**Міністерство освіти і науки України**

**Національний університет "Львівська політехніка"**

**Кафедра ЕОМ**



**Звіт з лабораторної роботи №1**

**з дисципліни “** **Паралельні та розподілені обчислення ”**

**на тему: ” ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНОЇ ДЕКОМПОЗИЦІЇ ДЛЯ РОЗВ’ЯЗКУ ОБЧИСЛЮВАЛЬНИХ ЗАДАЧ ”**

**Виконав: студент .гр. КІ-33**

**Кіндій В. А.**

**Прийняв: викладач**

**Козак Н. Б.**

**Львів 2020 р.**

**Мета роботи:** Вивчити методи декомпозицій задач. Набути навиків розв’язування задач з використанням функціональної декомпозиції.

**Завдання:**

1) Проаналізувати, наведені нижче, правила обрахунку елементів виразу;

2) Провести декомпозицію задачі, виходячи з можливості паралельного виконання окремих підзадач;

3) Об’єднати отримані проміжні результати і обрахувати кінцевий вираз;

4) Написати і продемонструвати програму обчислення виразу;

5) Визначити паралелізм якого рівня присутній в алгоритмі та зробити висновки щодо залежностей даних, керування, ресурсів, вводу/виводу;

6) Скласти звіт про виконану роботу.

Варіант – 5

означає операцію транспонування; i,j=1…n (n – вхідна розмірність)

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | число | | |
| bi=5i3 | A1(5b1-c1) | A2(B2+10C2)  Cij=1/(i2+j) |

**Хід роботи:**

1. Аналізую правила обрахунку елементів виразу
2. Проводжу декомпозиції задачі виходячи з можливості паралельного виконання окремих підзадач

Схема проведення декомпозиції зображена на рисунку 1.

1. Обєдную отримані проміжні результати і обраховую кінцевий вираз.
2. Пишу програму обчислення виразу.

Програма обчислення виразу представлена в додатку А.

