0000 21.0000 20.0000 19.0000
ProcRank = 1
Block:
18.0000 17.0000 16.0000 15.0000 14.0000 13.0000
ProcRank = 2
Block:
12.0000 11.0000 10.0000 9.0000 8.0000 7.0000
ProcRank = 3
Block:
6.0000 5.0000 4.0000 3.0000 2.0000 1.0000

C:\MsLabs\ParallelBubbleSort\debug>
```



Параллельный алгоритм чет-нечетной сортировки

№ и тип итерации	Процессоры			
	1	2	3	4
Исходные данные	13 55 59 88	29 43 71 85	2 18 40 75	4 14 22 43
1 нечет (1,2),(3.4)	13 55 59 88	29 43 71 85	2 18 40 75	4 14 22 43
	13 29 43 55	59 71 85 88	2 4 14 18	22 40 43 75
2 чет (2,3)	13 29 43 55	59 71 85 88	2 4 14 18	22 40 43 75
	13 29 43 55	2 4 14 18	59 71 85 88	22 40 43 75
3 нечет (1,2),(3.4)	13 29 43 55	2 4 14 18	59 71 85 88	22 40 43 75
	2 4 13 14	18 29 43 55	22 40 43 59	71 75 85 88
4 чет (2,3)	2 4 13 14	18 29 43 55	22 40 43 59	71 75 85 88
	2 4 13 14	18 22 29 40	43 43 55 59	71 75 85 88



Задание 6 – Локальная сортировка данных...

Функция ***ParallelBubble*** – параллельная сортировка данных

```
// Parallel bubble sort algorithm  
void ParallelBubble(double *pProcData,  
    int BlockSize);
```

Входные параметры:

- pProcData – блок данных процесса для сортировки
- BlockSize – размер блока данных процесса

Функция ***ParallelPrintData*** – отладочная печать
распределенных данных

```
// Function for parallel data output  
void ParallelPrintData(double *pProcData,  
    int BlockSize);
```

Входные параметры:

- pProcData – блок данных процесса
- BlockSize – размер блока данных процесса



Задание 6 – Локальная сортировка данных

- ❑ Реализуйте первую версию функции *ParallelBubble* и функцию *ParallelPrintData*
- ❑ Добавьте вызовы функций *ParallelBubble* и *ParallelPrintData* в функцию *main*
- ❑ Закомментируйте вызов функции *TestDistribution*
- ❑ Протестируйте работоспособность приложения



Задание 7 – Обмен отсортированными данными...

Функция ***ExchangeData*** – обмен процессами копиями своих отсортированных локальных данных

