Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА/GRADUATION THESIS

Разработка механизмов синтеза в системе автоматического проектирования специализированных вычислителей

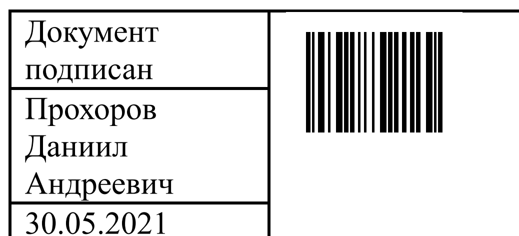
Автор/ Author
Прохоров Даниил Андреевич

Направленность (профиль) образовательной программы/Major
Программно-информационные системы 2017
Квалификация/ Degree level
Бакалавр

Руководитель ВКР/ Thesis supervisor
Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Группа/Group
P3417
Факультет/институт/кластер/ Faculty/Institute/Cluster
факультет программной инженерии и компьютерной техники
Направление подготовки/ Subject area
09.03.04 Программная инженерия

Обучающийся/Student

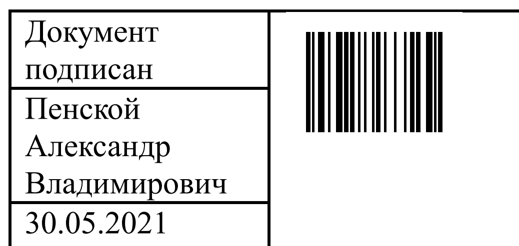


(эл. подпись/ signature)

Прохоров Даниил Андреевич

(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor



(эл. подпись/ signature)

Пенской Александр Владимирович

(Фамилия И.О./ name and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ / OBJECTIVES FOR A GRADUATION THESIS

Обучающийся / Student Прохоров Даниил Андреевич
Группа/Group P3417
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Квалификация/ Degree level Бакалавр
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Направленность (профиль) образовательной программы/Major Программно-информационные системы 2017
Специализация/ Specialization
Тема ВКР/ Thesis topic Разработка механизмов синтеза в системе автоматического проектирования специализированных вычислителей
Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Срок сдачи студентом законченной работы до / Deadline for submission of complete thesis 31.05.2021

Техническое задание и исходные данные к работе/ Requirements and premise for the thesis
Цель: повышение эффективности процесса синтеза в САПР специализированных вычислителей путем разработки новых механизмов анализа и преобразования модели целевого вычислителя, где эффективность это совокупность длительности процесса синтеза и длительность исполнения целевого вычислительного процесса. Задачи: 1) Анализ предметной области и «узких» мест в САПР. 2) Разработка механизма свертки констант. 3) Разработка механизмов оптимизации прикладного алгоритма под особенности аппаратной реализации специализированного процессора. 4) Разработка механизмов эффективного распределения функций прикладного алгоритма между вычислительными блоками специализированного процессора. 5) Разработка средств формирования набора данных для применения машинного обучения. 6) Оценка эффективности реализованных механизмов оптимизаций.

Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)/ Content of the thesis (list of key issues)
1) Введение. Постановка задачи.

2) Процесс синтеза в САПР специализированных вычислителей.

3) Механизмы оптимизации целевого алгоритма.

4) Механизмы оптимизации процесса синтеза.

5) Формирование наборов данных для машинного обучения.

6) Анализ эффективности.

7) Заключение.

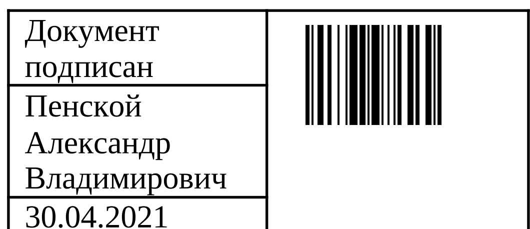
Перечень графического материала (с указанием обязательного материала) / List of graphic materials (with a list of required material)
Презентация работы

Исходные материалы и пособия / Source materials and publications
1) Пенской А.В. Разработка и исследование архитектурных стилей проектирования уровневой организации встроенных систем : ... канд. техн. наук : 05.13.12 / Пенской А.В. - Санкт-Петербург, 2016. - 169 с.
2) Prohorov D., Penskoi A. Verification of the CAD System for an Application-Specific Processor by Property-Based Testing //2020 9th Mediterranean Conference on Embedded Computing (MECO). – IEEE, 2020. – С. 1-4.
3) Душкин Р. В. Функциональное программирование на языке Haskell. М.: ДМК Пресс, 2007.

