Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологии

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ**

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Вариант - №4

Выполнил студент гр. 02001 Д.Р. Бараев

Руководитель, доцент А.П. Антонов

«23» декабря 2021

Санкт-Петербург

2021

**Содержание**

[1. Задание 4](#_Toc91258388)

[2. Введение 6](#_Toc91258389)

[3. Исходный код функции 8](#_Toc91258390)

[4. Исходный код теста 9](#_Toc91258391)

[5. Исходный код командного файла 10](#_Toc91258392)

[6. Результаты исследования и сравнение решений 12](#_Toc91258393)

[7. Анализ результатов 12](#_Toc91258394)

[8. Оптимизация решения. Добавление директив. Сравнение решений 13](#_Toc91258395)

[9. Анализ результатов 16](#_Toc91258396)

[10. Исходный код модернизированного теста 18](#_Toc91258397)

[11. Выводы 23](#_Toc91258398)

[Приложение 24](#_Toc91258399)

**Список иллюстраций**

[Рисунок 1 - Сравнение полученных решений при N = 8192 12](#_Toc91258423)

[Рисунок 2 - Таблица данных для всех решений 13](#_Toc91258424)

[Рисунок 3 - График зависимости данных для всех решений 13](#_Toc91258425)

[Рисунок 4 - Директивы для sol2 14](#_Toc91258426)

[Рисунок 5 - Директивы для sol3 14](#_Toc91258427)

[Рисунок 6 - Директивы для sol4 14](#_Toc91258428)

[Рисунок 7 - Директивы для sol5 15](#_Toc91258429)

[Рисунок 8 - Директивы для sol6 15](#_Toc91258430)

[Рисунок 9 - Сравнение полученных решений (с директивами) 16](#_Toc91258431)

[Рисунок 10 - Таблица данных для всех решений 16](#_Toc91258432)

[Рисунок 11 - График зависимости Latency (нс) 17](#_Toc91258433)

[Рисунок 12 - График зависимости FF и LUT 17](#_Toc91258434)

[Рисунок 13 - Таблица данных для всех решений при N = 512 18](#_Toc91258435)

[Рисунок 14 - График зависимости Latency (нс) при N = 512 18](#_Toc91258436)

[Рисунок 15 - График зависимости FF и LUT при N = 512 18](#_Toc91258437)

[Рисунок 16 - Параметры ПК (Частота = 2.7 Гц) 19](#_Toc91258438)

[Рисунок 17 - Временные показатели для модифицированного теста 22](#_Toc91258439)

[Рисунок 18 - Таблица данных для sol6 и программной реализации при N = 512 23](#_Toc91258440)

# Задание

Даны матрицы А, В, С квадратные, размер NxN. Тип элементов int a.

Найти (А+В) С, в тесте для проверки использовать АС+ВС.

**Исследование включает:**

* оптимизацию аппаратной реализации и оценку производительности,
* анализ производительности программной реализации на ПК,
* сравнительный анализ программной и аппаратной реализаций.

**Оптимизация аппаратной реализации и оценка ее производительности:**

* Параметр N и тип данных должны быть определены в include файле
* Оптимизация аппаратной реализации и оценка ее производительности осуществляется в пакете Vivado HLS.
* Способ оценки производительности: II (число тактов) \* Estimated time period
* В тесте, в рамках пакета Vivado HLS, исследуемую функцию следует запускать не менее двух раз.
* Заполнение матриц в тестах осуществлять случайными числами.
* Тест должен быть с самопроверкой результата (обязательно предусмотреть и отразить в записке проверку теста – при неправильных данных должен формировать ошибку).
* Оптимизация аппаратной реализации осуществлять для N=8192.
* Цель оптимизации – максимальная производительность. Ограничения – доступные аппаратные ресурсы микросхемы xa7a100tfgg484-2i
* Все варианты, созданные при исследовании производительности должны быть представлены в проекте и описаны в пояснительной записке. Должно быть приведено сравнение результатов (+заполнена xls таблица и приведены графики зависимости производительности и аппаратных затрат для созданных вариантов реализаций)
* Следует обосновать выбор директив для оптимизации.
* Следует обосновать достижения максимума производительности (невозможность дальнейшего увеличения производительности без превышения ограничений по аппаратным затратам; или обосновать особенностью алгоритма)
* Оценку производительности аппаратной реализации осуществить для выбранного, оптимального решения, при N=8192, 32768, 131072

