МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №1 по дисциплине «Параллельное программирование»

**Исследование особенностей реализации классического алгоритма умножения матриц**

|  |  |
| --- | --- |
| Выполнил: студент группы ФИб-4302-51-00 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПМиИ | / В.А. Бызов / |

Киров 2021

Задание 1

Создать две целочисленные матрицы размера N × N с использованием двумерных динамических массивов. Реализовать классический алгоритм умножения матриц. Провести тестирование программы на матрицах размерности N = 512, 1024 и 2048 без оптимизации (с ключом –O0 или –Od, в конфигурации Debug в MS Visual Studio) и с ключом оптимизации –O2 (конфигурация Release в MS Visual Studio). На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сделать выводы.

Код:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <chrono>  using namespace std;  const int DIMM = 2048;  int\*\* A = new int\* [DIMM];  int\*\* B = new int\* [DIMM];  int\*\* C = new int\* [DIMM];  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void init()  {  srand(time(0));  for (int i = 0; i < DIMM; i++)  {  A[i] = new int[DIMM];  B[i] = new int[DIMM];  C[i] = new int[DIMM];  for (int j = 0; j < DIMM; j++)  {  A[i][j] = rand() % 25;  B[i][j] = rand() % 25;  C[i][j] = 0;  }  }  }  void display(int\*\* A)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  cout << A[i][j] << " ";  }  cout << "" << endl;  }  cout << "" << endl;  }  void matr(int\*\* A, int\*\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  C[i][j] += A[i][k] \* B[k][j];  }  }  }  }  int main()  {  init();  start = chrono::high\_resolution\_clock::now();  matr(A, B);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time: " << diff.count() << endl;  delete[]A;  delete[]B;  delete[]C;  } |

Таблица 1 – Время выполнения алгоритма умножения матриц с двойными указателями

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Ключ оптимизации | |
| -O0 | -O2 |
| 512 | 0,753333333 | 0,373333333 |
| 1024 | 3,873333333 | 1,96 |
| 2048 | 28,31 | 12,46 |

Вывод: Время работы приложения с ключом оптимизации -O2 уменьшилось примерно в 2 раза.

Задание 2

Создать две целочисленные матрицы размера N × N с использованием одномерных динамических массивов. Реализовать классический алгоритм умножения матриц. Провести тестирование программы на матрицах размерности N = 512, 1024 и 2048 с ключом оптимизации –O2 (конфигурация Release в MS Visual Studio). На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сравнить полученные результаты. Сделать выводы.

Код:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <chrono>  using namespace std;  const int DIMM = 2048;  int\* A1 = new int[DIMM \* DIMM];  int\* B1 = new int[DIMM \* DIMM];  int\* C1 = new int[DIMM \* DIMM];  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void init2()  {  srand(time(0));  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  A1[DIMM \* i + j] = rand() % 25;  B1[DIMM \* i + j] = rand() % 25;  C1[DIMM \* i + j] = 0;  }  }  }  void display1(int\* A)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  cout << A[DIMM \* i + j] << " ";  }  cout << "" << endl;  }  cout << "" << endl;  }  void matr1(int\* A, int\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  int main()  {  init2();  start = chrono::high\_resolution\_clock::now();  matr1(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time: " << diff.count() << endl;  delete[]A1;  delete[]B1;  delete[]C1;  } |

Таблица 2 – Время выполнения алгоритма умножения матриц с одинарными и двойными указателями

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Способ хранения матрицы в памяти | |
| двойные указатели | одинарные указатели |
| 512 | 0,373333333 | 0,203333333 |
| 1024 | 1,96 | 1,27 |
| 2048 | 12,46 | 6,516666667 |

Вывод: Время работы приложения с использованием одинарных указателей уменьшилось примерно в 1.75 раза.

Задание 3

Для матриц, хранящихся в одномерных динамических массивах, реализовать классический алгоритм умножения со всеми возможными перестановками порядка циклов. Провести тестирование программ на матрицах размерности N = 512, 1024 и 2048. На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах. Сравнить результаты. Сделать выводы.

