Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Вариант: 15

Выполнил студент гр. Ляшенко В.В.

гр.5140901/21502

Руководитель, доцент Антонов А.П.

«\_\_» \_\_\_\_\_\_ 2023

Санкт-Петербург

2023

Оглавление

[1. Вариант задания 3](#_Toc153808298)

[2. Этапы курсовой работы 3](#_Toc153808299)

[2.1. Этап 1 3](#_Toc153808300)

[2.2. Этап 2 3](#_Toc153808301)

[3. Исходный код функции 4](#_Toc153808302)

[4. Исходный код теста 4](#_Toc153808303)

[4.1. Моделирование 5](#_Toc153808304)

[5. Исследование вариантов аппаратной реализации 7](#_Toc153808305)

[6. Оптимизация функции 9](#_Toc153808306)

[6.1. Решение 2 9](#_Toc153808307)

[6.2. Решения 2\_1 и 2\_2 11](#_Toc153808308)

[6.3. Решение 2\_3 14](#_Toc153808309)

[6.4. Решение 2\_4 16](#_Toc153808310)

[6.5. Решение 2\_5 18](#_Toc153808311)

[6.6. Решение 2\_6 20](#_Toc153808312)

[6.7. Поиск максимального unroll factor для разных типов данных 23](#_Toc153808313)

[6.8. Реализация решений в TCL 24](#_Toc153808314)

[6.9. Сравнение решений 26](#_Toc153808315)

[7. Измерение времени выполнения 27](#_Toc153808316)

[7.1. Измерение времени выполнения на ПК 27](#_Toc153808317)

[7.2. Измерение времени выполнения на аппаратной реализации 30](#_Toc153808318)

[7.3. Сравнение результатов 30](#_Toc153808319)

[8. Исследовании производительности решения для разных типов данных 31](#_Toc153808320)

[8.1. Тип данных long long 31](#_Toc153808321)

[8.2. Тип данных float 33](#_Toc153808322)

[8.3. Тип данных double 34](#_Toc153808323)

[9. Вывод 35](#_Toc153808324)

[Приложение 1. Описание содержимого архива 37](#_Toc153808325)

1. Вариант задания

Даны вектора A, B, C, Х, размер N.

* Надо найти все элементы вектора r, размером N: r[i] равно значению многочлена A[i]x[i] 2+B[i]x[i]+C[i].
* N при моделировании, оптимизации и Co-simulation – 8.
* N для исследования производительности – 8192, 16384, 32768, 65 536.
* Надо: провести исследования оптимизированного решения для типов данных int, long long, float, double.

2. Этапы курсовой работы

2.1. Этап 1

1. Разработка описания создаваемой функции на языке С++.

2. Разработка теста на языке С++.

3. Моделирование созданной функции.

4. Исследование вариантов аппаратной реализации функции для периода тактового сигнала 6, 10, 14 нс (uncertainty = 1нс). Построение в xls таблицы и графиков. Выбор оптимального (мин аппаратных затрат, мах производительности) решения (периода тактового сигнала) для дальнейших исследований.

* При данном исследовании следует выключить конвейеризацию всех циклов.

5. Использование известных Вам директив, актуальных для оптимизации вашей функции: цель – мах производительность, ограничения – доступные аппаратные ресурсы микросхемы xc7a200tsbv484-2.

* FIFO vs RAM для реализации памяти массивов.
* UNROLL, Array\_Partition, Array\_Reshape, Pipeline, DataFlow (с выбором FIFO vs PingPong), Inline, Loop Merge. Следует привести соображения по выбору тех или иных директив для оптимизации.
  + Все, даже неудачные, попытки оптимизации желательно оформить отдельными решениями и представить в пояснительной записке.
* Для сравнения результатов следует заполнить таблицу и привести графики (решение <-> аппаратные затраты и быстродействие).

2.2. Этап 2

1. Создание теста для анализа временных затрат при программной реализации функции (на Вашем ПК). Тест должен содержать не менее 32 запусков исследуемой функции.

2. Оценка производительности аппаратного решения.

3. Оценка производительности программной реализации на ПК.

Для оценки выбираем медиану среди полученных при каждом из запусков функции 32 значений временных затрат.

* Надо реализовать:
  + одноядерное выполнение вашей функции на вашем ПК;
  + многоядерное выполнение.
* При исследовании производительности решения для типов данных int, long long, float, double, int256 исходные данные входных массивов должны быть псевдослучайными из всего диапазона соответствующих типов данных.

