Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Лабораторная работа 5.

Параллельные алгоритмы обработки графов

Microsoft

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

Содержание

Упражнения:

- Постановка задачи
- Реализация последовательного алгоритма Флойда
- Разработка параллельного алгоритма Флойда
- Реализация параллельного алгоритма Флойда

© Гергель В.П.

Обработка графов…

- Математические модели в виде графов широко используются при моделировании разнообразных явлений, процессов и систем
- Граф G есть пара:

$$G = (V, R),$$

где:

- V – множество вершин графа:

$$v_i, 1 \le i \le n$$

- R – множество ребер графа:

$$r_j = (v_{s_j}, v_{t_j}), 1 \le j \le m$$

- В общем случае дугам графа могут приписываться некоторые числовые характеристики (веса) :

$$w_j, 1 \le j \le m$$

Обработка графов…

- Пример взвешенного ориентированного графа

Обработка графов…

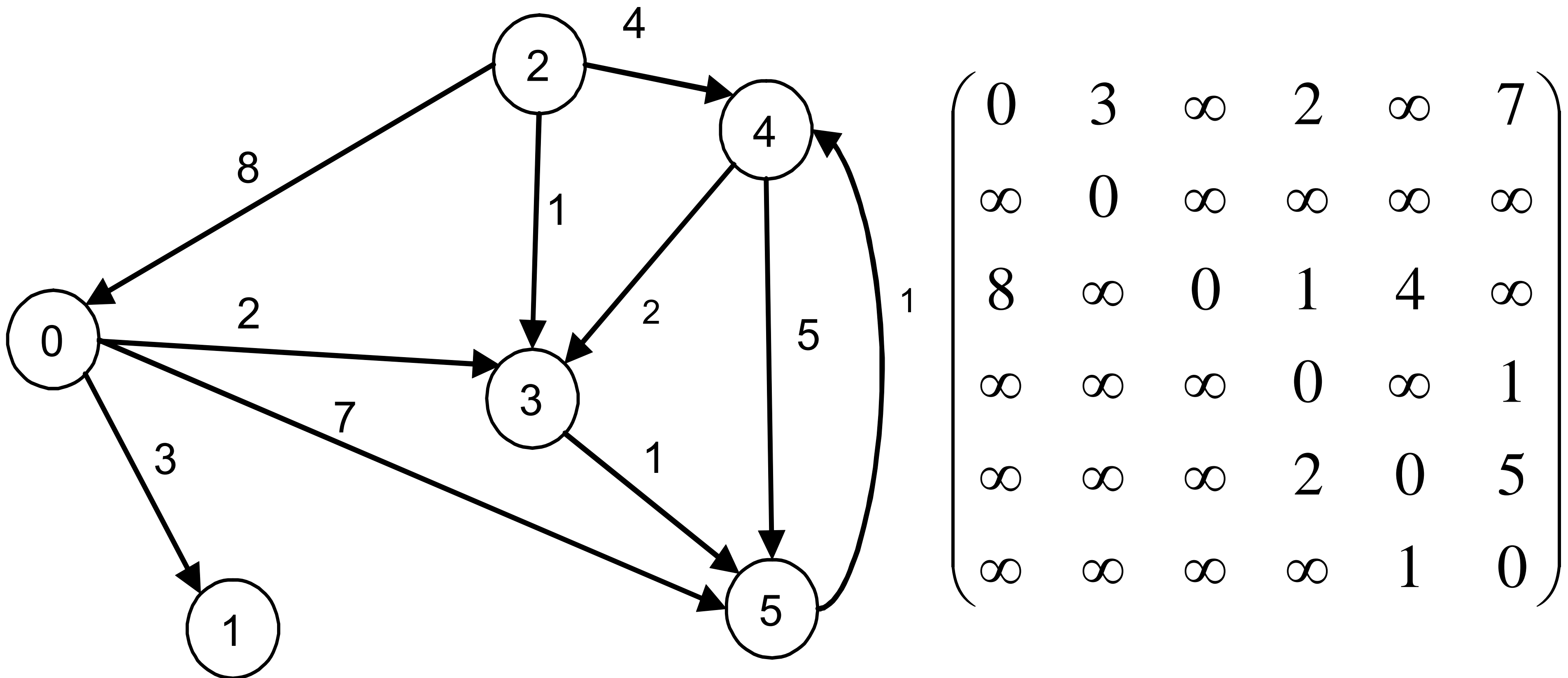
- При малом количестве дуг в графе целесообразно использовать для определения графов списки, перечисляющие имеющиеся в графах дуги
- Представление достаточно плотных графов, для которых почти все вершины соединены между собой дугами, может быть эффективно обеспечено при помощи *матрицы смежности*