Дата выдачи задания/ Objectives issued on 30.04.2021

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/ Thesis supervisor



(эл. подпись)

Пенской Александр Владимирович

Задание принял к исполнению/ Objectives assumed by

(эл. подпись)

Прохоров Даниил Андреевич

Руководитель ОП/ Head of educational program

(эл. подпись)

Муромцев Дмитрий Ильич

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

АННОТАЦИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ / SUMMARY OF A GRADUATION THESIS

Обучающийся/ Student
Прохоров Даниил Андреевич
Наименование темы ВКР / Title of the thesis
Разработка механизмов синтеза в системе автоматического проектирования специализированных вычислителей
Наименование организации, где выполнена ВКР/ Name of organization
Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ/ DESCRIPTION OF THE GRADUATION THESIS

1. Цель исследования / Research objective
Повышение эффективности процесса синтеза в САПР специализированных вычислителей путем разработки новых механизмов анализа и преобразования модели целевого вычислителя, где под эффективностью понимается длительность процесса синтеза и длительность исполнения целевого вычислительного процесса.
2. Задачи, решаемые в ВКР / Research tasks
1) анализ предметной области, поиск критических мест в САПР для применения оптимизаций; 2) разработка механизма свертки констант; 3) разработка механизмов оптимизации прикладного алгоритма под особенности аппаратной реализации специализированного процессора; 4) разработка механизмов эффективного распределения функций прикладного алгоритма между вычислительными блоками специализированного процессора; 5) разработка средств формирования набора данных для применения машинного обучения; 6) оценка результатов применённых оптимизаций.
3. Краткая характеристика полученных результатов / Short summary of results/conclusions
Проведен анализ критических мест и методов оптимизации в САПР. Система была дополнена новыми опциями процесса синтеза, которые позволили оптимизировать работу целевого вычислителя, где критерий эффективности – длительность работы целевого вычислителя. Реализован новый механизм связывания функций прикладного алгоритма с вычислительными блоками, удалось оптимизировать работу процесса синтеза, где критерий эффективности – длительность процесса синтеза. Была реализована функциональность в виде сбора данных для оценки результата работы процесса синтеза, которая будет актуальна для дальнейшего развития механизмов процесса синтеза.
4. Наличие публикаций по теме выпускной работы/ Have you produced any publications on the topic of the thesis

1. Прохоров Д.А., Пенской А.В. Оптимизация целевого алгоритма в системе автоматического проектирования специализированных вычислителей NITTA.// Сборник тезисов докладов конгресса молодых ученых. - 2021 (Тезисы)
2. Prohorov D., Penskoi A. Target algorithm optimisation for a custom processor unit in the ASIP//Сборник трудов XII международной научно-практической конференции «Программная инженерия и компьютерная техника (Майоровские чтения)» (СПб, 10-11декабря 2020г.), 2021, pp. 6 (Статья; РИНЦ)
3. Прохоров Д.А., Вахвиянова П.Д., Пенской А.В. Верификация работы системы автоматического проектирования специализированных вычислителей//Сборник тезисов докладов конгресса молодых ученых. - 2020 (Тезисы)
4. Prohorov D., Penskoi A. Verification of the CAD System for an Application-Specific Processor by Property-Based Testing//9th Mediterranean Conference on Embedded Computing, MECO 2020 - Proceedings, 2020, pp. 9134312 (Статья; Scopus, Web of Science)

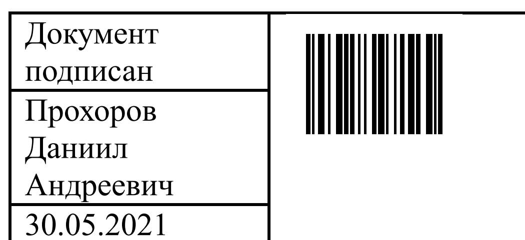
5. Наличие выступлений на конференциях по теме выпускной работы/ Have you produced any conference reports on the topic of the thesis

1. X Конгресс молодых ученых 2021, 14.04.2021 - 17.04.2021 (Конгресс, статус - всероссийский)
2. 12th The Majorov International Conference on Software Engineering and Computer Systems (MICSECS 2020), 10.12.2020 - 11.12.2020 (Конференция, статус - международный)
3. IX Конгресс молодых ученых, 15.04.2020 - 18.04.2020 (Конгресс, статус - всероссийский)
4. XLIII научная и учебно-методическая конференция ППС Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, 29.01.2019 - 01.02.2019 (Конференция, статус - региональный)

6. Полученные гранты, при выполнении работы/ Grants received while working on the thesis

7. Дополнительные сведения/ Additional information

Обучающийся/Student



(эл. подпись/ signature)

Прохоров Даниил Андреевич

(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor

(эл. подпись/ signature)

Пенской Александр Владимирович

(Фамилия И.О./ name and surname)

