**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ**

**«МЭИ»**

**Институт Информационных и Вычислительных технологий**

Кафедра Прикладной Математики и искусственного интеллекта

Лабораторная работа №2

**По курсу** «Параллельное программирование и параллельные системы»

**Выполнил:** Филиппов Е. И.

**Группа:** А-05м-23

**Преподаватель:** Задорин С.А.

Москва, 2024

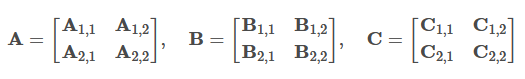
**Постановка задачи**

Реализовать алгоритм умножения матриц методом Штрассена с использованием языка FPTL.

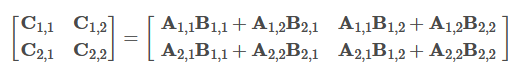
# Алгоритм

Алгоритм умножения матриц Страссена - это первый алгоритм умножения матриц, который асимптотически работает лучше . Данный алгоритм имеет асимптотическую сложность равную .

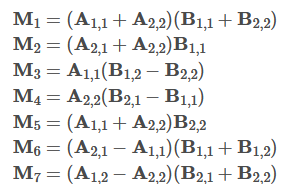
Пусть A и B — две (nxn)-матрицы, причём n — степень числа 2. Тогда можно разбить каждую матрицу A и B на четыре ((n/2)\*(n/2))-матрицы и через них выразить произведение матриц A и B:



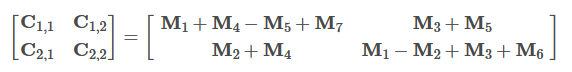
При стандартном подходе матрица C будет иметь такой вид:



Алгоритм Штрассена вводит новые элементы:



Элементы матрицы C будет вычисляться следующим образом:



Таким образом, количество умножений подматриц для приведенной выше формулы становится 7, а количество сложений подматриц для приведенной выше формулы становится 22. Сделано это с расчетом на то, что выполнение операции сложения обходится компьютеру намного дешевле, чем умножение.

Рекурсивный процесс будет продолжатся n раз до тех пор, пока размер матриц не станет достаточно малым, далее используется обычный метод умножения матриц. Это делают из-за того, что алгоритм Штрассена теряет эффективность по сравнению с обычным на малых матрицах в силу большего числа сложений.

# Параллельный алгоритм

При параллельном подходе предлагается вычислять каждый элемент в отдельном потоке. Алгоритм будет состоять из следующих этапов:

1. Задачи m1, m2, m3, m4, m5, m6, m7 выполняются параллельно и независимо друг от друга. Каждая из этих задач выполняет вычисление элементов . Параллелизм достигается с помощью операции «\*»
2. Затем происходит объединение результатов вычислений, которые также выполняется параллельно. Каждая из задач c11, c12, c21, c22 отвечает за соответствующую часть результирующей матрицы.

Таким образом, параллельное разбиение позволяет ускорить выполнение алгоритма за счет распараллеливания вычислений.

# Описание функций

**Scheme Strassen**

определяет схему и представляет собой обертку для реализации умножения матриц методом Штрассена.

Параметры:

[1] – размерность матриц

**StrassenImp**содержит основную реализацию алгоритма метода Штрассена.

Параметры:

[1] – размерность матриц

[2] ­­– матрица 1

[3] – матрица 2

**CombineMatrix**

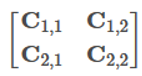
функция объединения 4 подматриц в единую матрицу.  
Параметры:

[1] – матрица С11

[2] ­­– матрица С12

[3] – матрица С21

[4] – матрица С22



**CombineMatrixVertical**

Является оберткой для функции реализации вертикального объединения матриц. Задействуется в функции CombineMatrix.

Параметры:

[1] – матрица 1

[2] ­­– матрица 2

**CombineMatrixVerticalImp**

Функция реализации вертикального объединения матриц.

Параметры:

[1] – текущий номер строки

[2] ­­– матрица 1

[3] ­­– матрица 2

[4] – итоговая матрица

**CombineMatrixHorizontal**  
Является оберткой для функции реализации горизонтального объединения матриц. Задействуется в функции CombineMatrix.

Параметры:

[1] – матрица 1

[2] ­­– матрица 2

**CombineMatrixHorizontalImp**  
Функция реализации горизонтального объединения матриц.  
Параметры:

[1] – текущий номер строки

[2] ­­– матрица 1

[3] ­­– матрица 2

[4] – итоговая матрица

**CombineArrayHorizontal**

Является оберткой для функции реализации горизонтального объединения одномерных массивов. Задействуется при горизонтальном объединении матриц.