4. Сравнительный анализ программных и аппаратной реализаций (составить таблицу и привести графики).

3. Исходный код функции

Исходный код синтезируемой функции приведен на рисунке 1. Код заголовочного файла указан на рисунке 2.

*Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание*

*Рис.1. Исходный код функции kp\_502\_15*

Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

Автоматически созданное описание

*Рис.2. Исходный код заголовочного файла* *kp\_502\_15*

Функция принимает на вход 4 входных массива и один выходной. Каждый массив имеет размер N и тип данных int. Функция выполняет вычисление значений массива r[i] по формуле a[i]\*[x]^2+b[i]\*x[i]+c[i] в цикле. Значение x[i] используется в формуле несколько раз, поэтому чтобы сократить число считываний на одной итерации цикла, создаётся временная переменная temp\_x и в неё записывается значение x[i].

4. Исходный код теста

Исходный код теста для проверки функции kp\_502\_15 на рисунке 3. В тесте задаётся переменная pass, определяющая успешное/неуспешное выполнение теста (строка 15); 4 входных массива (строка 16); два выходных массива – ожидаемый и полученный (строки 17–18). Затем функция в цикле запускается три раза. Перед каждым запуском выходные массивы обнуляются вспомогательной функцией set\_zero (строки 22–23).

После этого во внутреннем цикле входные массивы заполняются случайными числами и формируется ожидаемый результат (строки 25–30). Запускается функция kp\_502\_15 (строка 33) и её значения сравниваются с ожидаемым результатом (строки 35–41). Если значения расходятся, то переменной pass присваивается значение 1 и в консоль выводится информация об ошибке. По завершению формируется сообщение об успешном/неуспешном выполнении теста, исходя из значения pass (строки 43–47).

4.1. Моделирование

Результаты моделирования исходного кода синтезируемой функции приведены на рисунке 4. Результаты моделирования показывают, что тест успешно пройден — так как условия теста выполняются (функция *main* возвращает 0) и результат расчета совпадает с ожидаемыми значениями.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

*Рис.3. Исходный код теста*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание*

*Рис.4. Лог результата выполнения моделирования*

Для проверки правильности работы теста в него было внесено изменение в ожидаемый результат - expted\_R[j] = in\_A[j] \* in\_X[j] \* in\_X[j] + in\_B[j] \* in\_X[j] + in\_C[j] + 1. Результат моделирования приведён на рисунке 5. Он показывает, что тест отрабатывает ошибку корректно.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

*Рис.5. Лог результата при ошибке в тесте*

5. Исследование вариантов аппаратной реализации

Для исследования аппаратной реализации функции было реализованы три решения с периодами тактового сигнала 6, 10, 14. Скрипт для запуска решений представлен в файле *kp\_502\_15.tcl* (рисунок 6).

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

*Рис.6. Исследование вариантов аппаратной реализации (kp\_502\_15.tcl)*

Временные параметры, производительность и аппаратные затраты этих решений представлены на рисунке 7.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.7. Сравнение решений с разным периодом*

На основе результатов сравнения решений была составлена электронная таблица и построен график, в котором для всех решений отображены: Latency, II и использованные ресурсы. Результаты представлены на рисунках 8–9.

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.8. Электронная таблица для всех решений с разным периодом*

Изображение выглядит как снимок экрана, линия, текст, График

Автоматически созданное описание

*Рис.9. График для всех решений с разным периодом*

Оптимальным решением с точки зрения максимальной производительности и минимальных аппаратных затрат является sol1\_2. На его основе будут проводиться дальнейшие исследования.

6. Оптимизация функции

6.1. Решение 2

На основе sol1\_2 создаётся решение sol2, в котором выполняется распараллеливание функции с помощью директивы unroll с factor=2. По умолчанию для данного решения используется двухпортовая память. Конвейеризация отключена. Результаты сравнения представлены на рисунке 10–12.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.10. Сравнение решений с применением unroll factor=2*

*Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.11. Результаты планировщика для решения sol1\_2 (базовое решение)*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.12. Результаты планировщика для решения sol2 (unroll 2)*

На рисунках 11–12 видно, что число происходит дублирование функции и каждая операция выполняется дважды за счёт unroll factor=2.

