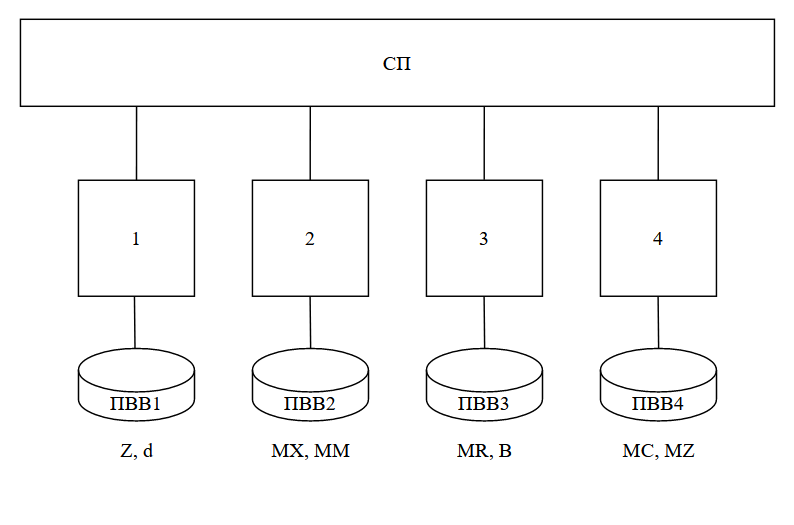
**Завдання**

Мова програмування: C++

Засоби організації взаємодії потоків: бар’єри, критичні секції OpenMP.

Математична задача: MX= (B\*Z)\*(MZ\*MM) - (MR\*MC)\*d

Структура ПКС:



**Розробка програми**

Етап 1. Побудова паралельного алгоритму

1. ai = (BH\*ZH), i=1..4

2. a = a+ai, i=1..4

3. MXH = a\*(MZ\*MMH) - (MR\*MCH)\*d

Спільні ресурси: a, а, MZ, MR, d.

Етап 2. Розробка алгоритмів

Бар‘єри,

Задача Т1 критичні ділянки

1. Бар’єр. Синхронізація вводу. Бар’єр

2. Копіювання MZ1 = MZ, MR1 = MR, d1 = d КД

3. Обчислення а1 = (BH\*ZH).

4. Обчислення а = a + a1. КД

5. Копіювання а1 = a. КД

6. Бар’єр. Синхронізація обчислень. Бар’єр

7. Обчислення MXH = a1\*(MZ1\*MMH) - (MR1\*MCH)\*d1

Задача Т2

1. Бар’єр. Синхронізація вводу. Бар’єр

2. Копіювання MB2 = MB, MC2 = MC, d2 = d КД

3. Обчислення а2 = (BH\*ZH).

4. Обчислення а = a + a2. КД

5. Копіювання a2 = a КД

6. Бар’єр. Синхронізація обчислень. Бар’єр

7. Обчислення MXH = a2\*(MZ2\*MMH) - (MR2\*MCH)\*d2

8. Бар’єр. Синхронізація обчислення MX. Бар’єр

9. Вивід MX.

Задача Т3

1. Бар’єр. Синхронізація вводу. Бар’єр

2. Копіювання MZ3 = MZ, MR3 = MR, d3 = d КД

3. Обчислення а3 = (BH\*ZH).

4. Обчислення а = a + a3. КД

5. Копіювання а3 = a. КД

6. Бар’єр. Синхронізація обчислень. Бар’єр

7. MXH = a3\*(MZ3\*MMH) - (MR3\*MCH)\*d3

Задача Т4

1. Бар’єр. Синхронізація вводу. Бар’єр

2. Копіювання MZ4 = MZ, MR4 = MR, d4 = d КД

3. Обчислення а4 = (BH\*ZH).

4. Обчислення а = a + a4. КД

5. Копіювання а4 = a. КД

6. Бар’єр. Синхронізація обчислень. Бар’єр

7. MXH = a4\*(MZ4\*MMH) - (MR4\*MCH)\*d4

Етап 3. Розробка структурної схеми взаємодії потоків

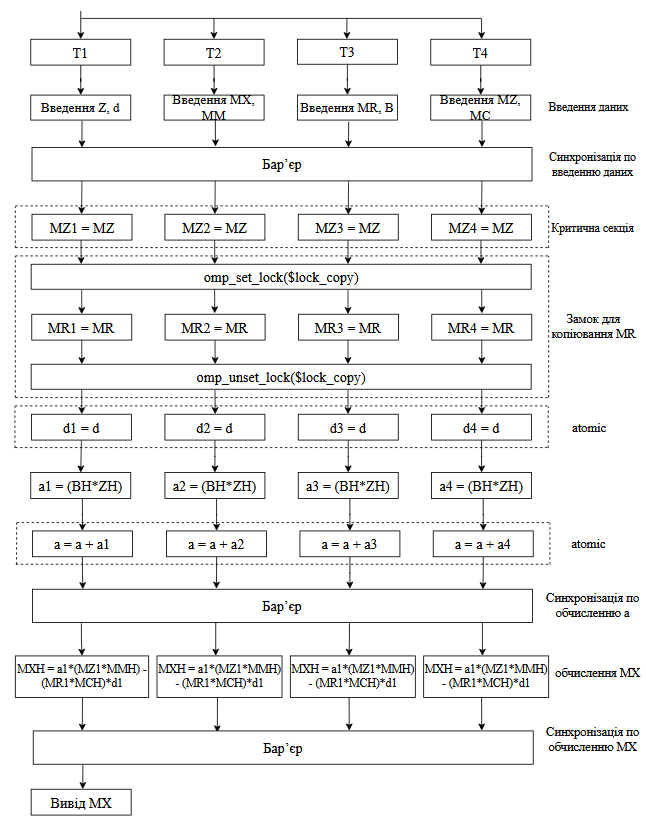
бар’єри для синхронізації потоків.