Параметры:

[1] – массив 1

[2] ­­– массив 2

**CombineArrayHorizontalImp**

Функция реализации горизонтального объединения одномерных массивов.  
Параметры:

[1] – текущий индекс массива

[2] ­­– массив 1

[3] ­­– массив 2

[4] – итоговый массив

**CreateMatrix**

Функция создания матрицы с числом строк [1], числом столбцов [2] и с заполнением всех элементов числом [3].

**SliceMatrix**

Функция-обертка получения подматрицы из матрицы.

Параметры:

[1] – исходная матрица

[2] ­­– номер строки отсчета

[3] – номер столбца отсчета

[4] ­­– число строк

[5] – число столбцов

**SliceMatrixImp**

Функция реализации получения подматрицы из матрицы.

Параметры:

[1] ­­– текущий номер строки исходной матрицы

[2] – текущий номер строки итоговой матрицы

[3] – номер столбца отсчета

[4] – исходная матрица

[5] ­­– число строк

[6] – число столбцов

[7] – итоговая матрица

**SliceArray**

Функция-обертка получения подмассива из массива.

Параметры:

[1] – исходный массив

[2] ­­– индекс отсчета

[3] – число элементов в подмассиве

**SliceArrayImp**

Функция реализации получения подмассива из массива.

Параметры:

[1] ­­– текущий индекс исходного массива

[2] – текущий индекс итогового подмассива

[3] – исходный массив

[4] – размер подмассива

[5] ­­– итоговый подмассив

**SumMatrix**

Является оберткой для функции реализации сложения матриц.

Параметры:

[1] – матрица 1

[2] ­­– матрица 2

**SumMatrixImp**

Функция реализации сложения матриц.

Параметры:

[1] ­­– текущий номер строки исходной матрицы

[2] – матрица 1

[3] – матрица 2

[4] – итоговая матрица

**SumArrays**

Является оберткой для функции реализации сложения массивов.

Параметры:

[1] – массив 1

[2] ­­– массив 2

**SumArraysImp**

Функция реализации поэлементного сложения двух массивов.

Параметры:

[1] ­­– текущий индекс массива

[2] – массив 1

[3] – массив 2

[4] – итоговый массив

**SubMatrix**

Является оберткой для функции реализации вычитания матриц.

Параметры:

[1] – матрица 1

[2] ­­– матрица 2

**SubMatrixImp**

Функция реализации вычитания матриц.

Параметры:

[1] ­­– текущий номер строки исходной матрицы

[2] – матрица 1

[3] – матрица 2

[4] – итоговая матрица

**SubArrays**

Является оберткой для функции реализации вычитания массивов.

Параметры:

[1] – массив 1

[2] ­­– массив 2

**SubArraysImp**

Функция реализации поэлементного вычитания двух массивов.

Параметры:

[1] ­­– текущий индекс массива

[2] – массив 1

[3] – массив 2

[4] – итоговый массив

**MulMatrix**

Является оберткой для функции реализации умножения матриц.

Параметры:

[1] – матрица 1

[2] ­­– матрица 2

**MulMatrixImp**

Функция реализации умножения матриц.

Параметры:

[1] ­­– текущий номер строки исходной матрицы

[2] – матрица 1

[3] – матрица 2

[4] – итоговая матрица

**MulRowInMatrix**

Является оберткой для функции реализации умножения одной строки матрицы на все столбцы другой матрицы.

Параметры:

[1] – строка матрицы 1

[2] – матрица 2

**MulRowInMatrixImp**

Функция реализации умножения одной строки матрицы на все столбцы другой матрицы.

Параметры:

[1] – текущий номер столбца

[2] – строка матрицы 1

[3] – матрица 2

[4] – итоговая матрица

**MulArrays**

Является оберткой для функции реализации поэлементного умножения одного массива на другой

Параметры:

[1] – массив 1

[2] ­­– массив 2

**MulArraysImp**

Функция реализации поэлементного умножения двух массивов.

Параметры:

[1] ­­– текущий индекс массива

[2] – массив 1

[3] – массив 2

[4] – итоговый массив

**TransposeColumn**

Является оберткой для функции реализации транспонирования столбца в строку.

Параметры:

[1] – исходный столбец

**TransposeColumnImp**

Функция реализации транспонирования столбца в строку.