Использование распараллеливания с factor=2 вместе двухпортовой памятью, которая устанавливается по умолчанию, привело к улучшению производительности примерно в два раза. Но при этом аппаратные затраты выросли.

6.2. Решения 2\_1 и 2\_2

Теперь необходимо сравнить работу unroll с однопортовой и двухпортовой памятью. Для этого создаются два решения: в sol2\_1 явно задаётся двухпортовая память, а в sol2\_2 – однопортовая и директива array\_partition с factor=2 и type=cyclic. Результаты сравнения представлены на рисунке 13–15.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

*Рис.13. Сравнение решений с разными типами памяти*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.14. Результаты планировщика для решения sol2\_1 (ram\_2p)*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.15. Результаты планировщика для решения sol2\_2 (ram\_1p)*

Как можно видеть на рисунке 13, оба решения имеют одинаковую производительность, однако sol2\_2 с однопортовой памятью затрачивает немного меньше аппаратных затрат по FF и LUT. Поэтому в последующих решениях будет использоваться однопортовая память.

6.3. Решение 2\_3

Для данного решения unroll factor увеличивается до 4, а также используется директива array\_partition для всех массивов с параметрами factor=4 и type=cyclic, чтобы организовать параллельное чтение данных. Конвейеризации отключена. Результаты сравнения представлены на рисунке 16–17.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.16. Сравнение решений с применением unroll factor=4 (array\_partition)*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.17. Результаты планировщика для решения sol2\_3 (unroll 4, array\_partition)*

В данном решении число параллельно выполняемых операций выросло в двое относительно предыдущих решений, а соответственно и производительность выросла в два раза.

6.4. Решение 2\_4

Для данного решения unroll factor=4, а также вместо директивы array\_partition используется array\_reshape для всех массивов с параметрами factor=4 и type=cyclic. Конвейеризации отключена. Результаты сравнения представлены на рисунке 18–19.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.18. Сравнение решений с применением unroll factor=4 (array\_reshape)*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.19. Результаты планировщика для решения sol2\_4 (unroll 4, array\_reshape)*

Как можно видеть на рисунке 18, применение array\_reshape директивы незначительно ухудшило показатели производительности. Число тактов Latency относительно sol2\_3 с array\_partition не изменилось, а значение Estimated timing выросло. Для дальнейших решений будет использоваться array\_partition.

6.5. Решение 2\_5

Для данного решения unroll factor увеличивается до 8, что равно максимальному числе элементов в каждом массиве, а также используется директива array\_partition для всех массивов с параметрами factor=8 и type=cyclic, чтобы организовать параллельное чтение данных. Конвейеризации отключена. Результаты сравнения представлены на рисунке 20–21.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

*Рис.20. Сравнение решений с применением unroll factor=8 (array\_ partition)*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание*

*Рис.21. Результаты планировщика для решения sol2\_5 (unroll 8, array\_partition)*

В данном решении производительность выросла относительно sol2\_3 и цикл выполняется за одну итерацию.

6.6. Решение 2\_6

Данное решение основано на sol2\_5, однако в нём включена конвейеризация. Результаты сравнения представлены на рисунке 22.

Изображение выглядит как текст, снимок экрана, музыка

Автоматически созданное описание

*Рис.22. Сравнение решений с применением unroll factor=8 pipeline (array\_ partition)*

Применение конвейеризации не даёт никаких изменений в производительности, так как весь цикл выполняется за одну итерацию (рисунок 23). Однако при дальнейшем исследовании на больших значениях N, конвейеризация требуется, для улучшения производительности.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.23. Результаты планировщика для решения sol2\_6 (unroll 8, array\_partition, pipeline)*

6.7. Поиск максимального unroll factor для разных типов данных

По результатам решений sol2-sol2\_6 видно, что с ростом unroll factor производительность возрастает, но также растут и аппаратные затраты. Необходимо определить, до какого значения можно увеличить значение unroll factor, чтобы задействовать все умножители использующейся микросхемы. На рисунке 24 видно, что микросхема имеет 740 умножителей и 134600 LUT.

Изображение выглядит как снимок экрана, текст, линия, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.24. Аппаратные ресурсы микросхемы*

Все предложенные для исследования типы данных будут иметь отличный максимальный unroll factor. Поэтому необходимо создать 4 решения sol2\_7-sol2\_10 в каждом из которых будет подобран unroll фактор для конкретного типа данных. Исследование проводится на больших размерах данных, так как изначальный размер массива равен N=8.