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ......................8

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ...9

ВВЕДЕНИЕ...10

Постановка задачи..14

1 ПРОЦЕСС СИНТЕЗА В САПР СПЕЦИАЛИЗИРОВАННЫХ ВЫЧИСЛИТЕЛЕЙ..15

2 МЕХАНИЗМЫ ОПТИМИЗАЦИИ ЦЕЛЕВОГО АЛГОРИТМА...............18

2.1 Оптимизация аккумулятора..18

2.2 Свёртка констант...23

2.3 Вывод...26

3 МЕХАНИЗМЫ ОПТИМИЗАЦИИ ПРОЦЕССА СИНТЕЗА.....................26

3.1 Эффективное распределение функций прикладного алгоритма между вычислительными блоками..27

3.2 Формирование наборов данных для машинного обучения..........30

3.3 Вывод...33

4 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ...33

4.1 Анализ эффективности целевого вычислителя при наличии механизма оптимизации сумматора в процессе синтеза....................33

4.2 Анализ эффективности целевого вычислителя при наличии механизма свёртки констант в процессе синтеза................................36

4.3 Анализ результатов группового связывания.................................39

4.4 Результаты сбора данных для оценки результата работы процесса синтеза..41

4.5 Вывод...42

ЗАКЛЮЧЕНИЕ...43

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..44
ПРИЛОЖЕНИЕ А (ПРИМЕНЕНИЕ ОПТИМИЗАЦИИ АККУМУЛЯТОРА В ПРОЦЕССЕ СИНТЕЗА)..45
ПРИЛОЖЕНИЕ Б (ПРИМЕНИНИЕ СВЕРТКИ КОНСТАНТ В ПРОЦЕССЕ СИНТЕЗА)..46
ПРИЛОЖЕНИЕ В (ЛИСТИНГ ВЫЧИСЛИТЕЛЬНОГО БЛОКА СУММАТОРА)..47
ПРИЛОЖЕНИЕ Г (ЛИСТИНГ ОПТИМИЗАЦИЯ АККУМУЛЯТОРА).....53
ПРИЛОЖЕНИЕ Д (ЛИСТИНГ СВЁРТКА КОНСТАНТ)............................56
ПРИЛОЖЕНИЕ Е (ЛИСТИНГ ГРУППОВОЕ СВЯЗЫВАНИЕ)..................58
ПРИЛОЖЕНИЕ Ж (ЛИСТИНГ СБОР ДАННЫХ ДЛЯ АНАЛИЗА ДЕРЕВА СИНТЕЗА)..60

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

САПР – система автоматического проектирования

NITTA – Not Instruction Transport Triggered Architecture

HLS – High-level Synthesis

API – Application Programming Interface

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Взаимная блокировка – ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать своё выполнение [1].

Оптимизация – процесс максимизации выгодных характеристик и минимизации расходов [2].

Синтез – инженерное построение сложных систем из предварительно подготовленных блоков или модулей разных типов.

Атомарная операция – в системе автоматического проектирования NITTA это операция, исполняющаяся без дополнительных пересылок данных во время вычисления. Работает по по следующему алгоритму: загрузка входных данных, вычисление, выгрузка выходных данных.

ВВЕДЕНИЕ

Процесс разработки высокопроизводительных систем реального времени, достаточно сложный и трудоёмкий, потому что предполагает понимание работы системы на низком уровне (для обеспечения высокой производительности и нахождении областей, где можно произвести оптимизацию). Для того, чтобы добиться высокой степени контроля над получаемым решением, разработчикам приходится использовать низкоуровневые средства разработки. Это усложняет и замедляет процесс разработки конечной системы, а также не даёт гарантий работоспособности получаемого результата.

Сфера вычислителей специального назначения развивается достаточно давно и представляет собой программно-аппаратный комплекс для создания различных вычислителей специального назначения, которые используются в определённых местах, где требуется высокая производительность системы [3]. Например, такими специализиванными сопроцессорами являются аппаратные декодеры видеосигнала, аппаратные ускорители, конвертеры сигнала и другие устройства.

Для решения данных задач существуют высокоуровневые системы синтеза (High-Level Synthesis), которые принимают на вход высокоуровневый код на одном из языков программирования и возвращают оптимизированную модель спецвычислителя для заданного алгоритма. Однако такие программы также имеют свои недостатки: не всегда можно явно проследить за ходом принятия решений и правильности выбора данных решений в процессе синтеза.

Поэтому, программирование с помощью таких систем усложняется дополнительными шаблонами, которые следует соблюдать при построении программы, для получения нужного результата. Отдельной проблемой таких систем, являются долгие, объёмные вычисления при изменении входного алгоритма, что усложняет процесс отладки программы, а зачастую делает его невозможным, потому что небольшие изменения во входном алгоритме сильно изменяют итоговую модель спецвычислителя.