[1] ­­– текущий индекс массива

[2] – исходный столбец

[3] – итоговая строка

# Конфигурация ПК

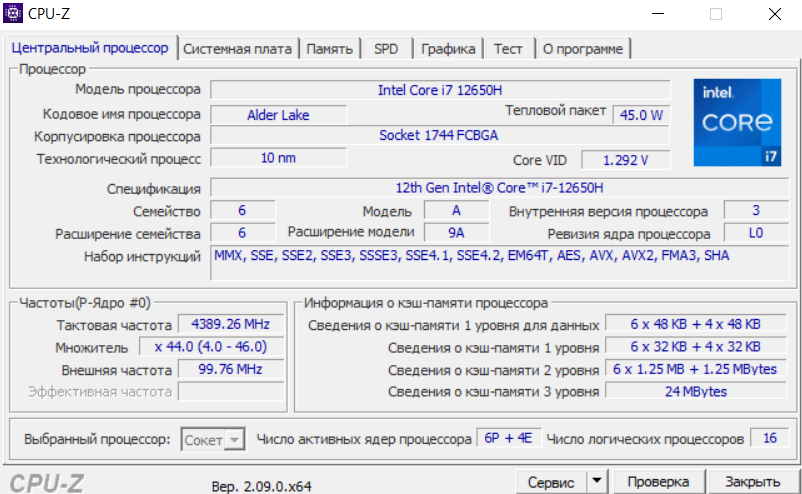


Рис. 1 Информация о процессоре.

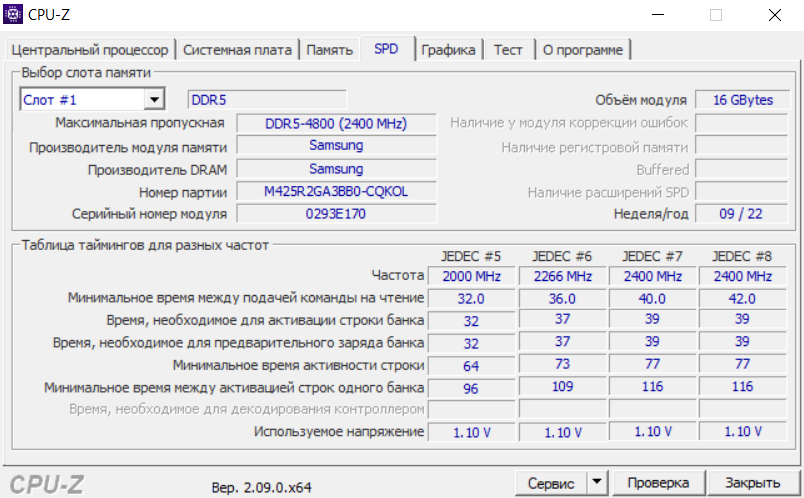
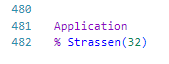
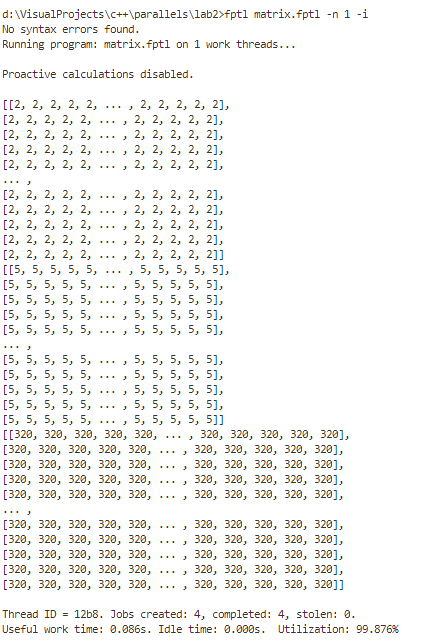


Рис. 2 Информация об оперативной памяти.

# Пример работы программы





# Тесты

При тестировании будем перемножать матрицы размером 64x64, 128x128, 256x256.

Таблица 1. Зависимость времени работы (в секундах) от числа потоков

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность матрицы  Число потоков | 64х64 | 128х128 | 256х256 |
| 1 | 0.592 | 7.181 | 148.776 |
| 2 | 0.351 | 4.470 | 67.866 |
| 4 | 0.320 | 3.403 | 48.409 |
| 8 | 0.182 | 3.188 | 45.030 |
| 16 | 0.178 | 3.158 | 51.564 |
| 32 | 0.172 | 4.088 | 53.424 |
| 64 | 0.184 | 4.645 | 71.343 |