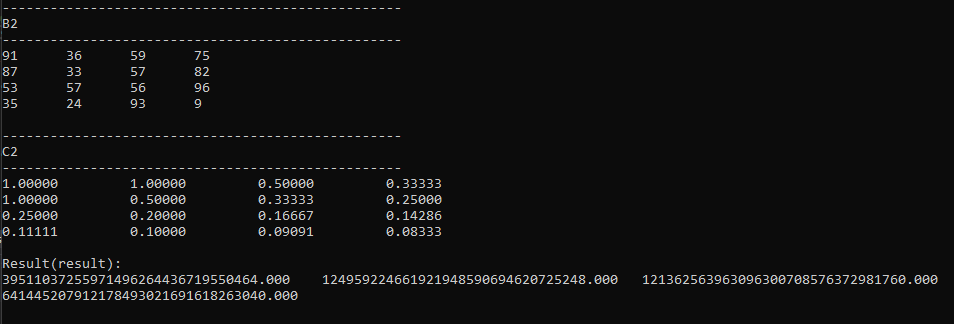


Рис 1. Приклад виконання програми

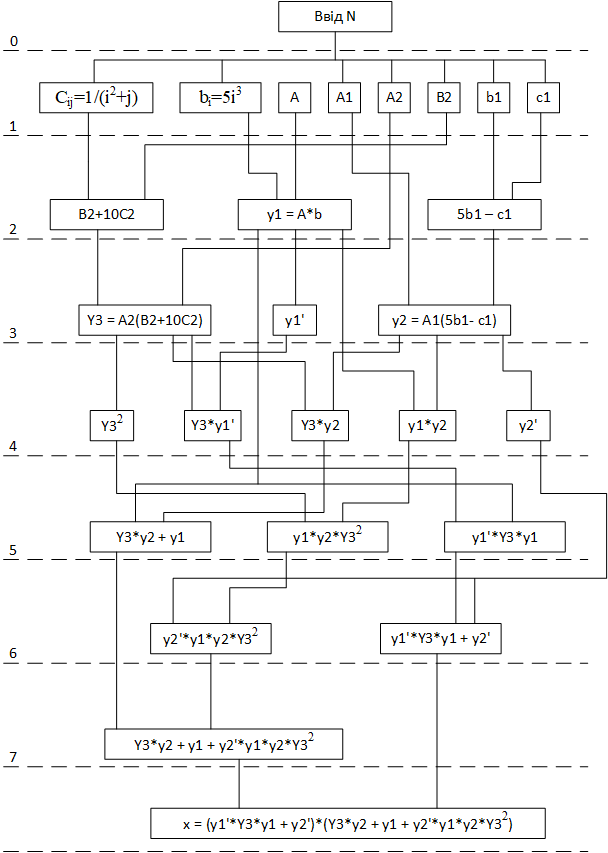


Рис 2. Схема проведення декомпозиції

**Висновок**: вивчив методи декомпозицій задач. Набув навиків розв’язування задач з використанням функціональної декомпозиції.