Для упрощения разработки спецвычислителя была разработана система автоматического проектирования специализированных вычислителей. Она предполагает использование архитектуры NITTA (Not Instruction Transport Triggered Architecture) (Рисунок 1) [4], которая позволяет реализовывать и верифицировать различные системы, соблюдая баланс между производительностью и площадью вычислителя, при этом пользователь может следить за процессом принятия решений в процессе синтеза и принимать решения самостоятельно, если данная опция может потребоваться при разработке.

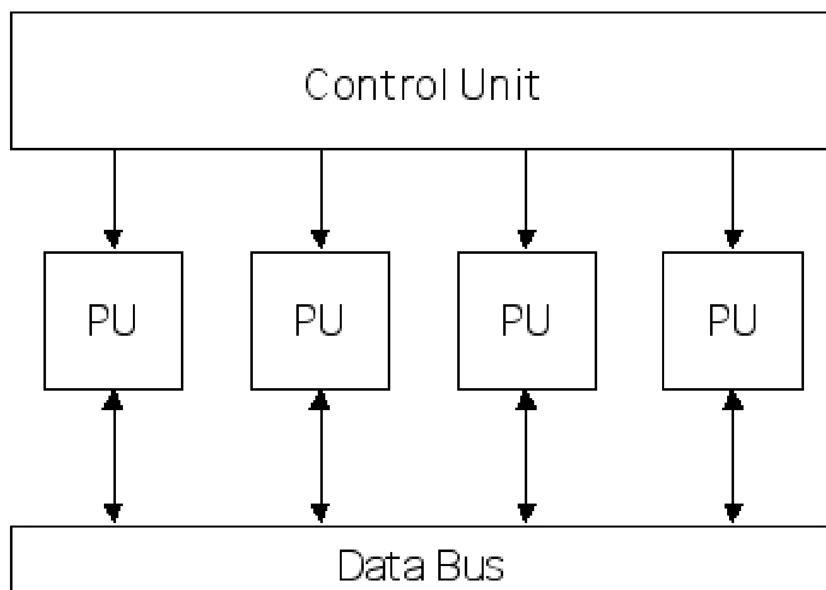


Рисунок 1 – Архитектура процессоров NITTA

Вычислитель, построенный на основе архитектуры NITTA состоит из нескольких простых блоков, взаимодействующих между собой: управляющий блок, вычислительный блок, шина данных. Вычислительный блок — основная действующая ячейка, реализующая логику работы различных операций, например: сложение, вычитание, деление, умножение, взаимодействие с внешним миром (SPI), логический сдвиг, хранение данных (Fram) и другие. Общая шина данных является связующим звеном между вычислительными блоками, позволяя им взаимодействовать между собой. Управляющий блок используется для подачи управляющих сигналов на каждый вычислительный блок в нужное время, согласно рассчитанному во время процесса синтеза расписанию.

САПР NITTA позволяет упростить процесс разработки специализированного вычислителя для прикладного программиста, исключая необходимость знания языков описания аппаратуры для разработки специализированных вычислителей(Verilog, VHDL), или сводя количество таких специалистов к минимуму, за счёт автоматизации процесса синтеза и верификации целевого вычислителя [5, 6].

Процесс синтеза – одна из основных операций в САПР специализированных вычислителей, которая напрямую влияет на итоговый результат работы системы. Поэтому разработка механизмов для процесса синтеза является актуальной задачей и позволяет увеличить количество опций процесса синтеза, что даёт возможность реализовать более эффективный целевой вычислитель.

Система автоматического проектирования NITTA представляет собой многоуровневое (аппаратура, сервер, клиент) программное обеспечение со сложной последовательностью взаимодействий (Рисунок 2).