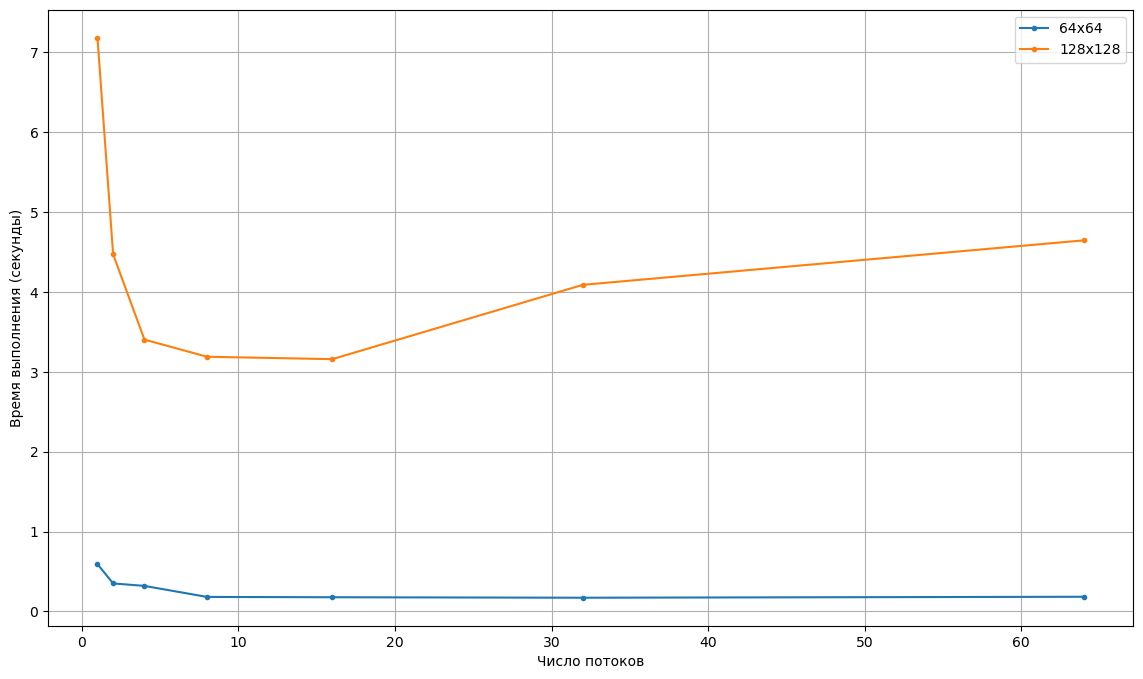


Рис. 3 График зависимости времени выполнения от числа потоков

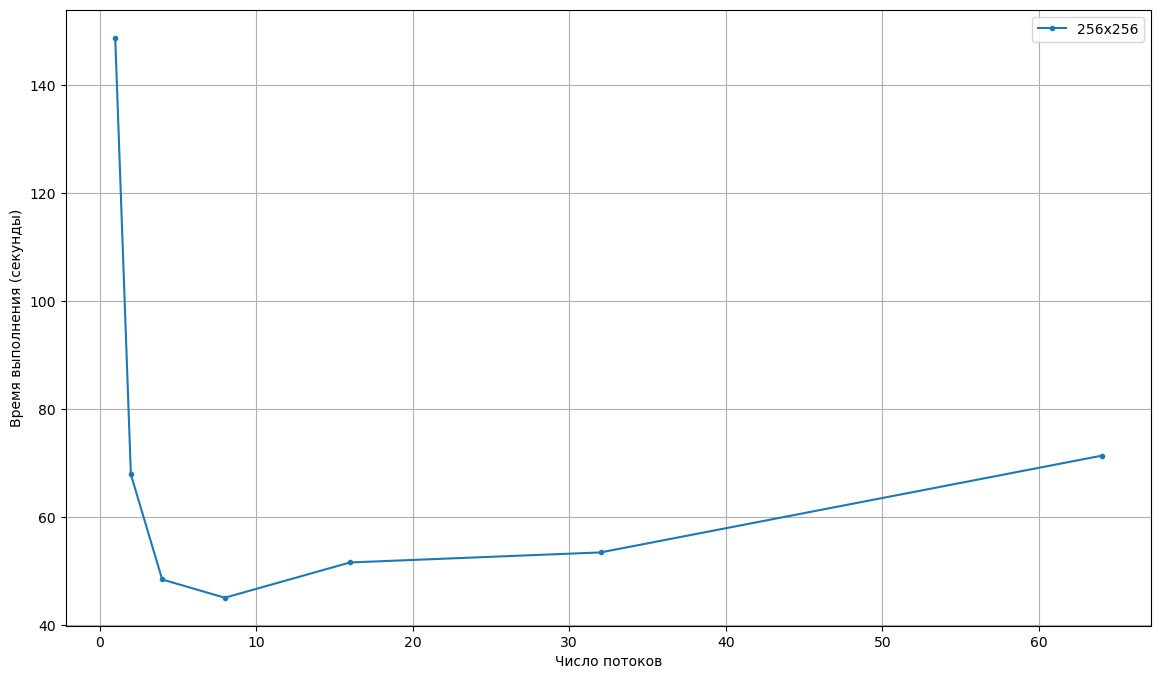


Рис. 4 График зависимости времени выполнения от числа потоков

Таблица 2. Зависимость коэффициента ускорения от числа потоков

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность матрицы  Число потоков | 64x64 | 128x128 | 256x256 |
| 1 | 1.000 | 1.000 | 1.000 |
| 2 | 1.687 | 1.606 | 2.192 |
| 4 | 1.850 | 2.110 | 3.073 |
| 8 | 3.253 | 2.253 | 3.304 |
| 16 | 3.326 | 2.274 | 2.885 |
| 32 | 3.442 | 1.757 | 2.785 |
| 64 | 3.217 | 1.546 | 2.085 |