**Анализ производительности программной реализации на ПК:**

* Для тестирования производительности программной реализации синтезируемой функции необходимо создать:
* отдельный, модернизированный, тест для проверки времени выполнения синтезируемой функции на ПК:
* добавить в тест операторы измерения времени выполнения синтезируемой функции
* Увеличить количество запусков синтезируемой функции до 32. Для каждого запуска измерить время, найти среднее значение и вывести как результат.
* Точность измерения времени (наносекунды).
* Тестирование проводить при N=8192, 32768, 131072
* Провести исследование времени выполнения синтезируемой функции на Вашем ПК
* Осуществить компиляцию модернизированного теста и запустить его как отдельное приложение при N=8192, 32768, 131072
* В отчете привести:
* Параметры Вашего ПК: тип процессора, частота работы процессора, объем ОЗУ
* результаты измерения времени выполнения при N=8192, 32768, 131072

**Сравнительный анализ программной и аппаратной реализаций**

* Привести табличные данные по производительности аппаратной и программной реализации (при N=8192, 32768, 131072), табличные данные для аппаратных затрат.
* Привести графики зависимости производительности (времени выполнения) аппаратной и программной реализации при N=8192, 32768, 131072. (По возможности совмещенные данные, можно использовать логарифмический масштаб).
* Сделать вывод

# Введение

**Сложение матриц**

Операция сложения определена для матриц одного размера.

Пусть *A, B ∈ Mm×n*:

Изображение выглядит как текст

Автоматически созданное описание

Суммой матриц *A* и *B* называется матрица:

Изображение выглядит как текст

Автоматически созданное описание

**Свойства операции сложения матриц**

1. Операция сложения матриц коммутативна и ассоциативна, т. е.

*A + B = B + A* — коммутативность.

*(A + B) + C = A + (B + C)* — ассоциативность.

Доказательство следует из соответствующих свойств действительных чисел.

1. *A + O = O + A* . Здесь *O* — матрица того же размера, что и матрица A .
2. Для любой матрицы *A* существует единственная матрица *B* такая, что *A*+*B* = *O*.

**Определение.** Матрица *B* называется противоположной матрице *A* и обозначается — *A*, если *A + B = О* . С помощью противоположной матрицы вводится понятие вычитания матриц, а именно *A − B = A+ (−B).*

1. Операция сложения и транспонирования связаны соотношением

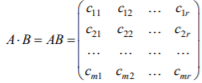
**Умножение матриц**

Даны две матрицы A и B:

Изображение выглядит как текст

Автоматически созданное описание

Если число столбцов матрицы A совпадает с числом строк матрицы B, тогда их произведение определяется так:



**Свойства операции умножения матриц**

1. Если *A ∈ Mm×n, B ∈ Mn×r, то AB ∈ Mm×r*
2. Умножение матриц, вообще говоря, некоммутативно, т.е., если *AB* и *BA* существует, то не обязательно *AB* = *BA*. В этом заключается одно из отличий операции умножения матриц от операции умножения чисел (последнее всегда коммутативно).
3. Умножение матриц ассоциативно: *(AB)C* = *A(BC)* (при условии существования указанных произведений).
4. Умножение матриц дистрибутивно относительно сложения: *(A + B)C =* *AC + BC, F(A + B) = FA+ FB*.
5. *(λA)B = λ(AB) = A(λB).*
6. Пусть E — единичная матрица. Тогда для любой квадратной матрицы A того же порядка что и E

*EA = AE = A.*

Отметим ещё одно отличительное свойство умножения матриц: произведение двух ненулевых матриц может оказаться равным нулю. Ненулевые матрицы *A* и *B*, удовлетворяющие условию *AB = O*, называются **истинными делителями нуля.**

# Исходный код функции

Исходный код синтезируемой функции *kurs* приведен в *Листинг 1*.