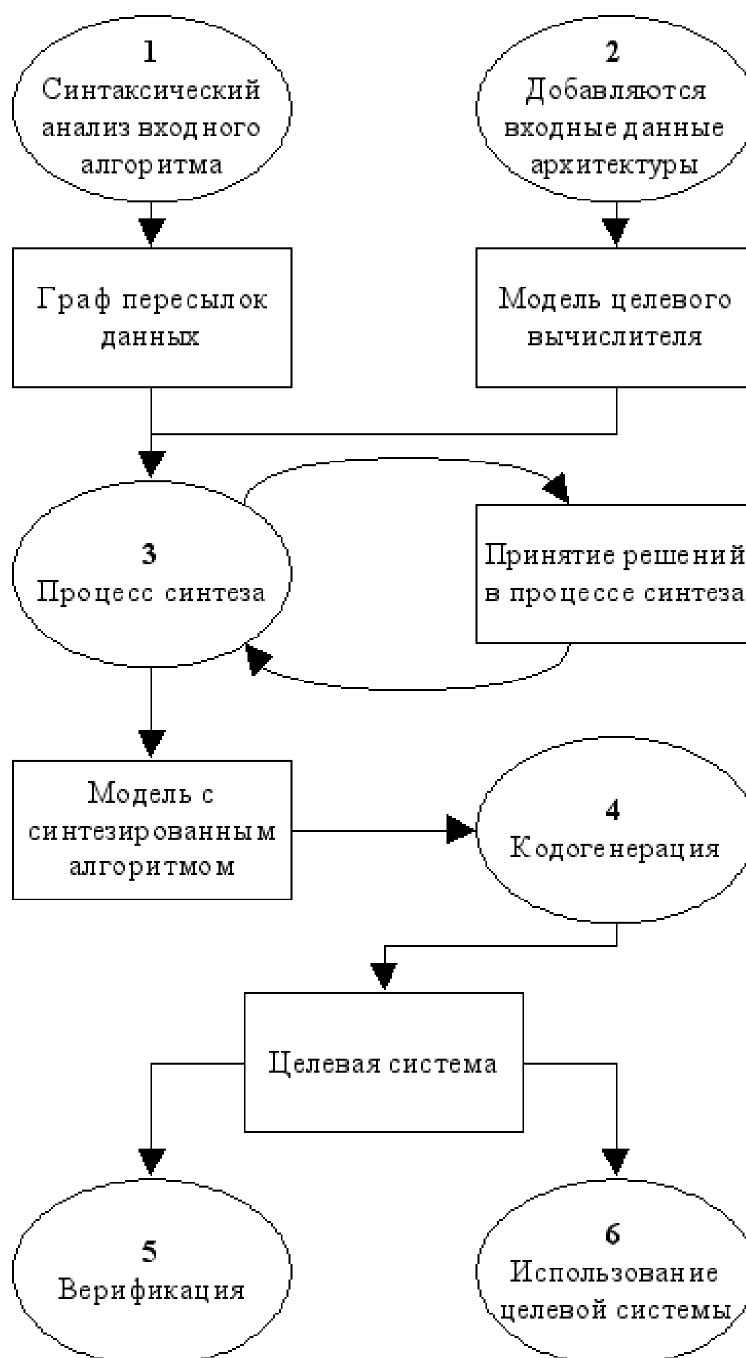


Рисунок 2 – Описание процесса работы САПР NITTA

При каждом использовании САПР NITTA выполняются следующие действия:

1. на вход подаётся текст программы, после чего применяется синтаксический анализатор, для получения абстрактного

синтаксического дерева, которое описывает входной алгоритм программы на языке программирования *Lua*, после чего оно преобразуется в граф пересылок данных;

2. задаётся микроархитектура вычислителя, включающая в себя список вычислительных блоков, их соединение, ширина шины обмена данными и другие специфические для каждого вычислительного блока параметры;

3. выполняется процесс синтеза целевого алгоритма под особенности модели вычислителя, на каждом этапе принимаются определённые решения о выборе операции в данный момент времени на основе метрик, где метрики – данные о текущем состоянии процесса синтеза, графе пересылок данных, модели вычислительной системы, позволяющие принимать правильные решения в процессе синтеза;

4. модель с синтезированным алгоритмом компилируются в код описания аппаратуры *Verilog*;

5. происходит процесс верификации целевой системы, позволяющий проверить соответствие функциональной и целевой модели вычислительной системы;

6. использование целевой системы.

Постановка задачи

Целью работы является повышение эффективности процесса синтеза в САПР специализированных вычислителей путём разработки новых механизмов анализа и преобразования модели целевого вычислителя, где под

эффективностью понимается длительность процесса синтеза и длительность исполнения целевого вычислительного процесса.

Для достижения поставленной цели требуется выполнить следующие задачи:

- анализ предметной области, поиск критических мест в САПР;
- разработка механизма свёртки констант;
- разработка механизмов оптимизации прикладного алгоритма под особенности аппаратной реализации специализированного процессора;
- разработка механизмов эффективного распределения функций прикладного алгоритма между вычислительными блоками специализированного процессора;
- разработка средств формирования набора данных для применения машинного обучения;
- оценка результатов применённых оптимизаций.

1 ПРОЦЕСС СИНТЕЗА В САПР СПЕЦИАЛИЗИРОВАННЫХ ВЫЧИСЛИТЕЛЕЙ

Основная задача САПР NITTA – построение вычислителя на основе входного алгоритма. Для реализации данной задачи требуется синтезировать алгоритм под особенности аппаратной архитектуры NITTA.