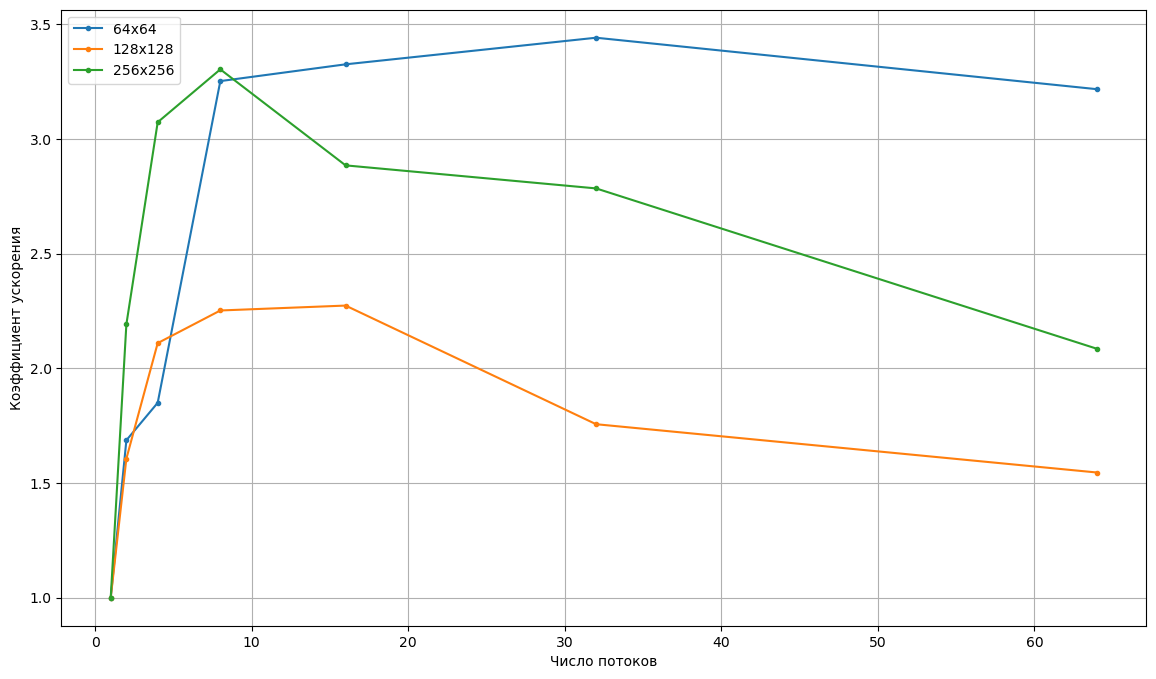


Рис. 5 График зависимости коэффициента ускорения от числа потоков

# Вывод

Грамотный подбор числа потоков позволяет нам достигать наилучшего соотношения выделяемых ресурсов к скорости вычисления. Из графиков видно, что на 8 потоках, как и в реализации на Open MP, заканчивается рост ускорения, после чего начинается планомерное его падение. Таким образом, на личном ПК для решения данной задачи оптимальным числом потоков является 8. При параллельном решении задачи удалось ускорить выполнение алгоритма примерно в 2.5 раза. Показатели ускорения практически совпали с результатами из предыдущей работы. Однако, в отличии от реализации на Open MP, время выполнение алгоритма значительно увеличилось, что побудило отказаться от использования в тестировании матриц более высокой размерности.

# Программный код

FPTL:

|  |
| --- |
| Scheme Strassen  {      n = [1];      matrix\_1 = (n \* n \* 2).CreateMatrix;      matrix\_2 = (n \* n \* 5).CreateMatrix;      @ = (n \* matrix\_1 \* matrix\_2).StrassenImp;      Fun StrassenImp      {          n = [1];          matrix\_1 = [2];          matrix\_2 = [3];            m = (n \* 2).div;          a = (matrix\_1 \* 0 \* 0 \* m \* m).SliceMatrix;          b = (matrix\_1 \* 0 \* m \* m \* m).SliceMatrix;          c = (matrix\_1 \* m \* 0 \* m \* m).SliceMatrix;          d = (matrix\_1 \* m \* m \* m \* m).SliceMatrix;          e = (matrix\_2 \* 0 \* 0 \* m \* m).SliceMatrix;          f = (matrix\_2 \* 0 \* m \* m \* m).SliceMatrix;          g = (matrix\_2 \* m \* 0 \* m \* m).SliceMatrix;          h = (matrix\_2 \* m \* m \* m \* m).SliceMatrix;          bds = (b \* d).SubMatrix;          gha = (g \* h).SumMatrix;          m1 = (m \* bds \* gha).StrassenImp;          ada = (a \* d).SumMatrix;          eha = (e \* h).SumMatrix;          m2 = (m \* ada \* eha).StrassenImp;          acs = (a \* c).SubMatrix;          efa = (e \* f).SumMatrix;          m3 = (m \* acs \* efa).StrassenImp;          aba = (a \* b).SumMatrix;          m4 = (m \* aba \* h).StrassenImp;          fhs = (f \* h).SubMatrix;          m5 = (m \* a \* fhs).StrassenImp;          ges = (g \* e).SubMatrix;          m6 = (m \* d \* ges).StrassenImp;          cda = (c \* d).SumMatrix;          m7 = (m \* cda \* e).StrassenImp;          s1s2a = (m1 \* m2).SumMatrix;          s6s4s = (m6 \* m4).SubMatrix;          c11 = (s1s2a \* s6s4s).SumMatrix;          c12 = (m4 \* m5).SumMatrix;          c21 = (m6 \* m7).SumMatrix;          s2s3s = (m2 \* m3).SubMatrix;          s5s7s = (m5 \* m7).SubMatrix;          c22 = (s2s3s \* s5s7s).SumMatrix;          @ = (n \* 32).lequal->          (matrix\_1 \* matrix\_2).MulMatrix,          (c11 \* c12 \* c21 \* c22).CombineMatrix;      }      Fun CombineMatrix      {          matrix\_11 = [1];          matrix\_12 = [2];          matrix\_21 = [3];          matrix\_22 = [4];          @ = ((matrix\_11 \* matrix\_12).CombineMatrixHorizontal \* (matrix\_21 \* matrix\_22).CombineMatrixHorizontal).CombineMatrixVertical;      }      Fun CombineMatrixVertical      {          matrix\_1 = [1];          matrix\_2 = [2];          rows = (matrix\_1.arrayLen \* matrix\_2.arrayLen).add;          cols = matrix\_1.arrayLen;          result\_matrix = (rows \* cols \* 0).CreateMatrix;          @ = (0 \* matrix\_1 \* matrix\_2 \* result\_matrix).CombineMatrixVerticalImp;          Fun CombineMatrixVerticalImp          {              index = [1];              matrix\_1 = [2];              matrix\_2 = [3];              result\_matrix = [4];              size\_row\_1 = matrix\_1.arrayLen;              size\_row\_2 = matrix\_2.arrayLen;              size\_row\_result = (size\_row\_1 \* size\_row\_2).add;              index\_2 = (index \* size\_row\_1).mod;              cur\_array\_1 = (matrix\_1 \* index).arrayGet;              cur\_array\_2 = (matrix\_2 \* index\_2).arrayGet;              fillMatrix = (index \* size\_row\_1).less->                  (                      (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* cur\_array\_1).arraySet                  ).CombineMatrixVerticalImp,                  (                      (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* cur\_array\_2).arraySet                  ).CombineMatrixVerticalImp;                @ = (index \* size\_row\_result).less->                  fillMatrix,                  result\_matrix;          }      }      Fun CombineMatrixHorizontal      {          matrix\_1 = [1];          matrix\_2 = [2];          rows = matrix\_1.arrayLen;          matrix\_1\_cols = (matrix\_1 \* 0).arrayGet.arrayLen;          matrix\_2\_cols = (matrix\_2 \* 0).arrayGet.arrayLen;          cols = (matrix\_1\_cols \* matrix\_2\_cols).add;          result\_matrix = (rows \* cols \* 0).CreateMatrix;          @ = (0 \* matrix\_1 \* matrix\_2 \* result\_matrix).CombineMatrixHorizontalImp;          Fun CombineMatrixHorizontalImp          {              index = [1];              matrix\_1 = [2];              matrix\_2 = [3];              result\_matrix = [4];              row\_size = matrix\_1.arrayLen;              cur\_array\_1 = (matrix\_1 \* index).arrayGet;              cur\_array\_2 = (matrix\_2 \* index).arrayGet;              result\_array = (cur\_array\_1 \* cur\_array\_2).CombineArrayHorizontal;              @ = (index \* row\_size).less->                  (                      (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* result\_array).arraySet                  ).CombineMatrixHorizontalImp,                  result\_matrix;          }          Fun CombineArrayHorizontal          {              array\_1 = [1];              array\_2 = [2];              size\_1 = array\_1.arrayLen;              size\_2 = array\_2.arrayLen;              size\_result = (size\_1 \* size\_2).add;              result\_array = (size\_result \* 0).arrayCreate;              @ = (0 \* array\_1 \* array\_2 \* result\_array).CombineArrayHorizontalImp;              Fun CombineArrayHorizontalImp              {                  index = [1];                  array\_1 = [2];                  array\_2 = [3];                  result\_array = [4];                  size\_1 = array\_1.arrayLen;                  size\_2 = array\_2.arrayLen;                  size\_result = result\_array.arrayLen;                  index\_2 = (index \* size\_1).mod;                  cur\_1 = (array\_1 \* index).arrayGet;                  cur\_2 = (array\_2 \* index\_2).arrayGet;                  fillArray = (index \* size\_1).less->                      (                          (index \* 1).add \* array\_1 \* array\_2 \* (result\_array \* index \* cur\_1).arraySet                      ).CombineArrayHorizontalImp,                      (                          (index \* 1).add \* array\_1 \* array\_2 \* (result\_array \* index \* cur\_2).arraySet                      ).CombineArrayHorizontalImp;                  @ = (index \* size\_result).less->                      fillArray,                      result\_array;              }          }      }      Fun CreateMatrix      {          size\_row = [1];          size\_col = [2];          value = [3];          @ = (size\_row \* (size\_col \* value).arrayCreate).arrayCreate;      }      Fun SliceMatrix      {          source\_matrix = [1];          offset\_i = [2];          offset\_j = [3];          size\_row = [4];          size\_col = [5];          result\_matrix = (size\_row \* size\_col \* 0).CreateMatrix;          @ = (offset\_i \* 0 \* offset\_j \* source\_matrix \* size\_row \* size\_col \* result\_matrix).SliceMatrixImp;          Fun SliceMatrixImp          {              offset\_i = [1];              copy\_offset\_i = [2];              offset\_j = [3];              source\_matrix = [4];              size\_row = [5];              size\_col = [6];              copy\_matrix = [7];              f\_array\_slice = ((source\_matrix \* offset\_i).arrayGet \* offset\_j \* size\_col).SliceArray;              @ = (copy\_offset\_i \* size\_row).less->                  (                      (offset\_i \* 1).add \* (copy\_offset\_i \* 1).add \* offset\_j \* source\_matrix \* size\_row \* size\_col \* (copy\_matrix \* copy\_offset\_i \* f\_array\_slice).arraySet                  ).SliceMatrixImp,              copy\_matrix;          }          Fun SliceArray          {              source\_array = [1];              offset = [2];              size = [3];              copy\_array = (size \* 0).arrayCreate;              @ = (offset \* 0 \* source\_array \* size \* copy\_array).SliceArrayImp;              Fun SliceArrayImp              {                  source\_index = [1];                  copy\_index = [2];                  source\_array = [3];                  size = [4];                  copy\_array = [5];                  @ = (copy\_index \* size).less->                      (                          (source\_index \* 1).add \* (copy\_index \* 1).add \* source\_array \* size \* (copy\_array \* copy\_index \* (source\_array \* source\_index).arrayGet).arraySet                      ).SliceArrayImp,                  copy\_array;              }          }      }      Fun SumMatrix      {          matrix\_1 = [1];          matrix\_2 = [2];          size\_row = matrix\_1.arrayLen;          size\_col = (matrix\_1 \* 0).arrayGet.arrayLen;          result\_matrix = (size\_row \* size\_col \* 0).CreateMatrix;          @ = (0 \* matrix\_1 \* matrix\_2 \* result\_matrix).SumMatrixImp;            Fun SumMatrixImp          {              index = [1];              matrix\_1 = [2];              matrix\_2 = [3];              result\_matrix = [4];              size\_row = matrix\_1.arrayLen;              array\_1 = (matrix\_1 \* index).arrayGet;              array\_2 = (matrix\_2 \* index).arrayGet;              result\_array = (array\_1 \* array\_2).SumArrays;              @ = (index \* size\_row).less->                  (                      (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* result\_array).arraySet                  ).SumMatrixImp,              result\_matrix;          }          Fun SumArrays          {              array\_1 = [1];              array\_2 = [2];              size = array\_1.arrayLen;              result\_array = (size \* 0).arrayCreate;              @ = (0 \* array\_1 \* array\_2 \* result\_array).SumArraysImp;              Fun SumArraysImp              {                  index = [1];                  array\_1 = [2];                  array\_2 = [3];                  result\_array = [4];                  size = array\_1.arrayLen;                  cur\_1 = (array\_1 \* index).arrayGet;                  cur\_2 = (array\_2 \* index).arrayGet;                  sum\_curs = (cur\_1 \* cur\_2).add;                  @ = (index \* size).less->                      (                          (index \* 1).add \* array\_1 \* array\_2 \* (result\_array \* index \* sum\_curs).arraySet                      ).SumArraysImp,                  result\_array;              }          }      }      Fun SubMatrix      {          matrix\_1 = [1];          matrix\_2 = [2];          size\_row = matrix\_1.arrayLen;          size\_col = (matrix\_1 \* 0).arrayGet.arrayLen;          result\_matrix = (size\_row \* size\_col \* 0).CreateMatrix;          @ = (0 \* matrix\_1 \* matrix\_2 \* result\_matrix).SubMatrixImp;            Fun SubMatrixImp          {              index = [1];              matrix\_1 = [2];              matrix\_2 = [3];              result\_matrix = [4];              size\_row = matrix\_1.arrayLen;              array\_1 = (matrix\_1 \* index).arrayGet;              array\_2 = (matrix\_2 \* index).arrayGet;              result\_array = (array\_1 \* array\_2).SubArrays;              @ = (index \* size\_row).less->                  (                      (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* result\_array).arraySet                  ).SubMatrixImp,              result\_matrix;          }          Fun SubArrays          {              array\_1 = [1];              array\_2 = [2];              size = array\_1.arrayLen;              result\_array = (size \* 0).arrayCreate;              @ = (0 \* array\_1 \* array\_2 \* result\_array).SubArraysImp;              Fun SubArraysImp              {                  index = [1];                  array\_1 = [2];                  array\_2 = [3];                  result\_array = [4];                  size = array\_1.arrayLen;                  cur\_1 = (array\_1 \* index).arrayGet;                  cur\_2 = (array\_2 \* index).arrayGet;                  sub\_curs = (cur\_1 \* cur\_2).sub;                  @ = (index \* size).less->                      (                          (index \* 1).add \* array\_1 \* array\_2 \* (result\_array \* index \* sub\_curs).arraySet                      ).SubArraysImp,                  result\_array;              }          }      }      Fun MulMatrix      {          matrix\_1 = [1];          matrix\_2 = [2];          size\_row = matrix\_1.arrayLen;          size\_col = (matrix\_1 \* 0).arrayGet.arrayLen;          result\_matrix = (size\_row \* size\_col \* 0).CreateMatrix;          @ = (0 \* matrix\_1 \* matrix\_2 \* result\_matrix).MulMatrixImp;      }      Fun MulMatrixImp      {          index = [1];          matrix\_1 = [2];          matrix\_2 = [3];          result\_matrix = [4];          cur\_row\_array = (matrix\_1 \* index).arrayGet;          size\_row = matrix\_1.arrayLen;          result\_mul\_row = (cur\_row\_array \* matrix\_2).MulRowInMatrix;          @ = (index \* size\_row).less->              (                  (index \* 1).add \* matrix\_1 \* matrix\_2 \* (result\_matrix \* index \* result\_mul\_row).arraySet              ).MulMatrixImp,              result\_matrix;      }      Fun MulRowInMatrix      {          row\_array = [1];          matrix = [2];          row\_size = (row\_array).arrayLen;          result\_row = (row\_size \* 0).arrayCreate;          @ = (0 \* row\_array \* matrix \* result\_row).MulRowInMatrixImp;      }      Fun MulRowInMatrixImp      {          index = [1];          row\_array = [2];          matrix = [3];          result\_row = [4];          size\_row = row\_array.arrayLen;          size\_col = (matrix \* 0).arrayGet.arrayLen;          column\_array = (matrix \* 0 \* index \* size\_row \* 1).SliceMatrix.TransposeColumn;          mul\_result\_array = (row\_array \* column\_array).MulArrays;          mul\_result = mul\_result\_array.arraySum;            @ = (index \* size\_col).less->              (                  (index \* 1).add \* row\_array \* matrix \* (result\_row \* index \* mul\_result).arraySet              ).MulRowInMatrixImp,              result\_row;      }      Fun MulArrays      {          array\_1 = [1];          array\_2 = [2];          size = array\_1.arrayLen;          result\_array = (size \* 0).arrayCreate;            @ = (0 \* array\_1 \* array\_2 \* result\_array).MulArraysImp;          Fun MulArraysImp          {              index = [1];              array\_1 = [2];              array\_2 = [3];              result\_array = [4];              size = array\_1.arrayLen;              cur\_1 = (array\_1 \* index).arrayGet;              cur\_2 = (array\_2 \* index).arrayGet;              mul\_curs = (cur\_1 \* cur\_2).mul;              @ = (index \* size).less->                  (                      (index \* 1).add \* array\_1 \* array\_2 \* (result\_array \* index \* mul\_curs).arraySet                  ).MulArraysImp,                  result\_array;          }      }      Fun TransposeColumn      {          array\_col = [1];          array\_size = array\_col.arrayLen;          result\_array = (array\_size \* 0).arrayCreate;            @ = (0 \* array\_col \* result\_array).TransposeColumnImp;            Fun TransposeColumnImp          {              index = [1];              array\_col = [2];              result\_array = [3];              cur = ((array\_col \* index).arrayGet \* 0).arrayGet;              size = result\_array.arrayLen;                @ = (index \* size).less->                  (                      (index \* 1).add \* array\_col \* (result\_array \* index \* cur).arraySet                  ).TransposeColumnImp,                  result\_array;          }      }  }  Application  % Strassen(256) |