```
// Function for data exchange between the  
neighboring processes  
void ExchangeData(double *pProcData,  
    int BlockSize, int DualRank, double *pDualData);
```

Входные параметры:

- pProcData – блок данных процесса
- BlockSize – размер блока данных процесса
- DualRank – ранг процесса, с которым производится обмен

Выходные параметры:

- pDualData – полученный блок данных процесса, с которым производился обмен



Задание 7 – Обмен отсортированными данными

- ❑ Закомментируйте использовавшуюся ранее отладочную печать
- ❑ Реализуйте функцию *ExchangeData*
- ❑ Добавьте вызов функции *ExchangeData* и отладочную печать в функцию *ParallelBubble*
- ❑ Протестируйте работоспособность приложения



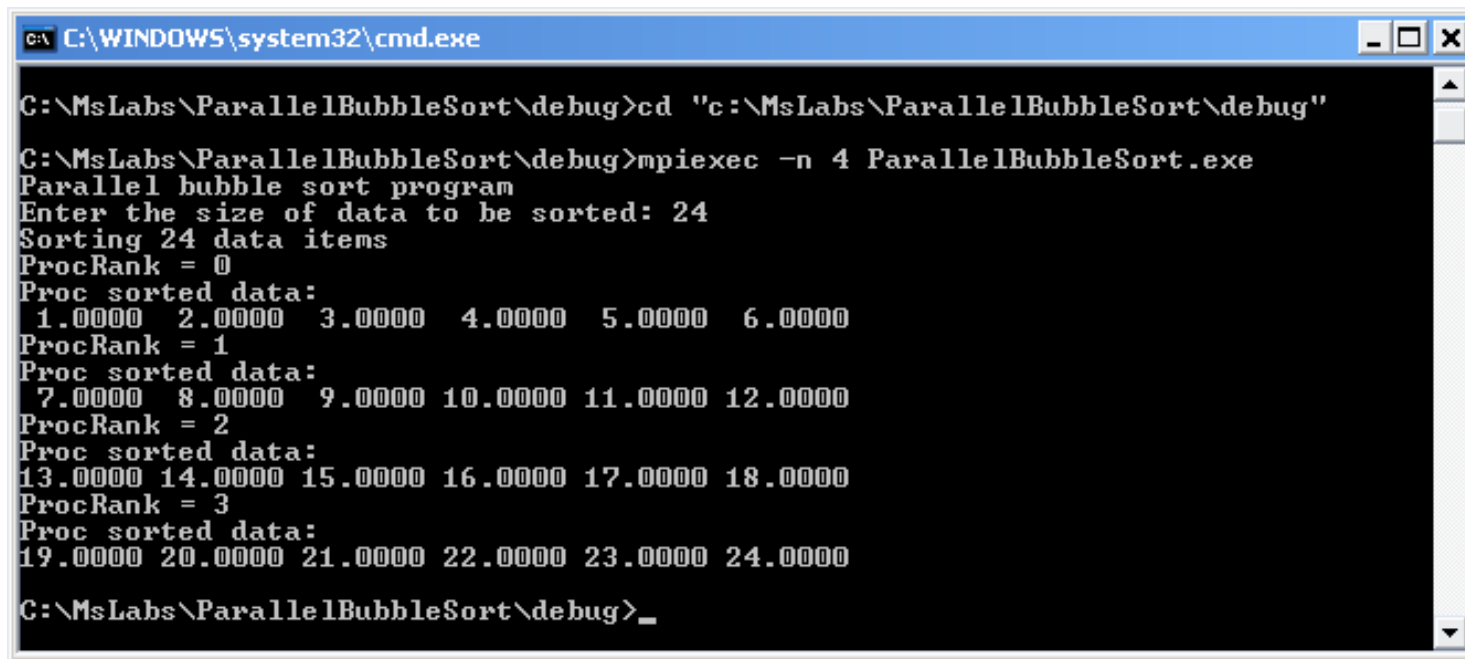
Задание 8 – Слияние и разделение данных

- ❑ Закомментируйте использовавшуюся ранее отладочную печать
- ❑ Добавьте алгоритмы слияния и разделения в функцию *ParallelBubble*
- ❑ Добавьте отладочную печать в функцию *ParallelBubble*
- ❑ Протестируйте работоспособность приложения



Задание 9 – Выполнение итераций параллельной чет-нечетной сортировки

- ❑ Модифицируйте функцию *ParallelBubble* для выполнения итераций алгоритма
- ❑ Протестируйте работоспособность приложения используя отладочную печать



```
C:\WINDOWS\system32\cmd.exe

C:\MsLabs\ParallelBubbleSort\debug>cd "c:\MsLabs\ParallelBubbleSort\debug"

C:\MsLabs\ParallelBubbleSort\debug>mpiexec -n 4 ParallelBubbleSort.exe
Parallel bubble sort program
Enter the size of data to be sorted: 24
Sorting 24 data items
ProcRank = 0
Proc sorted data:
1.0000 2.0000 3.0000 4.0000 5.0000 6.0000
ProcRank = 1
Proc sorted data:
7.0000 8.0000 9.0000 10.0000 11.0000 12.0000
ProcRank = 2
Proc sorted data:
13.0000 14.0000 15.0000 16.0000 17.0000 18.0000
ProcRank = 3
Proc sorted data:
19.0000 20.0000 21.0000 22.0000 23.0000 24.0000

C:\MsLabs\ParallelBubbleSort\debug>_
```



Задание 10 – Сбор отсортированных данных...

Функция **DataCollection** – сбор отсортированных данных на ведущем процессе

```
// Function for data collection  
void DataCollection(double *pData, int DataSize,  
    double *pProcData, int BlockSize);
```

Входные параметры:

- DataSize – количество данных
- pProcData – блок данных процесса
- BlockSize – размер блока данных процесса

Выходные параметры:

- pData – собранный массив данных на ведущем процессе



Задание 10 – Сбор отсортированных данных

- ❑ Закомментируйте использовавшуюся ранее отладочную печать
- ❑ Реализуйте функцию *DataCollection*
- ❑ Добавьте вызов функции *DataCollection* в функцию *main*
- ❑ Протестируйте работоспособность приложения используя функцию *PrintData* на ведущем процессе параллельного приложения



Задание 11 – Проверка правильности работы программы...

Функция ***TestResult*** – сравнение результатов последовательного и параллельного алгоритмов