Функция принимает 4 аргумента массива типа int — вычисляет сумму двух первых матриц, после чего вычисляет произведение результата сложения с третьей матрицей и записывает результат в выходной массив.

Листинг 1 - Исходный код функции (файл kurs.c)

#include "kurs.h"

void kursFunc(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

L3: for(int i = 0; i < N; i++) { //проход по строкам

L2: for(int j = 0; j < N; j++) { //проход по столбцам

outD[i][j] = 0;

L1: for(int k = 0; k < N; k++) {

outD[i][j] += (inA[i][k] + inB[i][k]) \* inC[k][j];

}

}

}

}

inA[i][k] + inB[i][k] *- в строке суммируем соответствующую строку с соответствующим столбцом в массивах A и B*

\* inC[k][j] *- после, результат сложения умножаем на соответствующий элемент в массиве С*

outD[i][j] *- аккумулируем результат в выходной массив*

# Исходный код теста

Исходный код теста для проверки функции *kurs* приведен в *Листинг 2*. Для проверки результата была написана проверочная функция, в которой вычисляется произведения первых двух матриц с третьей матрицей, после чего полученные матрицы складываются, и результат записывается во временную переменную, которая сравнивается с соответствующим элементом в выходном массиве.

AC[i][j] += inA[i][k] \* inC[k][j] *- аккумулируем результат произведения соответствующей строки с соответствующим столбцом в матрицах A и С*

BC[i][j] += inB[i][k] \* inC[k][j] *- аккумулируем результат произведения соответствующей строки с соответствующим столбцом в матрицах B и С*

temp\_res = AC[i][j] + BC[i][j] *- результаты двух вычислений складываем и записываем во временную переменную для сравнения с выходным массивом*

Листинг 2 - Исходный код теста (файл kurs\_test.c)

#include "kurs.h"

#include <stdio.h>

int cmp\_arr(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

data\_sc AC[N][N];

data\_sc BC[N][N];

for(int i = 0; i < N; i++) { // проход по строкам

data\_sc temp\_res = 0;

for(int j = 0; j < N; j++) { // проход по столбцам

AC[i][j] = 0;

BC[i][j] = 0;

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

AC[i][j] += inA[i][k] \* inC[k][j];

BC[i][j] += inB[i][k] \* inC[k][j];

}

temp\_res = AC[i][j] + BC[i][j];

if (outD[i][j] != temp\_res) {

return 0;

}

}

}

return 1;

}

int main() {

int pass = 0;

data\_sc inA[N][N];

data\_sc inB[N][N];

data\_sc inC[N][N];

data\_sc outD[N][N];

for (int i = 0; i < 3; i++) {

//set inA inB inC

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

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

inA[j][k] = rand();

inB[j][k] = rand();

inC[j][k] = rand();

}

}

kursFunc(inA, inB, inC, outD);

pass = cmp\_arr(inA, inB, inC, outD);

}

if (pass == 1) {

fprintf(stdout, "----------Pass!------------\n");

return 0;

} else {

fprintf(stderr, "----------Fail!------------\n");

return 1;

}

}

# Исходный код командного файла

В Листинг 3 представлен текст команд для автоматизированного создания следующих вариантов аппаратной реализации:

1. Для sol1 задается clock period 6: clock uncertainty 0.1
2. Для sol2 задается clock period 8. clock uncertainty 0.1
3. Для sol3 задается clock period 10. clock uncertainty 0.1
4. Для sol4 задается clock period 12. clock uncertainty 0.1
5. Для sol5 задается clock period 16. clock uncertainty 0.1

Листинг 3 - Исходный код командного файла для создания проекта (файл kurs.tcl)