Процесс синтеза в NITTA представляет собой процесс применения операций над моделью, до тех пор, пока они не закончатся. Результатом процесса синтеза будет граф пересылок данных, содержащий в себе

информацию о вычислительных блоках и пересылках данных между ними в каждый момент времени. Иными словами, происходит преобразование согласно алгоритму разрозненных операций и вычислительных в единую структуру вычислителя.

Процесс синтеза состоит из нескольких классов операций:

- связывание функций вычислительного блока;
- пересылка данных между вычислительными блоками;
- рефакторинг.

Операция связывания принимает алгоритм, который представляет собой набор функций, которые в процессе синтеза соединяются с вычислительными блоками. Несколько функций могут быть связаны с одним вычислительным блоком. Микроархитектура вычислительной системы может содержать несколько вычислительных блоков одного типа, тогда в процессе синтеза процесс связывание, используя различные метрики, выбирает наилучший способ нахождения соответствия функции и вычислительного блока.

Операция пересылки данных на основе информации о том, когда какие вычислительные блоки должны отправить данные, создаёт некоторый маршрут пересылок в каждый момент времени. С помощью данной операции формируется вычислительный процесс, за счёт обеспечения взаимодействия между вычислительными блоками.

Класс операций рефакторинга содержит в себе несколько операций:

- пересылка данных между циклами работы программы;

- добавление регистра для сохранения промежуточных результатов.

Специализированные вычислители работают циклически, поэтому в САПР NITTA обязательным условием программы является создание функции с бесконечной рекурсией. Для передачи данных между циклами используется функция *Loop*, которая в процессе синтеза с помощью операции рефакторинга разделяется на две операции, *LoopIn* и *LoopOut*, используемые соответственно для передачи входных данных в начале цикла и выходных данных в конце цикла. Передавая переменные в вычислительный блок памяти (*Fram)*.

Если в алгоритме используется одна переменная в нескольких местах, то происходит взаимная блокировка, для предотвращения которой добавляется дополнительный регистр для хранения промежуточного результата.

Рефакторинг и оптимизация это два схожих понятия, смысл которых — изменение алгоритма для получения какой-то выгоды, но при этом без изменения функциональности. Рефакторингом чаще всего называют операции, которые изменяют алгоритм внешне, часто это может быть переименование переменных, объединение повторяющегося кода в функции, правильные отступы, и другие изменения, которые слабо влияют, или почти не влияют на производительность системы, при этом улучшая читаемость и понятность алгоритма, что на самом деле очень важно, особенно если алгоритм может быть подвергнут изменению в будущем. Оптимизация, наоборот, чаще всего только усложняет понимание работы алгоритма, но при этом улучшая какие-либо параметры работы программы, например можно оптимизировать скорость работы программы, т. е. заставить её работать быстрее, или использовать меньше оперативной памяти.

Таким образом, операции оптимизации было решено добавить в существующий набор операций рефакторинга на основе похожего принципа работы.

2 МЕХАНИЗМЫ ОПТИМИЗАЦИИ ЦЕЛЕВОГО АЛГОРИТМА

Для оптимизации целевого алгоритма следует увеличить эффективность работы целевого вычислителя, где критерий эффективности — время выполнения цикла программы, и чем оно меньше, тем лучше.

На уровне процесса синтеза, алгоритм принятия решений не может самостоятельно что-либо оптимизировать, для этого ему нужны опции из которых можно выбрать один из вариантов.

Для появления таких опций и возможностей их принимать были разработаны дополнительные механизмы оптимизации целевого алгоритма.

2.1 Оптимизация аккумулятора

Операция сложения или вычитания обычно принимает на вход два числа и возвращает результат. Для вычисления нескольких операций сложения или вычитания, например: $res = a + b - c$, на уровне формирования графа пересылок данных данное выражение разбивается на два: $tmp1 = a + b; res = tmp1 + c$. Однако данную операцию можно оптимизировать, если рассмотреть аппаратную реализацию работы сумматора (Рисунок 3).