$$A = (a_{ij}),\ 1 \le i, j \le n,\ a_{ij} = \begin{cases} w(v_i, v_j), & если\ (v_i, v_j) \in R, \\ 0, & если\ i = j, \\ \infty, & иначе. \end{cases}$$

© Гергель В.П.

Обработка графов

- Пример матрицы смежности



Упражнение 1 – Постановка задачи

- **Задача поиска всех кратчайших путей**
  - Дан граф *G*, каждому ребру которого приписан неотрицательный вес,
  - Граф полагаем ориентированным,
  - Для имеющегося графа *G* требуется найти минимальные длины путей между каждой парой вершин графа,
  - В качестве метода, решающего задачу поиска кратчайших путей между всеми парами пунктов назначения, далее используется *алгоритм Флойда* (*Floyd*)

Задача поиска всех кратчайших путей…

- **Последовательный алгоритм Флойда**

```
// Алгоритм 11.1
// Последовательный алгоритм Флойда
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      A[i,j] = min(A[i,j],A[i,k]+A[k,j]);
```

Сложность алгоритма имеет порядок $O(n^3)$

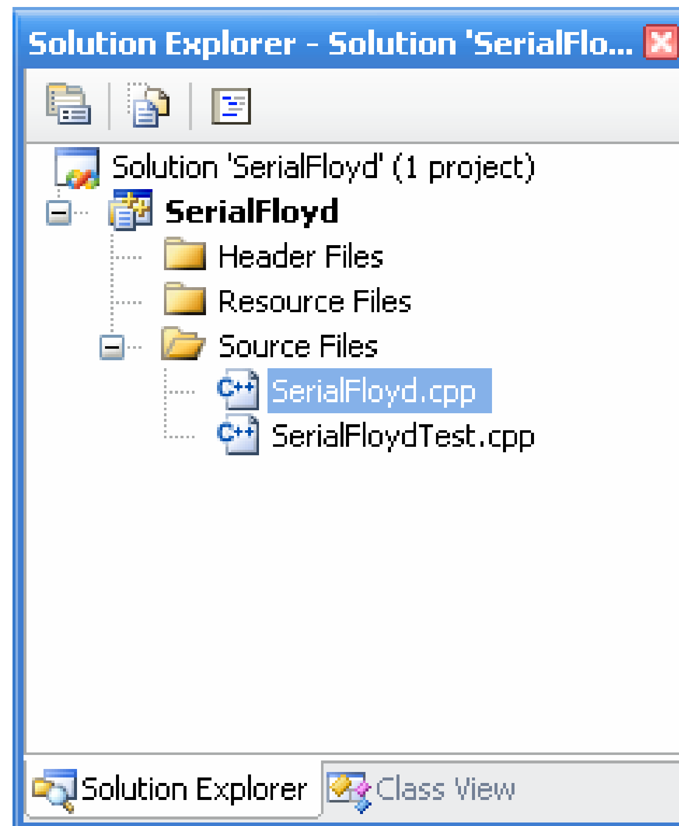
Упражнение 2 – Последовательный алгоритм Флойда: *этапы разработки*

Поэтапная разработка последовательного алгоритма:

- Задание 1 – Открытие проекта SerialFloyd
- Задание 2 – Ввод количества обрабатываемых вершин
- Задание 3 – Завершение процесса вычислений
- Задание 4 – Реализация алгоритма Флойда
- Задание 5 – Проведение вычислительных экспериментов