#############################################

# Lab #

#############################################

open\_project -reset kr\_02001\_4

set\_top kursFunc

add\_files ./source/kurs.c

add\_files -tb ./source/kurs\_test.c

open\_solution -reset sol1

create\_clock -period 6 -name clk

set\_clock\_uncertainty 0.1

set\_part {xa7a12tcsg325-1Q}

csim\_design

csynth\_design

#cosim\_design -trace\_level all

#############################################

# Solutions #

#############################################

set all\_solutions {sol2 sol3 sol4 sol5}

set all\_periods {{8} {10} {12} {16}}

foreach the\_solution $all\_solutions the\_period $all\_periods {

open\_solution -reset $the\_solution

create\_clock -period $the\_period -name clk

set\_clock\_uncertainty 0.1

set\_part {xa7a12tcsg325-1Q}

csim\_design

csynth\_design

#cosim\_design -trace\_level all

}

exit

# Результаты исследования и сравнение решений

На Рисунке 1 представлено сравнение из Vivado HLS GUI по аппаратным ресурсам, требуемых для реализации синтезируемой функции, и временным параметрам.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1 - Сравнение полученных решений при N = 8192

Target – планируемое время на один такт.

Estimated – оценочное время.

Latency (cycle) – количество тактов latency за один цикл.

Latency (absolute) – время затраченное на latency.

# Анализ результатов

На рисунке 2 представлена таблица с параметрами для всех решений, где рассчитывается Latency в нс.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2 - Таблица данных для всех решений

На рисунке 3 представлен график данных для сравнения всех решений.

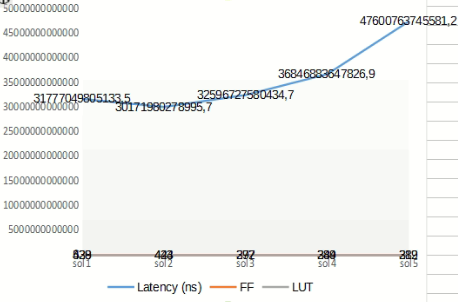


Рисунок 3 - График зависимости данных для всех решений

Исходя из результатов, видно, что наилучшее решение – sol4 с clock period 12 нс. У этого решения хороший результат по времени и почти самое низкое потребление аппаратных ресурсов.

# Оптимизация решения. Добавление директив. Сравнение решений

На основе лучшего решения создали еще 5 решений. В каждом решении для наглядности добавили разное сочетание директив:

* **sol1** – исходное решение (без добавления директив)
* **для решения sol2**

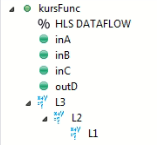


Рисунок 4 - Директивы для sol2

* **для решения sol3**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 - Директивы для sol3

* **для решения sol4**

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 6 - Директивы для sol4

* **для решения sol5**

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 7 - Директивы для sol5

* **для решения sol6**

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 8 - Директивы для sol6

На рисунке 9 представлено сравнение решений с добавлением директив из Vivado HLS GUI по аппаратным ресурсам, требуемых для реализации синтезируемой функции, и временным параметрам.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 9 - Сравнение полученных решений (с директивами)

# Анализ результатов

На рисунке 10 представлена таблица с параметрами для всех решений с добавлением директив, где рассчитывается Latency в нс.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 10 - Таблица данных для всех решений

На рисунках 11 и 12 представлен график данных для сравнения всех решений.

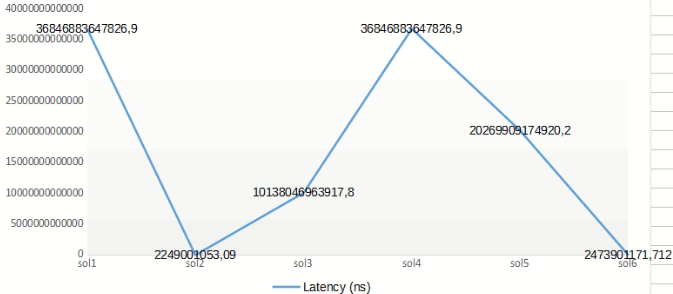


Рисунок 11 - График зависимости Latency (нс)