**Додаток А (програма обчислення виразу)**

|  |
| --- |
| #include <iostream>  #include <vector>  #include <ctime>  #include <iomanip>  using namespace std;  // Add matrix to matrix -> return result matrix  vector<vector<float>> MatrixAddMatrix(vector<vector<float>> a, vector<vector<float>> b, int size) {  vector<vector<float>> res;  res.resize(size);  for (int i = 0; i < size; ++i) {  res[i].resize(size);  }  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  res[i][j] = (a[i][j] + b[i][j]);  }  }  return res;  }  // Substract matrix from matrix -> return result matrix  vector<vector<float>> MatrixSubMatrix(vector<vector<float>> a, vector<vector<float>> b, int size) {  vector<vector<float>> res;  res.resize(size);  for (int i = 0; i < size; ++i) {  res[i].resize(size);  }  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  res[i][j] = (a[i][j] - b[i][j]);  }  }  return res;  }  // Multiply matrix to matrix -> return result matrix  vector<vector<float>> MatrixMulMatrix(vector<vector<float>> a, vector<vector<float>> b, int size) {  vector<vector<float>> res;  res.resize(size);  for (int i = 0; i < size; ++i) {  res[i].resize(size);  }  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  res[i][j] = 0;  for (int k = 0; k < size; k++) {  res[i][j] += a[i][k] \* b[k][j];  }  }  }  return res;  }  // vector add vector  vector<float> VectorAddVector(vector<float> a, vector<float> b, int size) {  vector<float> res;  res.resize(size);  for (int i = 0; i < size; i++) {  res[i] = a[i] + b[i];  }  return res;  }  // vector mul on vector  vector<float> VectorMulVector(vector<float> a, vector<float> b, int size) {  vector<float> res;  res.resize(size);  for (int i = 0; i < size; i++) {  res[i] = a[i] \* b[i];  }  return res;  }  // matrix mul on vector  vector<float> MatrixMulVector(vector<vector<float>> a, vector<float> b, int size) {  vector<float> res;  res.resize(size);  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  res[i] = (a[i][j] \* b[j]);  }  }  return res;  }  // transosition of matrix  vector<float> VectorTransposition(vector<float> a, int size) {  vector<float> res;  res = a;  res.reserve(size);  return res;  }  // mul matrix on number  vector<vector<float>> NumberMulMatrix(vector<vector<float>> a,float number, int size) {  vector<vector<float>> res;  res.resize(size);  for (int i = 0; i < size; ++i) {  res[i].resize(size);  }  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  res[i][j] = a[i][j] \* number;  }  }  return res;  }  // mul vector on number  vector<float> NumberMulVector(vector<float> a, float number, int size) {  vector<float> res;  res.resize(size);  for (int i = 0; i < size; i++) {  res[i] = a[i] \* number;  }  return res;  }  int main() {  #pragma region Array declaring section, size input  srand(time(NULL));  int n, i, j, k, choice = 0;  vector<float> result;  vector<float> b, c1, b1, x, y1, y2;  vector<vector<float>> A, A1, A2, B2, C2, Y3;  // enter size bigger than 2  cout << "x = (y1'\*Y3\*y1+y2')\*(Y3\*y2+y1+y1\*y2'\*Y3^2\*y2)" << endl << endl;  m:  cout << "Enter size(n):";  cin >> n;  if (n < 3) { cout << "n must be >= 3" << endl; goto m; }  #pragma endregion  #pragma region Array init section  result.resize(n); x.resize(n); y1.resize(n); y2.resize(n); Y3.resize(n); b.resize(n); b1.resize(n); c1.resize(n);  A.resize(n);  for (i = 0; i < n; i++) {  A[i].resize(n);  }  A1.resize(n);  for (i = 0; i < n; i++) {  A1[i].resize(n);  }  A2.resize(n);  for (i = 0; i < n; i++) {  A2[i].resize(n);  }  B2.resize(n);  for (i = 0; i < n; i++) {  B2[i].resize(n);  }  C2.resize(n);  for (i = 0; i < n; i++) {  C2[i].resize(n);  }  for (i = 0; i < n; i++) {  Y3[i].resize(n);  }  // calc b1  for (i = 0; i < n; i++) {  b[i] = 5 \* (pow(i, 3));  }  // calc Cij  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  if (i == 0 && j == 0) {  C2[i][j] = 1.0;  }  else {  C2[i][j] = 1.0 / ((i \* i) + j);  }  }  }  #pragma endregion  #pragma region Array input section  cout << endl << "1 Matrix manual input" << endl << "2 Matrix randomize" << endl << "Choice:";  cin >> choice;  if (choice == 1) {  for (i = 0; i < n; i++) {  cout << "b1[" << i + 1 << "]= ";  cin >> b1[i];  }  for (i = 0; i < n; i++) {  cout << "c1[" << i + 1 << "]= ";  cin >> c1[i];  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << "A[" << i + 1 << "][" << j + 1 << "]= ";  cin >> A[i][j];  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << "A1[" << i + 1 << "][" << j + 1 << "]= ";  cin >> A1[i][j];  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << "A2[" << i + 1 << "][" << j + 1 << "]= ";  cin >> A2[i][j];  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << "B2[" << i + 1 << "][" << j + 1 << "]= ";  cin >> B2[i][j];  }  }  }  else if (choice == 2) {  