директива atomic для обчислення a;

директива atomic для забезпечення послідовного доступу до спільного ресурсу d.

директива critical для керування доступом до спільного ресурсу MZ;

замок lock\_copy для керування доступом до спільного ресурсу MR.



Етап 4. Лістинг програми

1. /\*\*-- Lab #3. OpenMP

2. \* -- MX= (B\*Z)\*(MZ\*MM) - (MR\*MC)\*d

3. \* --

4. \* -- Date

5. \*/

6. #include <omp.h>

7. #include "Data.h"

8. #include <iostream>

9. #include <windows.h>

10. // Розмір стеку

11. #pragma comment(linker, "/stack:160000000")

12. const int P = 4;

13. int N = 8;

14. int main(){

15. int a = 0;

16. int d = 0;

17. vector B = new int[N];

18. vector Z = new int[N];

19. matrix MZ;

20. matrix MM;

21. matrix MR;

22. matrix MC;

23. matrix MX;

24. const int H = N / P;

25. void cs();

26. omp\_lock\_t lock\_copy;

27. omp\_init\_lock(&lock\_copy);

28. std::cout << "Lab4 started.\n" << std::endl;

29. omp\_set\_num\_threads(P);

30. #pragma omp parallel

31. {

32. int id = omp\_get\_thread\_num();

33. std::cout << "T" << id << " started.\n" << std::endl;

34. //---------------------------- Ввід даних ---------------------------

35. switch (id)

36. {

37. case 0:

38. Z = VectorFill(1);

39. d = 1;

40. break;

41. case 1:

42. MM = MatrixFill(1);

43. break;

44. case 2:

45. MR = MatrixFill(1);

46. B = VectorFill(1);

47. break;

48. case 3:

49. MZ = MatrixFill(1);

50. MC = MatrixFill(1);

51. break;

52. }

53. //-------------Бар'єр для синхронізації введення--------

54. #pragma omp barrier

55. int ai = 0;

56. for (int i = id \* H; i < (id + 1) \* H; i++)

57. ai += B[i] \* Z[i];

58.

59. // Обчислення a

60. #pragma omp\_atomic

61. {

62. a += ai;

63. }

64. //-------------Бар'єр. Синхронізація обчислення а-------

65. #pragma omp barrier

66. int di = 0;

67. matrix MZi = new vector[N];

68. matrix MRi = new vector[N];

69. // atomic для копіювання a

70. #pragma omp\_atomic

71. {

72. ai = a;

73. }

74. // atomic для копіювання d

75. #pragma omp\_atomic

76. {

77. di = d;

78. }

79. // Критична секція копіювання MZ

80. #pragma omp critical(cs)

81. {

82. MZi = MatrixCopy(MZ);

83. }

84. // Замок копіювання MR

85. omp\_set\_lock(&lock\_copy);

86. MRi = MatrixCopy(MR);

87. omp\_unset\_lock(&lock\_copy);

88. // Обчислення MX = a4\*(MZ4\*MMH) - (MR4\*MCH)\*d4

89. int notDiv;

90. int unDiv;

91. for (int i = id \* H; i < (id + 1) \* H; i++) {

92. for (int j = 0; j < N; j++) {

93. notDiv = 0;

94. unDiv = 0;

95. for (int k = 0; k < N; k++){

96. notDiv += MZi[i][k] \* MM[k][j];

97. unDiv += MRi[i][k] \* MC[k][j];

98. MX[i][k] += ai\*notDiv - unDiv \* di;

99. }

00. }

101. }

102. //-------------Бар'єр. Синхронізація обчислення MX-------

103. #pragma omp barrier

104. if (id == 0 && N < 15) {

105. MatrixOutput(MX, 'MX');

106. }

107. Sleep(100);

108. std::cout << "T" << id + 1 << " finished." << std::endl;

109. }

110. std::cout << "\nLab4 finished!\n" << std::endl;

111. getchar();

112. return 0;

113. }

Data.cpp

1.#include <windows.h>

2.#include <iostream>

3.#include "Data.h"

4.vector VectorFill() {

5. vector ves = new int[N];

6. for (int i = 0; i < N; i++)

7. ves[i] = 1;

8. return ves;

9.}

10.matrix MatrixFill() {

11. matrix flynn = new vector[N];

12. for (int i = 0; i < N; i++)

13. flynn[i] = new int[N];

14. for (int i = 0; i < N; i++)

15. for (int j = 0; j < N; j++)

16. flynn[i][j] = 1;

17. return flynn;

18.}

19.matrix MatrixCopy(matrix mat) {

20. matrix result = new vector[N];

21. for (int i = 0; i < N; i++)

22. result[i] = new int[N];

23. for (int i = 0; i < N; i++)

24. for (int j = 0; j < N; j++)

25. result[i][j] = mat[i][j];

26. return result;

27.}

28.void MatrixOutput(matrix MA, char name) {

29. std::cout << name << ": " << std::endl;

30. for (int i = 0; i < N; i++) {

31. for (int j = 0; j < N; j++) {

32. std::cout << MA[i][j] << " " << std::endl;

33. }

34. }

35. std::cout << "\n" << std::endl;

36.}

Data.h

1.typedef int\* vector;

2.typedef int\*\* matrix;

3.extern int N;

4.vector VectorFill(int);

5.matrix MatrixFill(int);

6.void MatrixOutput(matrix, char);

7.matrix MatrixCopy(matrix);