**НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ**

**“КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”**

**ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**Параллельное программирование**

##### **Лабораторная работа №8**

Программирование для компьютерных систем с локальной памятью.  
Библиотека MPI

Выполнила:

студентка 3-го курса

группы ИВ-01

Наумова К.С.

Киев – 2013 г.

техническое задание

Разработать программу для решения в ПКС з ЛП (структура на рис. Рисунок 1) математической задачи: .

Библиотека: MPI.



Рисунок 1 – Структура ПКС з ЛП

выполнение работы

Этап 1. Разработка параллельного математического алгоритма



Этап 2. Разработка алгоритмов процессоров

Задача T0

1. Ввод данных.
2. Передать  каждой из задач .
3. Счет .
4. Принять результат  от каждой из задач .
5. Вывод результата *A*.

Задачи T1..TP-1

1. Принять  от задачи .
2. Передать  каждой из задач .
3. Счет .
4. Принять результат  от каждой из задач .
5. Передать результат  задаче .

Задачи T(P-1)1…T(P-1)k

1. Принять  от задачи .
2. Счет .
3. Передать результат  задаче .

.

Этап 3. Разработка структурной схемы взаимодействия задач

Структурная схема взаимодействий задач приведена на рис. Рисунок 2.



Рисунок 2 – Структурная схема взаимодействия задач

Этап 4. Разработка программы

1. // PRO lab8 MPI

2. // Naumova Kristina IO-01

3. // A = B + C(MO\*MX)alpha

4. // 04.05.2013

5.

6.

7. #include <mpi.h>

8. #include "data.h"

9.

10. //N=P: 1, 2, 3, 4, 5, 6

11. //p: 1, 2, 2, 2, 2, 2

12. //k: 0, 0, 1, 2, 3, 4

13.

14. //N = P = (p - 1) \* k + p

15. const int N = 6;

16. const int p = 2;

17. const int k = 4;

18. int P,

19. H;

20.

21. int \*getFirstRanks()

22. {

23. int \*firstRanks = new int[p];

24. int value = 1;

25. for(int i = 0; i < p; i++)

26. {

27. firstRanks[i] = value;

28. value += (k+1);

29. }

30. return firstRanks;

31. }

32.

33. bool isFirstRank(int rank)

34. {

35. int\* array = getFirstRanks();

36. for(int i = 0; i < p; i++)

37. {

38. if(rank == array[i])

39. {

40. return true;

41. }

42. }

43. return false;

44. }

45.

46. int getFirstRank(int secondRank)

47. {

48. int \*array = getFirstRanks();

49. for(int i = 0; i < p; i++)

50. {

51. if(secondRank > array[i] && secondRank < array[i+1])

52. {

53. return array[i];

54. }

55. }

56. }

57.

58.

59. int main(int args, char\* argv[])

60. {

61.

62. MPI\_Init(&args, &argv);

63. int rank;

64. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

65. MPI\_Comm\_size(MPI\_COMM\_WORLD, &P);

66. H = N / P;

67.

68. cout << "Task " << rank << " started" << endl;

69.

70. int alpha = 0, cols\_A\_B = H, rows\_MX = H;

71. if(rank == 0)

72. {

73. cols\_A\_B = N;

74. rows\_MX = N;

75. } else if(isFirstRank(rank))

76. {

77. cols\_A\_B = (k+1) \* H;

78. rows\_MX = (k+1) \* H;

79. }

80.

81. Vector A(cols\_A\_B);

82. Vector B(cols\_A\_B);

83. Vector C(N);

84. Matrix MO(N);

85. Matrix MX(rows\_MX, N);

86.

87.

88. // Ввод данных

89. if(rank == 0)

90. {

91. alpha = 1;

92. B.fill(1);

93. C.fill(1);

94. MO.fill(1);

95. MX.fill(1);

96. MX.transpose();

97. }

98.

99. if(rank == 0)

100. {

101. int firstRank = 1;

102. int firstAdress = H;

103.