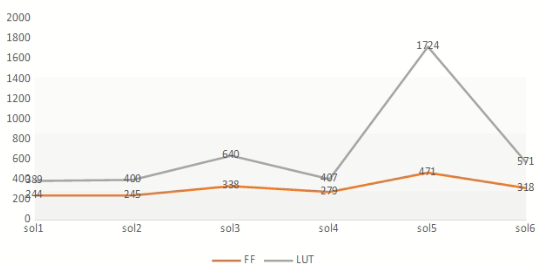


Рисунок 12 - График зависимости FF и LUT

Исходя из результатов, видно, что наилучшее решение – sol6 с добавлением директив dataflow, array partition, resource ram\_2p, pipeline. У этого решения хороший результат по времени и относительно небольшое потребление аппаратных ресурсов.

Так как запуск теста на ПК проводился с N = 512 для того, чтобы сравнить результаты на лабораторном компьютере и на личном ПК, мы также провели тестирование с N = 512. При тестировании с N = 1024 на личном ПК выводилась ошибка сегментирования (ошибку можно увидеть на рисунке 17).

Ниже приведены результаты синтезирования функции при N = 512 на лабораторном компьютере.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 13 - Таблица данных для всех решений при N = 512

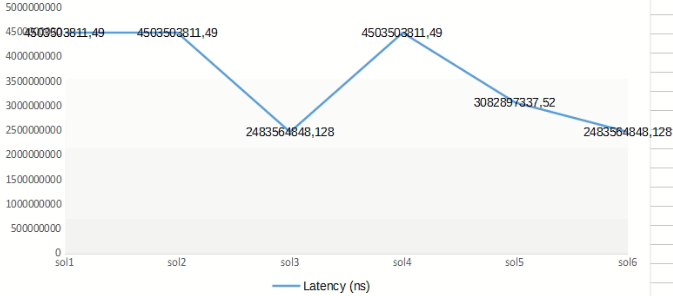


Рисунок 14 - График зависимости Latency (нс) при N = 512

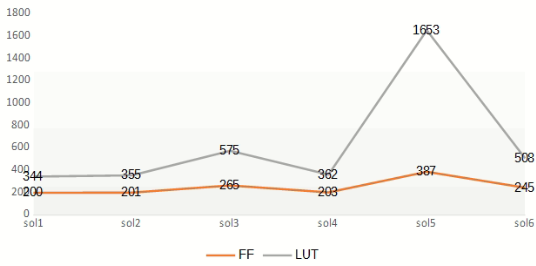


Рисунок 15 - График зависимости FF и LUT при N = 512

# Исходный код модернизированного теста

На листинге 4 представлен исходный код модифицированного теста для проверки функции *kurs*. Тест обеспечивает проверку производительности функции (Компилятор gcc-9.3.0).

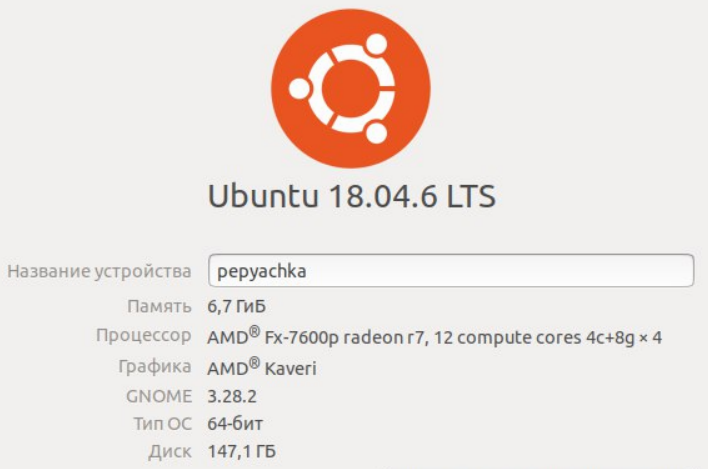


Рисунок 16 - Параметры ПК (Частота = 2.7 Гц)

Было выбрано N = 512, так как при увеличении N результата приходится ждать слишком долго.