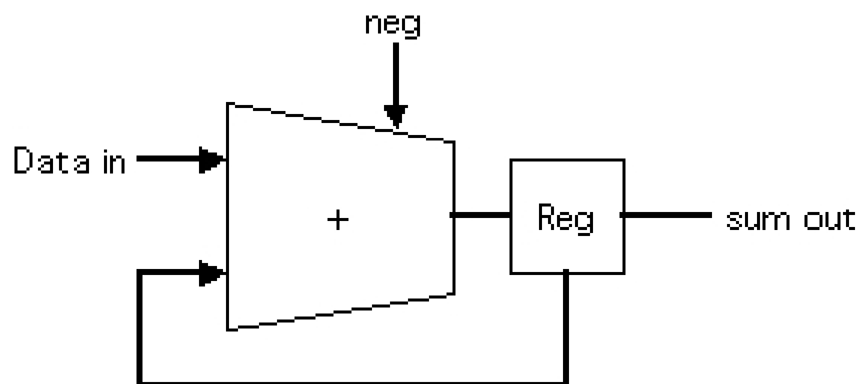


Рисунок 3 – Аппаратный блок сумматора

Вычислительный блок сумматора уже содержит в себе регистр, используемый для суммирования, поэтому в процессе вычисления делается лишняя работа работа: добавление промежуточного результата в блок памяти (Fram), для чего нужно использовать дополнительные пересылки данных, которые в архитектуре вычислительной системы NITTA требуют использования большого количества ресурсов(память и время).

Для реализации данной задачи требовалось создать новую функцию сумматора, которая может на вход принимать несколько значений со знаком и выдавать результат суммирования, позволяя тем самым избавиться от лишних пересылок данных которые присутствуют на текущей реализации процесса суммирования (Рисунок 3).

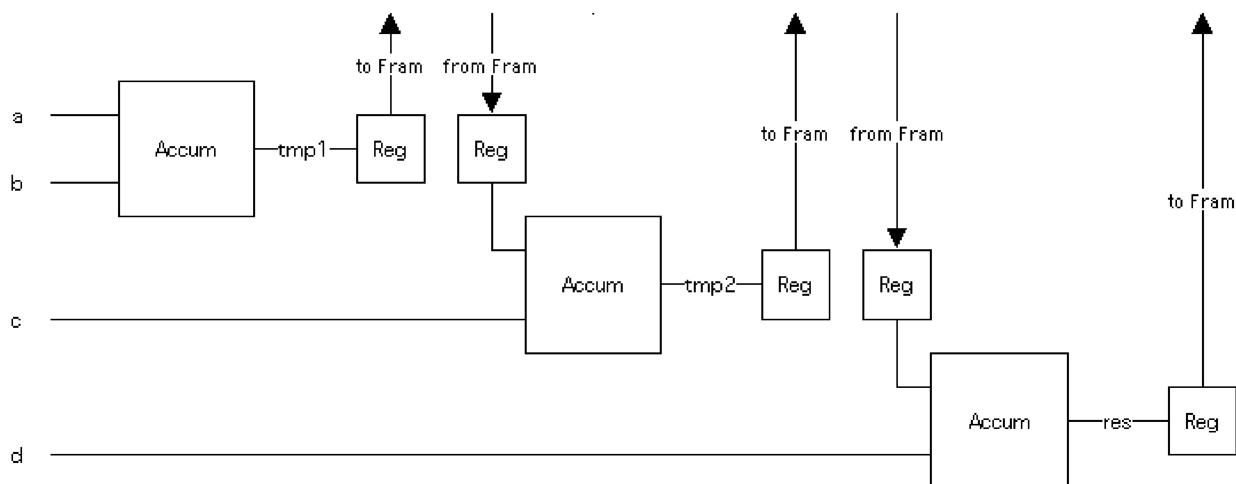


Рисунок 4 – Организация процесса четырёх элементов

На рисунке 4 показан процесс вычисления выражения $res = a + b + c + d$, который разделяется на 3 последовательные операции суммирования: $tmp1 = a + b$, $tmp2 = tmp1 + c$, $res = tmp\ 2 + d$. Такое разделение позволяет построить гибкую системы вычисления, когда данные операции суммирования могут идти в разное время, не обязательно последовательно и на разных вычислительных блоках сумматора, что в теории может дать лучший показатель производительности в сравнении с одним большим последовательным процессом суммирования (Рисунок 5).

Однако, при использовании микроархитектуры с одним вычислительным блоком сумматора оптимальным будет вариант с суммированием нескольких значений за одну атомарную операцию. Чтобы гибко выбирать наиболее подходящий вариант суммирования, была добавлена ещё одна функция для вычислительного блока аккумулятора, которая может выполнять операцию суммирования без пересылок данных. Таким образом, в процессе синтеза можно гибко выбирать: оставить несколько операций суммирования или оптимизировать и преобразовать в одну большую операцию.