104. // Передать alpha, MO, C, B\_(k+1)H, MX\_(k+1)H задачам T(1)..T(P-1)

105. for(int i = 1; i < p; i++)

106. {

107. MPI\_Send(&alpha, 1, MPI\_INT, firstRank, 0, MPI\_COMM\_WORLD);

108. MPI\_Send(MO.get\_adress(0), N\*N, MPI\_INT, firstRank, 0, MPI\_COMM\_WORLD);

109. MPI\_Send(C.get\_adress(0), N, MPI\_INT, firstRank, 0, MPI\_COMM\_WORLD);

110. MPI\_Send(B.get\_adress(firstAdress), (k+1)\*H, MPI\_INT, firstRank, 0, MPI\_COMM\_WORLD);

111. MPI\_Send(MX.get\_adress(firstAdress \* N), (k+1)\*H\*N, MPI\_INT, firstRank, 0, MPI\_COMM\_WORLD);

112. firstRank += (k+1);

113. firstAdress += (k+1)\*H;

114. }

115. } else if(isFirstRank(rank)) {

116. // Принять alpha, MO, C, B\_(k+1)H, MX\_(k+1)H от задачи T0

117. MPI\_Recv(&alpha, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

118. MPI\_Recv(MO.get\_adress(0), N\*N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

119. MPI\_Recv(C.get\_adress(0), N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

120. MPI\_Recv(B.get\_adress(0), (k+1)\*H, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

121. MPI\_Recv(MX.get\_adress(0), (k+1)\*H\*N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

122.

123. int secondRank = rank+1;

124.

125. // Передать alpha, MO, C, BH, MXH задачам T((P-1)1)..T((P-1)k)

126. for(int i = 0; i < k; i++)

127. {

128. MPI\_Send(&alpha, 1, MPI\_INT, secondRank, 0, MPI\_COMM\_WORLD);

129. MPI\_Send(MO.get\_adress(0), N\*N, MPI\_INT, secondRank, 0, MPI\_COMM\_WORLD);

130. MPI\_Send(C.get\_adress(0), N, MPI\_INT, secondRank, 0, MPI\_COMM\_WORLD);

131. MPI\_Send(B.get\_adress(H\*i+H), H, MPI\_INT, secondRank, 0, MPI\_COMM\_WORLD);

132. MPI\_Send(MX.get\_adress((H\*i+H)\*N), H\*N, MPI\_INT, secondRank, 0, MPI\_COMM\_WORLD);

133. ++secondRank;

134. }

135. }

136. else

137. {

138. // Принять alpha, MO, C, BH, MXH от задач T(1)..T(P-1)

139. MPI\_Recv(&alpha, 1, MPI\_INT, getFirstRank(rank), 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

140. MPI\_Recv(MO.get\_adress(0), N\*N, MPI\_INT, getFirstRank(rank), 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

141. MPI\_Recv(C.get\_adress(0), N, MPI\_INT, getFirstRank(rank), 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

142. MPI\_Recv(B.get\_adress(0), H, MPI\_INT, getFirstRank(rank), 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

143. MPI\_Recv(MX.get\_adress(0), H\*N, MPI\_INT, getFirstRank(rank), 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

144. }

145.

146. Vector A\_H(H);

147. // Счет AH = BH + C\*(MO\*MXH)\*alpha

148. calculation(0, H, alpha, B, C, MO, MX, A\_H);

149.

150. if(rank == 0)

151. {

152. // Принять AH от всех задач

153. MPI\_Gather(A\_H.get\_adress(0), H, MPI\_INT, A.get\_adress(0), H, MPI\_INT, 0, MPI\_COMM\_WORLD);

154. // Вывод результата A

155. A.output();

156. }

157. else

158. {

159. // Передать AH задаче T(0)

160. MPI\_Gather(A\_H.get\_adress(0), H, MPI\_INT, NULL, 0, MPI\_INT, 0, MPI\_COMM\_WORLD);

161. }

162.

163. cout << "Task " << rank << " finished" << endl;

164.

165. MPI\_Finalize();

166. if(rank == 0)

167. {

168. system("pause");

169. }

170.

171. return 0;

172. }

1. #pragma once

2.

3. #include "vector.h"

4. #include "matrix.h"

5.

6. void vector\_matrix\_multiply(const int start, const int end,

7. const Vector &B, const Matrix &MO, Vector &T);

8.

9. void calculation(const int start, const int end, const int alpha,

10. const Vector &B, const Vector &C, const Matrix &MO, const Matrix &MX,

11. Vector &A);

1. #include "data.h"

2.

3. void vector\_matrix\_multiply(const int start, const int end,

4. const Vector &B, const Matrix &MO, Vector &T)

5. {

6. int sum;

7. for (int i = start; i < end; i++)

8. {

9. sum = 0;

10. for (int j = 0; j < B.cols; j++)

11. {

12. sum += B.get(j) \* MO.get(i, j);

13. }

14. T.set(i, sum);

15. }

16. }

17.

18.