Задание 1 – Открытие проекта SerialFloyd

- Откройте проект **SerialFloyd**
- Откройте файл **SerialFloyd.cpp**



Задание 2 – Ввод количества обрабатываемых вершин…

Функция *ProcessInitialization*:

- инициализация переменных,
- ввод размера матрицы смежности,
- выделение памяти для матрицы смежности,
- заполнение памяти начальными значениями

```
// Function for process initialization
void ProcessInitialiazation(int *&pMatrix,
  int& Size);
```

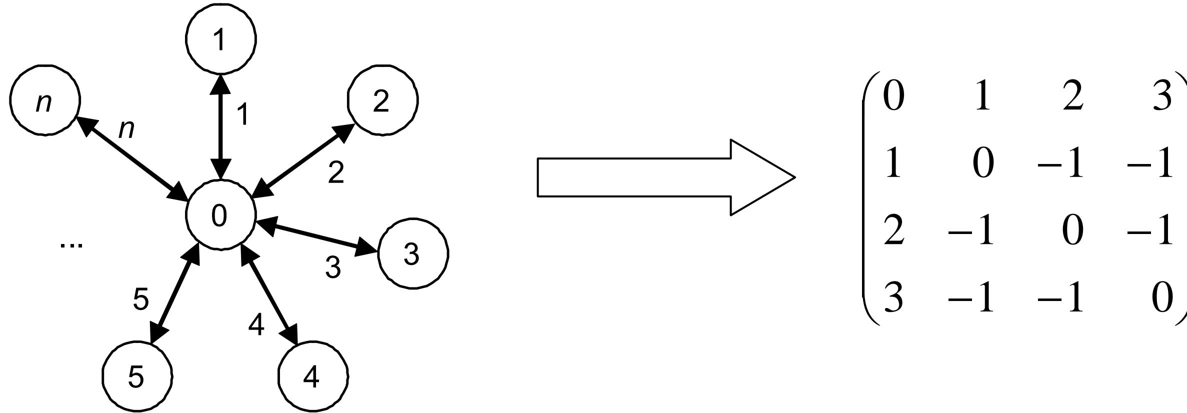
Выходные параметры:

- `pMatrix` – матрица смежности
- `Size` – размер матрицы смежности

Задание 2 – Ввод количества обрабатываемых вершин…

Функция *DummyDataInitialization* – инициализация данных

- Простое правило формирования данных:



$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & -1 & -1 \\ 2 & -1 & 0 & -1 \\ 3 & -1 & -1 & 0 \end{pmatrix}$$

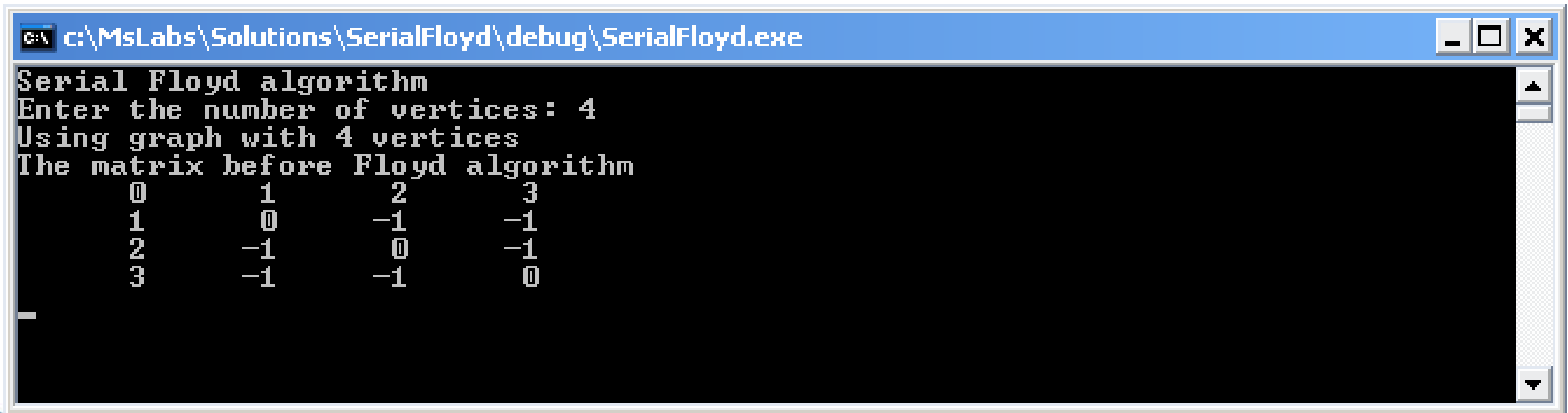
```
// Function for simple setting the initial data
void DummyDataInitialization(int *pMatrix, int Size);
```