Решение для типа int – sol2\_7. Размер данных N=64. В результате исследования было установлено, что для массивов типа int максимальной производительности с учётом возможностей микросхемы достигает unroll factor=64. В таком случае DSP=384, а LUT=7693 (рисунок 25).

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

*Рис.25. Максимальный unroll factor для int*

Аналогичные исследования для других типах данных привели к следующим результатам. Размер массива N в исследовании соответствует unroll factor для каждого из типов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **sol** | **Тип** | **Unroll** | **Estimated Timing** | **DSP** | **FF** | **LUT** |
| sol2\_7 | int | 64 | 11.240 | 384 | 2050 | 7693 |
| sol2\_8 | long long | 32 | 10.450 | 384 | 6148 | 7573 |
| sol2\_9 | float | 64 | 10.252 | 512 | 39116 | 33525 |
| sol2\_10 | double | 16 | 11.825 | 400 | 19840 | 19409 |

6.8. Реализация решений в TCL

В результате выполнения оптимизаций, был создан TCL скрипт *kp\_502\_15\_2.tcl*, представленный на рисунке 26 (старая нумерация 25). В данном скрипе представлены базовое решение с периодом 14 нс и решения, полученные в ходе оптимизации.

Решение sol1\_2 – базовое решение с периодом 14 нс и отключенной конвейеризацией.

Решение sol2 – выполнение первой оптимизации. Распараллеливание функции с помощью unroll factor=2. Конвейеризация отключена.

Решения sol2\_1-sol2\_2 – решения для сравнения влияния типа памяти на производительность. В sol2\_1 заданы unroll factor=2 и двухпортовая память. В решении sol2\_2 - unroll factor=2, array\_partition cyclic, factor=2 и однопортовая память. Конвейеризация отключена.

Решения sol2\_3-sol2\_4 – решение для сравнения влияния array\_partition и array\_reshape на производительность. В sol2\_3 - unroll factor=4, array\_partition cyclic, factor=4. В sol2\_4 - unroll factor=4, array\_reshape cyclic, factor=4. Конвейеризация отключена.

Решение sol2\_5 – повышение unroll factor до максимального значения, при размере массивов N=8, array\_partition cyclic, factor=8. Конвейеризация отключена.

Решение sol2\_6 – включение конвейеризации относительно sol2\_5, unroll factor=8, array\_partition cyclic, factor=8.

Решения sol2\_7-sol2\_10 – решения с максимальным unroll factor для каждого из 4 типов данных, предложенных к исследованию. Исследование проводится на массивах больших размеров (значение N в h-файле увеличено). Значения unroll factor определены экспериментальным путём. Каждое из 4 решений соответствует своему типу данных: sol2\_7 – int, sol2\_8 – long long, sol2\_9 – float, sol2\_10 – double.

Изображение выглядит как текст, снимок экрана, документ, число

Автоматически созданное описание

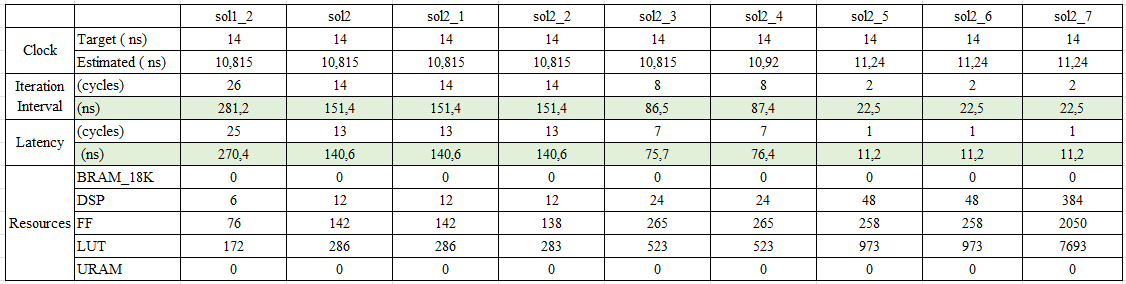
Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

*Рис.26. Поиск оптимального решения (kp\_502\_15\_2.tcl)*

6.9. Сравнение решений

На основе результатов сравнения решений была составлена электронная таблица и построен график, в котором для всех решений отображены: Latency, II и использованные ресурсы. Результаты представлены на рисунках 27–28.