Листинг 4 - Исходный код модернизированного теста (файл kurs\_mod\_test.c)

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include "kurs.h"

int cmp\_arr(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

data\_sc AC[N][N];

data\_sc BC[N][N];

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

data\_sc temp\_res = 0;

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

AC[i][j] = 0;

BC[i][j] = 0;

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

AC[i][j] += inA[i][k] \* inC[k][j];

BC[i][j] += inB[i][k] \* inC[k][j];

}

temp\_res = AC[i][j] + BC[i][j];

if (outD[i][j] != temp\_res) {

return 0;

}

}

}

return 1;

}

int main() {

int pass = 0;

data\_sc inA[N][N];

data\_sc inB[N][N];

data\_sc inC[N][N];

data\_sc outD[N][N];

struct timespec t0, t1;

double acc\_time = 0.0;

for (int i = 0; i < 32; i++) {

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

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

inA[j][k] = rand() % (N - 1);

inB[j][k] = rand() % (N - 1);

inC[j][k] = rand() % (N - 1);

}

}

if(clock\_gettime(CLOCK\_REALTIME, &t0) != 0) {

perror("Error in calling clock\_gettime\n");

exit(EXIT\_FAILURE);

}

kursFunc(inA, inB, inC, outD);

if(clock\_gettime(CLOCK\_REALTIME, &t1) != 0) {

perror("Error in calling clock\_gettime\n");

exit(EXIT\_FAILURE);

}

double diff\_time = (((double)(t1.tv\_sec - t0.tv\_sec)) \* 1000000000.0) + ((double)(t1.tv\_nsec - t0.tv\_nsec));

acc\_time += diff\_time;

double temp\_avg\_time = acc\_time / (i + 1); // take average time

printf("Elapsed time: %.4lf nanoseconds\n", temp\_avg\_time);

pass = cmp\_arr(inA, inB, inC, outD);

}

if (pass == 1) {

fprintf(stdout, "----------Pass!------------\n");

return 0;

} else {

fprintf(stderr, "----------Fail!------------\n");

return 1;

}

}

На рисунке 17 представлены результаты запуска функции на ПК. Как видно из рисунка, среднее время выполнения функции равно 1611779119.00 нс.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 - Временные показатели для модифицированного теста

На рисунке 18 приведено сравнение лучшего аппаратного решения (sol6) с программной реализацией. Программная реализация оказалась почти в 1.5 раза быстрее, чем решение, полученное при синтезировании функции на лабораторном ПК при N = 512. При дальнейшем увеличении N (>2048), личный ПК будет проигрывать по времени лабораторному ПК.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 18 - Таблица данных для sol6 и программной реализации при N = 512

# Выводы

В ходе выполнения работы было произведено исследование реализации требуемого алгоритма в системе Vivado. По итогам исследования было выявлено оптимальное решение, использующее тактовое значение 12 наносекунд, а также директивы dataflow, array partition, resource ram\_2p, pipeline.

# Приложение

1. **kurs.tcl**

#############################################

# Lab #

#############################################

open\_project -reset kr\_02001\_4

set\_top kursFunc

add\_files ./source/kurs.c

add\_files -tb ./source/kurs\_test.c

open\_solution -reset sol1

create\_clock -period 6 -name clk

set\_clock\_uncertainty 0.1

set\_part {xa7a12tcsg325-1Q}

csim\_design

csynth\_design

#cosim\_design -trace\_level all

#############################################

# Solutions #

#############################################

set all\_solutions {sol2 sol3 sol4 sol5}

set all\_periods {{8} {10} {12} {16}}

foreach the\_solution $all\_solutions the\_period $all\_periods {

open\_solution -reset $the\_solution

create\_clock -period $the\_period -name clk

set\_clock\_uncertainty 0.1

set\_part {xa7a12tcsg325-1Q}

csim\_design

csynth\_design

#cosim\_design -trace\_level all

}

exit

1. **kurs.h**

#ifndef KURS\_IO\_H\_

#define KURS\_IO\_H\_

typedef int data\_sc;