Функция *PrintData* – отладочная печать

```
// Function for formatted matrix output
void PrintMatrix(int *pMatrix,int RowCount,int ColCount);
```

Задание 2 – Ввод количества обрабатываемых вершин

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



Задание 3 – Завершение процесса вычислений…

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

```
// Function for computational process termination
void ProcessTermination(int *pMatrix);
```

Входные параметры:

- `pMatrix` – матрица смежности

Задание 3 – Завершение процесса вычислений

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

© Гергель В.П.

Задание 4 – Реализация алгоритма Флойда…

Функция *SerialFloyd* – алгоритм Флойда

```
// Serial Floyd algorithm
void SerialFloyd(int *pMatrix, int Size);
Входные параметры:
- pMatrix - исходная матрица смежности
- Size    - размер матрицы смежности
```

Функция *Min* – выбор меньшего среди двух чисел с учетом представления бесконечного значения

```
int Min(int A, int B);
Входные параметры:
- A, B - числа, среди которых выбирается наименьшее
Выходное значение:
- наименьшее из чисел A и B
```

Задание 4 – Реализация алгоритма Флойда

- Реализуйте функции *SerialFloyd* и *Min*
- Добавьте отладочную печать в функцию *main*
- Протестируйте работоспособность приложения

c:\MsLabs\Solutions\SerialFloyd\debug\SerialFloyd.exe

```
Serial Floyd algorithm
Enter the number of vertices: 4
Using graph with 4 vertices
The matrix before Floyd algorithm
      0      1      2      3
      1      0     -1     -1
      2     -1      0     -1
      3     -1     -1      0
The matrix after Floyd algorithm
      0      1      2      3
      1      0      3      4
      2      3      0      5
      3      4      5      0
```

Задание 5 – Проведение вычислительных экспериментов…

Функция *RandomDataInitialization* – задание матрицы смежности при помощи датчика случайных чисел

```
// Function for random generating the initial data
void RandomDataInitialization(int *pMatrix,
  int Size);
Входные параметры:
- pMatrix - матрица смежности
- Size    - размер матрицы смежности
```

Задание 5 – Проведение вычислительных экспериментов

- Реализуйте функцию *RandomDataInitialization*
- Замените вызов функции *DummyDataInitialization* на *RandomDataInitialization* в функции *ProcessInitialization*
- Добавьте вычисление и вывод времени выполнения алгоритма Флойда
- Протестируйте работоспособность приложения
- Определите времена работы алгоритма Флойда при различных размерах матрицы смежности
- Вычислите время выполнения одной операции сравнения элементов матрицы смежности и рассчитайте теоретическое время работы
- Заполните таблицу результатов вычислений

Упражнение 3 – Разработка параллельного алгоритма Флойда…

- **Разделение вычислений на независимые части:**
  – Эффективный способ организации параллельных вычислений состоит в одновременном выполнении нескольких операций обновления значений матрицы *A*