*Рис.27. Электронная таблица для всех решений с разным unroll*

Изображение выглядит как снимок экрана, линия, График, диаграмма

Автоматически созданное описание

*Рис.28. График для всех решений с разным unroll*

С точки зрения производительности оптимальными решениями являются sol2\_5-sol2\_7, так как они имеют наименьшие значения Latency и затрачивают меньше времени. Это обусловлено использованием директив оптимизации. Однако решение sol2\_7 может быть более выигрышным в том случае, если значение N будет увеличено и не будет равно числу элементов массива, тогда будет применена конвейеризация, а также число параллельно обрабатываемых элементов будет больше, чем в остальных двух решениях.

7. Измерение времени выполнения

7.1. Измерение времени выполнения на ПК

Измерение времени выполнения синтезируемой функции было выполнено на ПК со следующими характеристиками:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип процессора** | **Базовая частота работы** | **Максимальная частота работы** | **ОЗУ** |
| 12th Gen Intel(R) Core(TM) i5-12450H | 2 ГГц | 4,4 ГГц | 16 Гб |

На базе исходного теста был создан отдельный, модернизированный, тест kp\_502\_15\_testSW.cpp, который содержит операторы измерения времени выполнения и имеет большее число запусков равное 32. Код теста представлен на рисунке 29.

В нём выделяется память для входных массивов (строки 24–27), задаются переменные для измерения времени (строки 31–33) и число запусков функции равное 32 (строка 43).

Затем в цикле аналогично тесту для аппаратной реализации очищаются выходные массивы (строки 38–39), случайным образом заполняются входные массивы и формируется ожидаемый результат (строки 41–47). После этого выполняется запуск функции и замер времени её выполнения (строки 49–51). Полученный результат сравнивается с ожидаемым (строки 53–58). Время выполнения итерации переводится в нс и сохраняется в векторе (строки 60–64).

После всех запусков функции полученные временные результаты сортируются (строка 72) и на их основе вычисляются минимальное, максимальное и медианное время выполнения (74–76).

Изображение выглядит как текст, снимок экрана, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

*Рис.29. Код теста* *kp\_502\_15\_testSW.cpp*

Исходная функция на рисунке 1, представляет собой однопоточную реализацию на ПК. Для того, чтобы создать многопоточную реализацию, необходимо задействовать библиотеку <omp.h>, которая позволит использовать pragma и распараллелить цикл между заданным числом потоков.

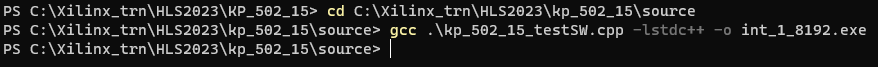
Для этого следует изменить исходную функцию и в исходный код для реализации многопоточности добавить pragma omp со значением num\_threads=8, что соответствует числу ядер на ПК (рисунок 30). При однопоточном исследовании, а также при запуске аппаратной реализации эту pragma и библиотеку следует закомментировать.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

*Рис.30. Исходный код многопоточной функции kp\_502\_15*

Для компиляции теста был установлен компилятор GCC версии 13.2.0. После этого была выполнена компиляции на одном ядре и на 8 для четырёх значений N – 8192, 16384, 32768, 65536. По результатам компиляции были получены исполняемы exe файлы (рисунок 31).



*Рис.31. Выполнение компиляции теста*

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 17906.2 | 19750 | 19000 |
| 16384 | 34833.3 | 42000 | 38461.5 |
| 32768 | 66000 | 79000 | 68875 |
| 65536 | 130462 | 270000 | 158800 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 20312.5 | 30000 | 24235.3 |
| 16384 | 36093.8 | 63000 | 41437.5 |
| 32768 | 54600 | 58750 | 57269.2 |
| 65536 | 130781 | 259000 | 144118 |

7.2. Измерение времени выполнения на аппаратной реализации

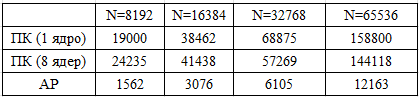
Для измерения времени используется аппаратная реализация sol2\_7. Данная реализация имеет следующие директивы: unroll factor=64, array\_partition cyclic, factor=64, конвейеризация включена. Решение является оптимальным для типа данных int.