```
// Function for testing the result of parallel  
bubble sort
```

```
void TestResult(double *pData,  
    double *pSerialData, int DataSize);
```

Входные параметры:

- pData – отсортированный параллельным алгоритмом массив данных
- pSerialData – исходный массив для сортировки последовательным алгоритмом
- DataSize – количество данных



Задание 11 – Проверка правильности работы программы

- ❑ Закомментируйте использовавшуюся ранее отладочную печать
- ❑ Реализуйте функцию *TestResult*
- ❑ Добавьте вызов функции *TestResult* в функцию *main*
- ❑ Замените вызов функции *DummyDataInitialization* на вызов функции *RandomDataInitialization* в функции *ProcessInitialization*
- ❑ Протестируйте работоспособность приложения



Задание 12 – Реализация сортировки для любого количества сортируемых данных

- ❑ Внесите необходимые изменения в функции *ProcessInitialization*, *DataDistribution*, *DataCollection*, *ExchangeData*
- ❑ Проверьте правильность сортировки при помощи функции *TestResult*



Задание 13 – Проведение вычислительных экспериментов

- ☐ Добавьте вычисление и вывод времени выполнения сортировки
- ☐ Протестируйте работоспособность приложения
- ☐ Проведите вычислительные эксперименты
- ☐ Измерьте времена работы пузырьковой сортировки при различных количествах исходных данных и различном числе процессов
- ☐ Рассчитайте ускорение
- ☐ Вычислите теоретическое время работы параллельного алгоритма
- ☐ Заполните таблицу результатов вычислений



Заключение

- ❑ Рассмотрен способ параллельного выполнения метода упорядочения данных – алгоритм пузырьковой сортировки
- ❑ Разработаны приложения, реализующие последовательный и параллельный алгоритмы пузырьковой сортировки
- ❑ Проведены вычислительные эксперименты, проведено сравнение последовательного и параллельного алгоритмов



Темы заданий для самостоятельной работы

- ❑ Модифицируйте разработанное приложение таким образом, чтобы в качестве локальной сортировки использовалась сортировка из стандартной библиотеки. Проведите вычислительные эксперименты и сравните результаты нового варианта программы с полученными ранее
- ❑ Изучите другие параллельные алгоритмы сортировки (сортировка Шелла, различные виды быстрой сортировки – см. раздел 10 "Параллельные методы сортировки данных" учебных материалов курса). Разработайте программы, реализующие эти алгоритмы



Литература

- ❑ **Akl, S. G.** (1985). *Parallel Sorting Algorithms*. – Orlando, FL: Academic Press
- ❑ **Knuth, D.E.** (1973). *The Art of Computer Programming: Sorting and Searching*. – Reading, MA: Addison-Wesley.
- ❑ **Кормен Т., Лейзерсон Ч., Ривест Р.** (1999). *Алгоритмы: построение и анализ*. – М.: МЦНТО.
- ❑ **Kumar V., Grama, A., Gupta, A., Karypis, G.** (1994). *Introduction to Parallel Computing*. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)
- ❑ **Quinn, M. J.** (2004). *Parallel Programming in C with MPI and OpenMP*. – New York, NY: McGraw-Hill.



Следующая тема

- Параллельные методы работы с графами



Авторский коллектив

Гергель В.П., профессор, д.т.н., руководитель

Гришагин В.А., доцент, к.ф.м.н.

Сысоев А.В., ассистент (раздел 1)

Лабутин Д.Ю., ассистент (система ПараЛаб)

Абросимова О.Н., ассистент (раздел 10)

Гергель А.В., аспирант (раздел 12)

Лабутина А.А., аспирант (разделы 7,8,9, система ПараЛаб)

Сенин А.В. (раздел 11)



Целью проекта является создание образовательного комплекса "Многопроцессорные вычислительные системы и параллельное программирование", обеспечивающий рассмотрение вопросов параллельных вычислений, предусматриваемых рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM. Данный образовательный комплекс может быть использован для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы параллельного программирования"** и **лабораторный практикум "Методы и технологии разработки параллельных программ"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения для решения сложных вычислительно-трудоемких задач на высокопроизводительных вычислительных системах.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (<http://www.software.unn.ac.ru>). Выполнение проекта осуществлялось при поддержке компании Microsoft.