for (i = 0; i < n; i++) {  b1[i] = rand() % 100;  }  for (i = 0; i < n; i++) {  c1[i] = rand() % 100;  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  A[i][j] = rand() % 100;  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  A1[i][j] = rand() % 100;  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  A2[i][j] = rand() % 100;  }  }  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  B2[i][j] = rand() % 100;  }  }  }  #pragma endregion  #pragma region Print array section  cout << "--------------------------------------------------" << endl;  //--------------------------------------array out start  cout << "b" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  cout << b1[i] << "\t";  }  cout << endl << endl;  cout << "--------------------------------------------------" << endl;  cout << "b1" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  cout << b1[i] << "\t";  }  cout << endl << endl;  cout << "--------------------------------------------------" << endl;  cout << "c1" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  cout << c1[i] << "\t";  }  cout << endl << endl;  cout << "--------------------------------------------------" << endl;  cout << "A" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << A[i][j] << "\t";  }  cout << endl;  }  cout << endl;  cout << "--------------------------------------------------" << endl;  cout << "A1" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << A1[i][j] << "\t";  }  cout << endl;  }  cout << endl;  cout << "--------------------------------------------------" << endl;  cout << "A2" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << A2[i][j] << "\t";  }  cout << endl;  }  cout << endl;  cout << "--------------------------------------------------" << endl;  cout << "B2" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << B2[i][j] << "\t";  }  cout << endl;  }  cout << endl;  cout << "--------------------------------------------------" << endl;  cout << "C2" << endl;  cout << "--------------------------------------------------" << endl;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  cout << fixed << setprecision(5) << C2[i][j] << setw(3) << "\t";  }  cout << endl;  }  cout << endl;  //------------------------------------------array out end  #pragma endregion  #pragma region Calulating section  //-----------------------------------------calculating start  // 1 - level start  // temp\_1\_mat\_1 = B2+10C2  vector<vector<float>> temp\_1\_mat\_1 = MatrixAddMatrix( NumberMulMatrix(C2, 10, n), B2, n );  // y1 = A\*b  y1 = MatrixMulVector(A, b, n);  // temp\_1\_vec\_2 = 5b1 - c1  vector<float> temp\_1\_vec\_2 = VectorAddVector(NumberMulVector(b1, 5, n), NumberMulVector(c1, -1, n), n);  // 1 - level end  // 2 - level start  // temp\_2\_vec\_1 = y1'  vector<float> temp\_2\_vec\_1 = VectorTransposition(y1, n);  // Y3 = A2 (B2 + 10C2)  Y3 = MatrixMulMatrix(A2, temp\_1\_mat\_1, n);  // y2 = A1(5b1 - c1)  y2 = MatrixMulVector(A1, temp\_1\_vec\_2, n);  // 2 - level end  // 3 - level start  // Y3^2  vector<vector<float>> Y3\_square = MatrixMulMatrix(Y3, Y3, n);  // temp\_3\_vec\_1 = Y3\*y1'  vector<float> temp\_3\_vec\_1 = MatrixMulVector(Y3, temp\_2\_vec\_1, n);  // temp\_3\_vec\_2 = Y3\*y2  vector<float> temp\_3\_vec\_2 = MatrixMulVector(Y3, y2, n);  // temp\_3\_vec\_3 = y1 \* y2  vector<float> temp\_3\_vec\_3 = VectorMulVector(y1, y2, n);  // temp\_3\_vec\_4 = y2'  vector<float> temp\_3\_vec\_4 = VectorTransposition(y2, n);  // 3 - level end  // 4 - level start  // temp\_4\_vec\_1 = Y3\*y2 + y1  vector<float> temp\_4\_vec\_1 = VectorAddVector(temp\_3\_vec\_2, y1, n);  // temp\_4\_vec\_2 = y1\*y2\*Y3^2  vector<float> temp\_4\_vec\_2 = MatrixMulVector(Y3\_square, temp\_3\_vec\_3, n);  // temp\_4\_vec\_3 = y1'\*Y3\*y1  vector<float> temp\_4\_vec\_3 = VectorMulVector(temp\_3\_vec\_1, y1, n);  // 4 - level end  // 5 - level start  // temp\_5\_vec\_1 = y2'\*y1\*y2\*Y3^2  vector<float> temp\_5\_vec\_1 = VectorMulVector(temp\_4\_vec\_2, temp\_3\_vec\_4, n);  // temp\_5\_vec\_2 = y1'\*Y3\*y1 + y2'  vector<float> temp\_5\_vec\_2 = VectorAddVector(temp\_4\_vec\_3, temp\_3\_vec\_4, n);  // 5 - level end  // 6 - level start  // temp\_6\_vec\_1  vector<float> temp\_6\_vec\_1 = VectorAddVector(temp\_5\_vec\_1, temp\_4\_vec\_1, n);  // 6 - level end  // 7 - level start  result = VectorMulVector(temp\_6\_vec\_1, temp\_5\_vec\_2, n);  // 7 - level end  //-----------------------------------------calculating end  #pragma endregion  //result out start  cout << "Result(result): " << endl;  for (i = 0; i < n; i++) {  cout << fixed << setprecision(3) << result[i] << "\t";  }  cout << endl << endl;  //result out end  //-----------------------------------------check end  int r = 0;  cin >> r;  return 0;  } |