Синтез решения sol2\_7 был осуществлён для четырёх значений N – 8192, 16384, 32768, 65536. Время выполнения = II \* Estimated period для каждого случая:

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 132 | 11.832 | 1561.824 |
| 16384 | 260 | 11.832 | 3076.32 |
| 32768 | 516 | 11.832 | 6105.312 |
| 65536 | 1028 | 11.832 | 12163.296 |

7.3. Сравнение результатов

На основе результатов, полученных при измерениях времени выполнения на ПК и аппаратной реализации, была составлена электронная таблица и построен график, в котором отражено соотношение размера массивов данных задачи (N) ко времени выполнения. Результаты представлены на рисунках 32–33.



*Рис.32. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации (тип int)*

Изображение выглядит как снимок экрана, линия, График, Параллельный

Автоматически созданное описание

*Рис.33. График для ПК (1 и 8 ядер) и аппаратной реализации (тип int)*

На графики видно, что аппаратная реализация значительно выигрывает по производительности у обеих реализаций на ПК за счёт распараллеливания и конвейеризации. При этом однопоточная и многопоточная реализации на ПК на малых значениях имеют приблизительно одинаковые значения производительности, но на больших многопоточная начинает опережать однопоточную.

8. Исследовании производительности решения для разных типов данных

Необходимо выполнить аналогичные измерения времени на ПК и аппаратной реализации для других типов данных.

8.1. Тип данных long long

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 15677.4 | 19000 | 16833.3 |
| 16384 | 31000 | 45000 | 44066.7 |
| 32768 | 72714.3 | 138000 | 88142.9 |
| 65536 | 134500 | 176231 | 164409 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 21093.8 | 32000 | 24187.5 |
| 16384 | 44724.1 | 49461.5 | 47500 |
| 32768 | 64400 | 77000 | 66750 |
| 65536 | 125750 | 143536 | 140667 |

Результаты синтеза аппаратного решения sol2\_8. Данная реализация имеет следующие директивы: unroll factor=32, array\_partition cyclic, factor=32, конвейеризация включена. Решение является оптимальным для типа данных long long.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 263 | 10.45 | 2748.35 |
| 16384 | 519 | 10.45 | 5423.55 |
| 32768 | 1031 | 10.45 | 10773.95 |
| 65536 | 2055 | 10.45 | 21474.75 |

Сравнение результатов long long (рисунки 34–35).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

*Рис.34. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации (тип long long)*

Изображение выглядит как снимок экрана, линия, График, скат

Автоматически созданное описание

*Рис.35. График для ПК (1 и 8 ядер) и аппаратной реализации (тип long long)*

На графике можно видеть, что при типе данных long long аппаратная реализация начиная с малых значений превосходит реализации на ПК по производительности. На малых значениях однопоточная и многопоточная реализации довольно близки по производительности, но однопоточная выигрывает. При этом на больших значениях, начиная с N=32768, происходит перегиб, и многопоточная имеет лучшую производительность.

8.2. Тип данных float

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 19218.8 | 24000 | 22000 |
| 16384 | 38250 | 48000 | 41958.3 |
| 32768 | 70655.2 | 103000 | 73125 |
| 65536 | 148438 | 236000 | 165312 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 25406.2 | 40833.3 | 33277.8 |
| 16384 | 47187.5 | 134000 | 59875 |
| 32768 | 63000 | 75909.1 | 70409.1 |
| 65536 | 134000 | 156286 | 139677 |

Результаты синтеза аппаратного решения sol2\_9. Данная реализация имеет следующие директивы: unroll factor=64, array\_partition cyclic, factor=64, конвейеризация включена. Решение является оптимальным для типа данных float.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 143 | 11.569 | 1654.367 |
| 16384 | 271 | 11.569 | 3135.199 |
| 32768 | 527 | 11.569 | 6096.863 |
| 65536 | 1039 | 11.569 | 12020.191 |

Сравнение результатов float (рисунки 36–37).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

*Рис.36. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации (тип float)*

Изображение выглядит как снимок экрана, линия, График, Параллельный

Автоматически созданное описание

*Рис.37. График для ПК (1 и 8 ядер) и аппаратной реализации (тип float)*

На графике можно видеть, что при типе данных float аппаратная реализация начиная с малых значений значительно превосходит реализации на ПК по производительности. На малых значениях однопоточная имеет значительны отрыв от многопоточной. При этом на больших значениях, начиная с N=32768, происходит перегиб, и многопоточная имеет лучшую производительность.