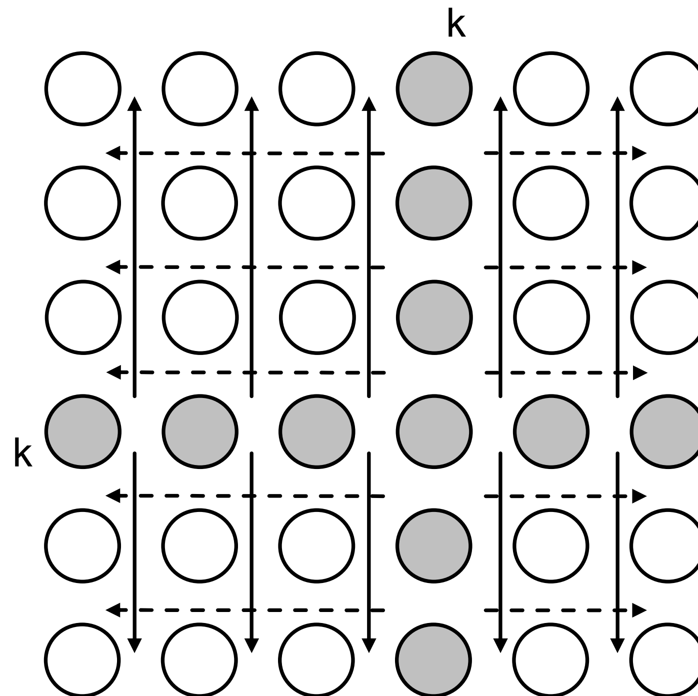
```
A[i,j] = min(A[i,j],A[i,k]+A[k,j]);
```

Упражнение 3 – Разработка параллельного алгоритма Флойда…

- **Выделение информационных зависимостей…**
  - Выполнение вычислений в подзадачах становится возможным только тогда, когда каждая подзадача (i,j) содержит необходимые для расчетов элементы a_{ij}, a_{ik}, a_{kj} матрицы A,
  - Каждый элемент a_{kj} строки k матрицы A должен быть передан всем подзадачам (k,j), $1 \le j \le n$, а каждый элемент a_{ik} столбца k матрицы A должен быть передан всем подзадачам (i,k), $1 \le i \le n$

Упражнение 3 – Разработка параллельного алгоритма Флойда…

- **Выделение информационных зависимостей:**
  – Информационная зависимость базовых подзадач (стрелками показаны направления обмена значениями на итерации *k*)



Упражнение 3 – Разработка параллельного алгоритма Флойда…

- **Масштабирование и распределение подзадач по процессорам:**
  - Возможный способ укрупнения вычислений состоит в использовании *ленточной схемы* разбиения матрицы *A*:
    - обновление элементов одной или нескольких строк (*горизонтальное* разбиение),
    - обновление элементов одной или нескольких строк (*вертикальное* разбиение)
  - Далее будем рассматривать только разбиение матрицы *A* на горизонтальные полосы

<u>Упражнение 4</u> – Параллельный алгоритм Флойда: *этапы разработки…*

Поэтапная разработка параллельного алгоритма:

- Задание 1 – Открытие проекта ParallelFloyd
- 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 – Разработка алгоритма Флойда

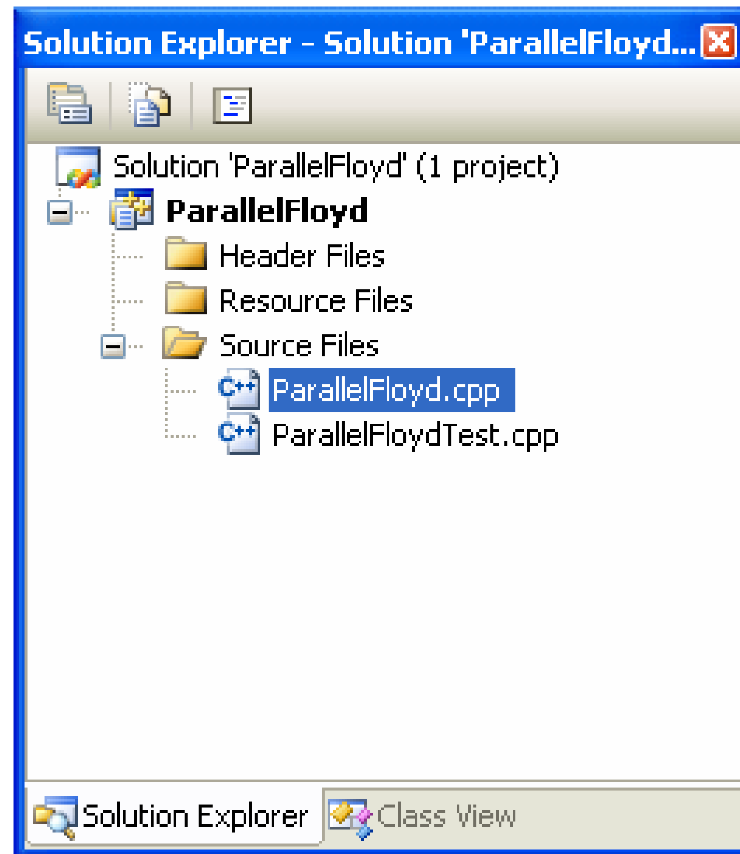
Упражнение 4 – Параллельный алгоритм Флойда: *этапы разработки*