#define N 8192

void kursFunc(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]);

#endif

1. **kurs.c**

#include "kurs.h"

void kursFunc(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

L3: for(int i = 0; i < N; i++) {

L2: for(int j = 0; j < N; j++) {

outD[i][j] = 0;

L1: for(int k = 0; k < N; k++) {

outD[i][j] += (inA[i][k] + inB[i][k]) \* inC[k][j];

}

}

}

}

1. **kurs\_test.c**

#include "kurs.h"

#include <stdio.h>

int cmp\_arr(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

data\_sc AC[N][N];

data\_sc BC[N][N];

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

data\_sc temp\_res = 0;

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

AC[i][j] = 0;

BC[i][j] = 0;

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

AC[i][j] += inA[i][k] \* inC[k][j];

BC[i][j] += inB[i][k] \* inC[k][j];

}

temp\_res = AC[i][j] + BC[i][j];

if (outD[i][j] != temp\_res) {

return 0;

}

}

}

return 1;

}

int main() {

int pass = 0;

data\_sc inA[N][N];

data\_sc inB[N][N];

data\_sc inC[N][N];

data\_sc outD[N][N];

for (int i = 0; i < 3; i++) {

//set inA inB inC

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

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

inA[j][k] = rand() % (N - 1);

inB[j][k] = rand() % (N - 1);

inC[j][k] = rand() % (N - 1);

}

}

kursFunc(inA, inB, inC, outD);

pass = cmp\_arr(inA, inB, inC, outD);

}

if (pass == 1) {

fprintf(stdout, "----------Pass!------------\n");

return 0;

} else {

fprintf(stderr, "----------Fail!------------\n");

return 1;

}

}

1. **kurs\_mod\_test.c**

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include "kurs.h"

int cmp\_arr(data\_sc inA[N][N], data\_sc inB[N][N], data\_sc inC[N][N], data\_sc outD[N][N]) {

data\_sc AC[N][N];

data\_sc BC[N][N];

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

data\_sc temp\_res = 0;

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

AC[i][j] = 0;

BC[i][j] = 0;

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

AC[i][j] += inA[i][k] \* inC[k][j];

BC[i][j] += inB[i][k] \* inC[k][j];

}

temp\_res = AC[i][j] + BC[i][j];

if (outD[i][j] != temp\_res) {

return 0;

}

}

}

return 1;

}

int main() {

int pass = 0;

data\_sc inA[N][N];

data\_sc inB[N][N];

data\_sc inC[N][N];

data\_sc outD[N][N];

struct timespec t0, t1;

double acc\_time = 0.0;

for (int i = 0; i < 32; i++) {

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

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

inA[j][k] = rand() % (N - 1);

inB[j][k] = rand() % (N - 1);

inC[j][k] = rand() % (N - 1);

}

}

if(clock\_gettime(CLOCK\_REALTIME, &t0) != 0) {

perror("Error in calling clock\_gettime\n");

exit(EXIT\_FAILURE);

}

kursFunc(inA, inB, inC, outD);

if(clock\_gettime(CLOCK\_REALTIME, &t1) != 0) {

perror("Error in calling clock\_gettime\n");

exit(EXIT\_FAILURE);

}

double diff\_time = (((double)(t1.tv\_sec - t0.tv\_sec)) \* 1000000000.0) + ((double)(t1.tv\_nsec - t0.tv\_nsec));

acc\_time += diff\_time;

double temp\_avg\_time = acc\_time / (i + 1); // take average time

printf("Elapsed time: %.4lf nanoseconds\n", temp\_avg\_time);

pass = cmp\_arr(inA, inB, inC, outD);

}

if (pass == 1) {

fprintf(stdout, "----------Pass!------------\n");

return 0;

} else {

fprintf(stderr, "----------Fail!------------\n");

return 1;

}

}