8.3. Тип данных double

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 18000 | 26000 | 21388.9 |
| 16384 | 39218.8 | 47181.8 | 43562.5 |
| 32768 | 81937.5 | 154000 | 87200 |
| 65536 | 165036 | 245000 | 179909 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 23677.4 | 37800 | 28562.5 |
| 16384 | 42437.5 | 97000 | 47812.5 |
| 32768 | 65000 | 80142.9 | 77958.3 |
| 65536 | 152519 | 192000 | 155684 |

Результаты синтеза аппаратного решения sol2\_10. Данная реализация имеет следующие директивы: unroll factor=16, array\_partition cyclic, factor=16, конвейеризация включена. Решение является оптимальным для типа данных double.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 532 | 11.825 | 6290.9 |
| 16384 | 1044 | 11.825 | 12345.3 |
| 32768 | 2068 | 11.825 | 24454.1 |
| 65536 | 4116 | 11.825 | 48671.7 |

Сравнение результатов float (рисунки 38–39).

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

*Рис.38. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации (тип double)*

Изображение выглядит как снимок экрана, линия, График, скат

Автоматически созданное описание

*Рис.39. График для ПК (1 и 8 ядер) и аппаратной реализации (тип double)*

На графике можно видеть, что при типе данных double аппаратная реализация так же, как и на других типах данных выигрывает, но производительность ухудшается быстрее. При этом различие между многопоточной и однопоточной реализацией не очень большое, но с ростом N разрыв увеличивается.

9. Вывод

В результате выполнения данной курсовой работы были закреплены знания по оптимизации аппаратных реализаций программ. Для этого на языке C++ была написана программа, для поиска элементов массива r, которые равны значению многочлена.

На этой программе было выполнено исследование вариантов аппаратной реализации с разными периодами тактового сигнала. Наилучшее значение с точки зрения производительности и аппаратных затрат было решение с периодом 14 нс.

На основе данного решения была выполнена оптимизации функции. Для этого использовались директивы Unroll, Array\_Partition, Array\_Reshape, Pipeline и RAM. Для N=8 лучшие результаты показало решение с unroll factor=8 и конвейеризацией. Также был выявлен максимальный unroll factor, который позволял получить максимальную производительность для 4 типов данных. Этот unroll factor использовался при замере времени.

Далее был выполнен анализа временных затрат при программной и аппаратной реализациях функции. Были исследованы разные размеры массивов и типы данных. В результате было выявлено, что для всех четырёх рассматриваемых типов аппаратная реализация является лучшей, однако при значениях long long разрыв по производительности с реализациями на ПК наименьший.

Приложение 1. Описание содержимого архива

Архив проекта содержит следующие элементы.

Папка doc – папка, которая содержит файлы, предназначенные для описания проекта. Здесь находится kp\_502\_15.xlsx – файл с графиками и таблицами, построенными на основе анализа решений, и ПРГВС\_КР\_21502\_Ляшенко.docx – отчет по курсовой работе в редактируемом формате.

Папка kp\_502\_15 – папка проекта. Содержит все созданные с помощью tcl скриптов решения (sol1-sol1\_2 и sol2-sol2\_10).

Папка source – папка, которая содержит все исходные коды. Файл с исходным кодом функции kp\_502\_15.cpp, заголовочный файл kp\_502\_15.h с типами данных массивов и размерностью, тестовый файл для аппаратной реализации kp\_502\_15\_test.cpp, тестовый файл для реализации на ПК kp\_502\_15\_testSW.cpp. Также source содержит внутреннюю папку test\_SW, в которой находятся исполняемые файлы для всех вариаций решений в формате exe. Они распределены по папкам в соответствии с исследуемым типом данных: int, long, float, double. Пример исполняемого файла: int\_1\_8192.exe – тип данных int, число потоков 1, размер N=8192.

Также архив содержит 2 tcl скрипта: kp\_502\_15.tcl и kp\_502\_15\_2.tcl. Скрипт kp\_502\_15.tcl предназначен для создания и исследования решений с разным периодом (решения sol1-sol1\_2). Скрипт kp\_502\_15\_2.tcl предназначен для создания и исследования оптимизирующих исходную функцию (решения sol2-sol2\_10).