- Задание 7 – Выполнение итераций алгоритма Флойда
- Задание 8 – Сбор результирующей матрицы
- Задание 9 – Проверка правильности работы программы
- Задание 10 – Реализация алгоритма Флойда для графов с произвольным количеством вершин
- Задание 11 – Проведение вычислительных экспериментов

© Гергель В.П.

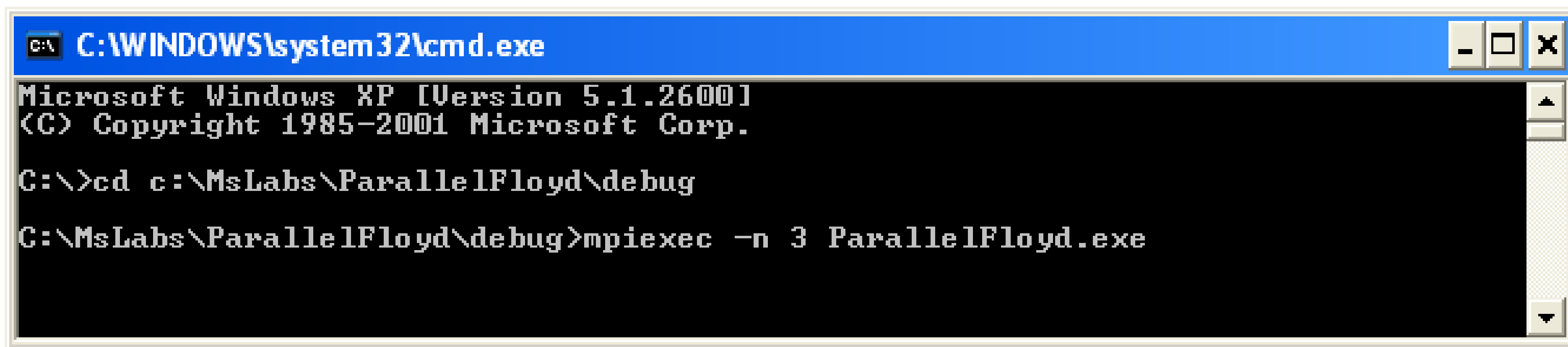
Задание 1 – Открытие проекта ParallelFloyd

- Откройте проект **ParallelFloyd**
- Откройте файл **ParallelFloyd.cpp**



Задание 2 – Инициализация и завершение параллельной программы

- Добавьте инициализацию среды выполнения MPI-программ
- Протестируйте работоспособность приложения
- Протестируйте запуск параллельной программы



Задание 3 – Ввод исходных данных…

Функция *ProcessInitialization*:

- инициализация переменных,
- ввод вершин в графе *ведущим процессом* (процессом с рангом 0),
- распределение количества вершин между процессами,
- выделение памяти для матрицы смежности на ведущем процессе,
- выделение памяти для полос матрицы на всех процессах

© Гергель В.П.