Код:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <chrono>  using namespace std;  const int DIMM = 2048;  int\* A1 = new int[DIMM \* DIMM];  int\* B1 = new int[DIMM \* DIMM];  int\* C1 = new int[DIMM \* DIMM];  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void init2()  {  srand(time(0));  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  A1[DIMM \* i + j] = rand() % 25;  B1[DIMM \* i + j] = rand() % 25;  C1[DIMM \* i + j] = 0;  }  }  }  void display1(int\* A)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  cout << A[DIMM \* i + j] << " ";  }  cout << "" << endl;  }  cout << "" << endl;  }  void kij(int\* A, int\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  void ijk(int\* A, int\* B)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  for (int k = 0; k < DIMM; k++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  void ikj(int\* A, int\* B)  {  for (int i = 0; i < DIMM; i++)  {  for (int k = 0; k < DIMM; k++)  {  for (int j = 0; j < DIMM; j++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  void jik(int\* A, int\* B)  {  for (int j = 0; j < DIMM; j++)  {  for (int i = 0; i < DIMM; i++)  {  for (int k = 0; k < DIMM; k++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  void jki(int\* A, int\* B)  {  for (int j = 0; j < DIMM; j++)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  void kji(int\* A, int\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int j = 0; j < DIMM; j++)  {  for (int i = 0; i < DIMM; i++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  int main()  {  int h = 0;  while (h<3)  {  cout << endl;  {  init2();  start = chrono::high\_resolution\_clock::now();  ijk(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time ijk: " << diff.count() << endl;  }  {  init2();  start = chrono::high\_resolution\_clock::now();  ikj(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time ikj: " << diff.count() << endl;  }  {  init2();  start = chrono::high\_resolution\_clock::now();  jik(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time jik: " << diff.count() << endl;  }  {  init2();  start = chrono::high\_resolution\_clock::now();  jki(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time jki: " << diff.count() << endl;  }  {  init2();  start = chrono::high\_resolution\_clock::now();  kij(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time kij: " << diff.count() << endl;  }  {  init2();  start = chrono::high\_resolution\_clock::now();  kji(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time kji: " << diff.count() << endl;  }  h++;  }  delete[]A1;  delete[]B1;  delete[]C1;  } |

Таблица 3 – Время выполнения классического алгоритма умножения матриц

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 512 | 0,267317333 | 0,086795433 | 0,251307 | 0,697008667 | 0,1160648 | 0,740291 |
| 1024 | 17,6773 | 1,13226 | 13,15626667 | 33,02856667 | 1,422153333 | 34,10643333 |
| 2048 | 114,243 | 8,329586667 | 128,4096667 | 275,944 | 9,138473333 | 317,607 |

Вывод: Наилучшую асимптотику имеет порядок циклов ikj.

Задание 4

Выполнить задания 1-2 с использованием представления матриц в виде одномерных динамических массивов.

Код:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <chrono>  using namespace std;  const int DIMM = 2048;  int\* A1 = new int[DIMM \* DIMM];  int\* B1 = new int[DIMM \* DIMM];  int\* C1 = new int[DIMM \* DIMM];  chrono::time\_point<chrono::high\_resolution\_clock>start, ennd;  void init2()  {  srand(time(0));  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  A1[DIMM \* i + j] = rand() % 25;  B1[DIMM \* i + j] = rand() % 25;  C1[DIMM \* i + j] = 0;  }  }  }  void display1(int\* A)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  cout << A[DIMM \* i + j] << " ";  }  cout << "" << endl;  }  cout << "" << endl;  }  void matr1(int\* A, int\* B)  {  for (int k = 0; k < DIMM; k++)  {  for (int i = 0; i < DIMM; i++)  {  for (int j = 0; j < DIMM; j++)  {  C1[DIMM \* i + j] += A[DIMM \* i + k] \* B[DIMM \* k + j];  }  }  }  }  int main()  {  init2();  //display1(A1);  //display1(B1);  start = chrono::high\_resolution\_clock::now();  matr1(A1, B1);  ennd = chrono::high\_resolution\_clock::now();  chrono::duration<double> diff = ennd - start;  cout << "time: " << diff.count() << endl;  //cout << " res = " << endl;  //display1(C1);  delete[]A1;  delete[]B1;  delete[]C1;  } |

Таблица 4 – Время выполнения алгоритма умножения матриц с одинарными указателями

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Ключ оптимизации | |
| -O0 | -O2 |
| 512 | 0,711614 | 0,203333333 |
| 1024 | 4,142327 | 1,27 |
| 2048 | 30,67953 | 6,516666667 |

Вывод: Приложение с ключом оптимизации -О2 быстрее примерно в 3,83 раза.