19. //AH = BH + C(MO\*MXH)alpha

20. //AH = T(MY\*MTH + alpha\*MXH)

21. void calculation(const int start, const int end, const int alpha,

22. const Vector &B, const Vector &C, const Matrix &MO, const Matrix &MX,

23. Vector &A)

24. {

25. long int sum1, sum2;

26.

27. for(int i = start; i < end; i++) {

28. sum2 = 0;

29. for(int j = 0; j < MO.cols; j++) {

30. sum1 = 0;

31. for(int k = 0; k < MO.cols; k++) {

32. sum1 += MX.get(i, k) \* MO.get(j, k) \* alpha;

33. }

34. sum2 += C.get(j) \* sum1;

35. }

36. sum2 += B.get(i);

37. A.set(i, sum2);

38. }

39. }

1. #pragma once

2.

3. #include <assert.h>

4. #include <iostream>

5. using namespace std;

6.

7. class Vector

8. {

9. public:

10. Vector(int cols);

11. ~Vector();

12.

13. void \* get\_adress(int element)

14. {

15. return this->data + element;

16. }

17.

18. long int get(int i) const

19. {

20. assert(i < this->cols);

21. return this->data[i];

22. }

23.

24. void set(int i, long int value)

25. {

26. assert(i < this->cols);

27. this->data[i] = value;

28. }

29.

30. void fill(long int value);

31. void output();

32.

33. const int cols;

34.

35. private:

36. long int \* data;

37. };

38.

1.

2.

3. #include "Vector.h"

4.

5.

6. Vector::Vector(int cols) :

7. cols(cols),

8. data(new long int[cols])

9. {

10. }

11.

12.

13. Vector::~Vector()

14. {

15. delete [] data;

16. }

17.

18. void Vector::fill(long int value)

19. {

20. for(int i = 0; i < this->cols; i++)

21. {

22. set(i, value);

23. }

24. }

25.

26. void Vector::output()

27. {

28. for(int i = 0; i < this->cols; i++)

29. {

30. cout << this->data[i] << " ";

31. }

32. }

1. #pragma once

2. #include "vector.h"

3.

4. class Matrix :

5. public Vector

6. {

7. public:

8. Matrix(int rows, int cols);

9. Matrix(int N);

10. Matrix(const Matrix &other);

11. ~Matrix();

12.

13. long int get(int i, int j) const

14. {

15. assert(i < this->rows);

16. assert(j < this->cols);

17. return Vector::get(i \* cols + j);

18. }

19.

20. void set(int i, int j, long int value)

21. {

22. assert(i < this->rows);

23. assert(j < this->cols);

24. Vector::set(i \* cols + j, value);

25. }

26.

27. void output();

28. void get\_column(int col, Vector &vector);

29. void transpose();

30.

31.

32. const int rows;

33. const int cols;

34.

35.

36. };

37.

1. #include "matrix.h"

2.

3.

4. Matrix::Matrix(int rows, int cols) :

5. Vector(rows \* cols),

6. rows(rows),

7. cols(cols)

8. {

9. }

10.

11.

12. Matrix::Matrix(int N) :

13. Vector(N \* N),

14. rows(N),

15. cols(N)

16. {

17. }

18.

19. Matrix::Matrix(const Matrix &other) :

20. Vector(other.rows \* other.cols),

21. rows(other.rows),

22. cols(other.cols)

23. {

24. for(int i = 0; i < this->rows; i++)

25. {

26. for(int j = 0; j < this->cols; j++)

27. {

28. Matrix::set(i, j, other.get(i, j));

29. }

30. }

31. }

32.

33.

34. Matrix::~Matrix()

35. {

36. }

37.

38.

39. void Matrix::output()

40. {

41. for(int i = 0; i < this->rows; i++)

42. {

43. for(int j = 0; j < this->cols; j++)

44. {

45. cout << Vector::get(i \* this->cols + j) << "\t";

46. }

47. cout << endl;

48. }

49. }

50.

51. void Matrix::get\_column(int col, Vector &vector)

52. {

53. assert(col < this->cols);

54. for(int i = 0; i < this->rows; i++)

55. {

56. Vector::set(i, Vector::get(i \* this->cols + col));

57. }

58. }

59.

60. void Matrix::transpose()

61. {

62. Matrix copy(\*this);

63. for(int i = 0; i < this->rows; i++)

64. {

65. for(int j = 0; j < this->cols; j++)

66. {

67. Matrix::set(j, i, copy.get(i, j));

68. }

69. }

70. }