Задание 3 – Ввод исходных данных…

```
// Function for process initialization
void ProcessInitialiazation(int *&pMatrix,
  int *&pProcRows, int& Size, int& RowNum);
```

Выходные параметры:

- `pMatrix` – исходная матрица смежности на ведущем процессе
- `pProcRows` – полоса матрицы процесса
- `Size` – размер матрицы смежности
- `RowNum` – количество строк в полосе матрицы процесса

Задание 3 – Ввод исходных данных

- Реализуйте функцию *ProcessInitialization*
- Добавьте вызов функции *ProcessInintialization* 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(int *pMatrix,
  int *pProcRows);
```

Входные параметры:

- `pMatrix` – матрица смежности
- `pProcRows` – полоса матрицы смежности 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(int *pMatrix, int *pProcRows,
  int Size, int RowNum);
```

Входные параметры:

- pMatrix – матрица смежности для распределения
- Size – количество вершин в графе
- RowNum – размер полосы процесса

Выходные параметры:

- pProcRows – полоса матрицы смежности процесса

Задание 5 – Распределение данных между процессами…

Функция *TestDistribution* – проверка правильности выполнения распределения данных

```
// Function for testing the data distribution
void TestDistribution(int *pMatrix, int *pProcRows,
  int Size, int RowNum);
Входные параметры:
- pMatrix    - матрица смежности на ведущем процессе
- pProcRows  - полоса матрицы смежности процесса
- Size       - количество вершин в графе
- RowNum     - размер полосы 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:\MsLabs\ParallelFloyd\debug>mpiexec -n 3 ParallelFloyd.exe
Parallel Floyd algorithm
Enter the number of vertices: 6
Using graph with 6 vertices
Initial adjacency matrix:
        0       1       2       3       4       5
        1       0      -1      -1      -1      -1
        2      -1       0      -1      -1      -1
        3      -1      -1       0      -1      -1
        4      -1      -1      -1       0      -1
        5      -1      -1      -1      -1       0
ProcRank = 0
Proc rows:
        0       1       2       3       4       5
        1       0      -1      -1      -1      -1
ProcRank = 1
Proc rows:
        2      -1       0      -1      -1      -1
        3      -1      -1       0      -1      -1
ProcRank = 2
Proc rows:
        4      -1      -1      -1       0      -1
        5      -1      -1      -1      -1       0

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

Задание 6 – Разработка алгоритма Флойда…

Функция *ParallelFloyd* – параллельный алгоритм Флойда

```
// Parallel Floyd algorithm
void ParallelFloyd(int *pProcRows, int Size,
  int RowNum);
```

Входные параметры:

- `pProcRows` – полоса матрицы смежности процесса
- `Size` – размер матрицы смежности
- `RowNum` – количество строк в полосе матрицы смежности

© Гергель В.П.

<u>Задание 6</u> – Разработка алгоритма Флойда…

Функция *RowDistribution* – распределение строки матрицы смежности среди всех процессов

```
// Function for row broadcasting
void RowDistribution(int *pProcRows, int Size,
  int RowNum, int k, int *pRow);
Входные параметры:
- pProcRows - полоса матрицы смежности процесса
- Size      - размер матрицы смежности
- RowNum    - количество строк в полосе матрицы
              смежности
- k         - номер строки для распределения
Выходные параметры:
- pRow - распределенная строка матрицы смежности
```

Задание 6 – Разработка алгоритма Флойда

- Реализуйте первую версию функции *ParallelFloyd* и функцию *RowDistribution*
- Добавьте вызов функции *ParallelFloyd* в функцию *main*
- Закомментируйте вызов функции *TestDistribution*
- Протестируйте работоспособность приложения

C:\WINDOWS\system32\cmd.exe

```
C:\MsLabs\ParallelFloyd\debug>mpiexec -n 3 ParallelFloyd.exe
Parallel Floyd algorithm
Enter the number of vertices: 6
Using graph with 6 vertices
Row 0 after distribution:      0      1      2      3      4      5
Row 1 after distribution:      1      0     -1     -1     -1     -1
Row 2 after distribution:      2     -1      0     -1     -1     -1
Row 3 after distribution:      3     -1     -1      0     -1     -1
Row 4 after distribution:      4     -1     -1     -1      0     -1
Row 5 after distribution:      5     -1     -1     -1     -1      0

C:\MsLabs\ParallelFloyd\debug>
```

Задание 7 – Выполнение итераций алгоритма Флойда

- Добавьте в функцию *ParallelFloyd* выполнение итераций алгоритма Флойда
- Закомментируйте использовавшуюся ранее отладочную печать
- Протестируйте работоспособность приложения, добавив отладочную печать в функцию *main*

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

C:\MsLabs\ParallelFloyd\debug>mpiexec -n 3 ParallelFloyd.exe
Parallel Floyd algorithm
Enter the number of vertices: 6
Using graph with 6 vertices
ProcRank = 0
Proc rows:
      0      1      2      3      4      5
      1      0      3      4      5      6
ProcRank = 1
Proc rows:
      2      3      0      5      6      7
      3      4      5      0      7      8
ProcRank = 2
Proc rows:
      4      5      6      7      0      9
      5      6      7      8      9      0

C:\MsLabs\ParallelFloyd\debug>
```

Задание 8 – Сбор результирующей матрицы…

Функция *ResultCollection* – сбор результирующей матрицы на ведущем процессе

```
// Function for process result collection
void ResultCollection(int *pMatrix, int *pProcRows,
  int Size, int RowNum);
```

Входные параметры:

- pMatrix – массив для сбора результирующей матрицы
- pProcRows – полоса матрицы смежности процесса
- Size – размер матрицы смежности
- RowNum – количество строк в полосе матрицы смежности

Задание 8 – Сбор результирующей матрицы

- Реализуйте функцию *ResultCollection*
- Добавьте вызов функции *ResultCollection* в функцию *main*
- Закомментируйте использовавшуюся ранее отладочную печать
- Протестируйте работоспособность приложения используя функцию *PrintMatrix* на ведущем процессе

Задание 9 – Проверка правильности работы программы…

Функция *TestResult* – сравнение результатов последовательного и параллельного алгоритмов

```
// Testing the result of parallel Floyd algorithm
void TestResult(int *pMatrix, int *pSerialMatrix,
  int Size);
Входные параметры:
- pMatrix       - результат работы параллельного
                  алгоритма
- pSerialMatrix - исходная матрица смежности для
                  обработки последовательным
                  алгоритмом Флойда
- Size          - размер матрицы смежности
```

Задание 9 – Проверка правильности работы программы

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

© Гергель В.П.

Задание 10 – Реализация алгоритма Флойда для графов с произвольным количеством вершин

- Внесите необходимые изменения в функции *ProcessInitialization*, *DataDistribution*, *ResultCollection*, *RowDistribution*
- Проверьте правильность работы алгоритма Флойда при помощи функции *TestResult*

Задание 11 – Проведение вычислительных экспериментов

- Добавьте в функцию *main* вычисление времени выполнения параллельной программы
- Протестируйте работоспособность приложения
- Проведите вычислительные эксперименты
- Определите времена работы алгоритма Флойда при различных количествах вершин в графах и различном числе процессов
- Вычислите ускорение вычислений, получаемое за счет распараллелливания
- Вычислите теоретическое время работы параллельного алгоритма
- Заполните таблицу результатов вычислений

Заключение

- Рассмотрен алгоритм решения типовой задачи обработки графов - алгоритм Флойда
- Разработаны программы, реализующие последовательный и параллельный алгоритмы Флойда
- Проведены вычислительные эксперименты, проведено сравнение последовательного и параллельного алгоритмов

Темы заданий для самостоятельной работы

- Изучите другие параллельные алгоритмы обработки графов (алгоритм Прима для решения задачи о нахождении минимального охватывающего дерева, метод Дейкстры для решения задачи о нахождении кратчайших путей от одной из вершин графа до всех остальных)
- Разработайте программы, реализующие эти алгоритмы

© Гергель В.П.

Литература

- **Cormen, T.H., Leiserson, C. E. , Rivest, R. L. , Stein C.** (2001). Introduction to Algorithms, 2nd Edition. - The MIT Press. (русский перевод Кормен Т., Лейзерсон Ч., Ривест Р. (1999). Алгоритмы: построение и анализ. – М.: МЦНТО.)
- **Schloegel, K., Karypis, G., Kumar, V.** (2000). Graph Partitioning for High Performance Scientific Simulations.

Следующая тема

- Параллельные методы решения дифференциальных уравнений в частных производных

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)

О проекте

Microsoft

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.