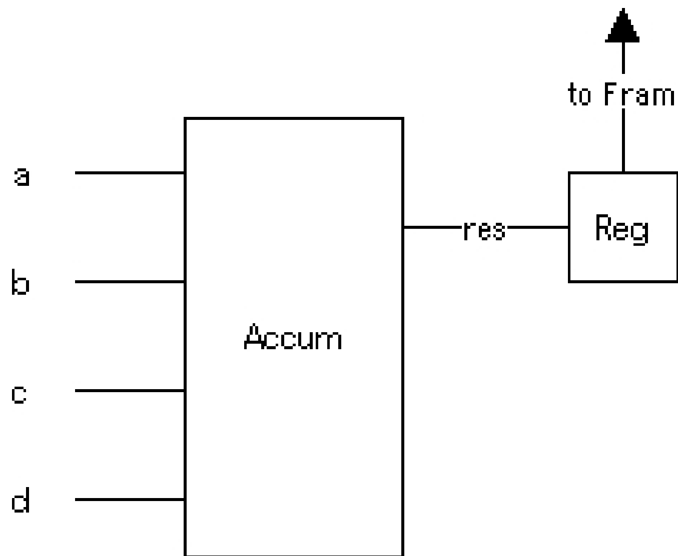


Рисунок 5 – Организация процесса суммирования 3 и более элементов без пересылки значений временных переменных

Был добавлен тип функции сумматора *Accum*, который содержит в себе список входных и выходных переменных. Входные переменные (*Push*) – содержат в себе идентификатор переменной (*I String*) и арифметический знак (*Sign*). Выходные переменные (*Pull*) – содержат в себе только идентификатор переменной (*O [String]*).

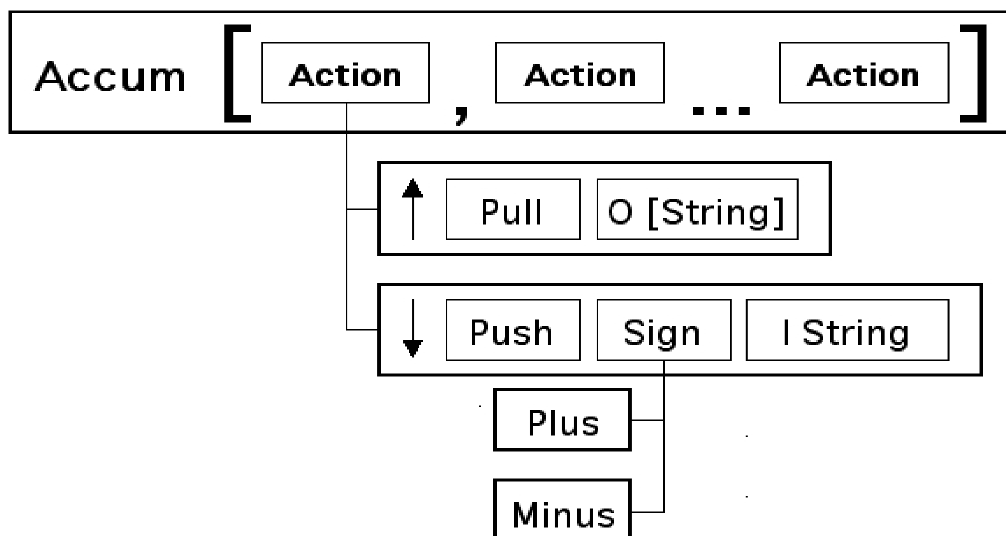


Рисунок 6 – Тип функции сумматора, позволяющий производить вычисления без дополнительных пересылок данных

Для обозначения идентификаторов входных (*I, от англ. Input*) и выходных (*O, от англ. Output)* переменных используются разные идентификаторы, потому что такие обозначения принято использовать в данном проекте, чтобы различать входные и выходные переменные и использовать дополнительные функции, которые работают за счёт разделения данных типов[7].

После синтаксического анализатора и преобразования абстрактного синтаксического дерева, в графе пересылок данных не существует такой операции, как суммирование. Используются только операции сложения (*Add*) и вычитания (*Sub*). Поэтому требовалось реализовать дополнительную операцию в процессе синтеза из класса операций *Refactor,* которая объединяет несколько операций сложения и вычитания на основе информации о том, что их идентификаторы входных и выходных переменных пересекаются, такие операции добавляются в отдельные контейнеры.

Каждый контейнер с операциями сложения и вычитания преобразовывается в одну операцию суммирования и добавляется в процесс синтеза как оптимизация сумматора (*accumOptimization*). После чего в процессе синтеза появляется дополнительная опция в классе операций *Refactor*, после применения которой, функции сложения и вычитания из контейнера заменяются на функцию аккумулятора.

При использовании такой оптимизации можно добиться уменьшения количества пересылок данных между вычислительными блоками в *(n-2)*2* раз, где *n* – количество слагаемых для вычисления, что наглядно видно на рисунке 7.

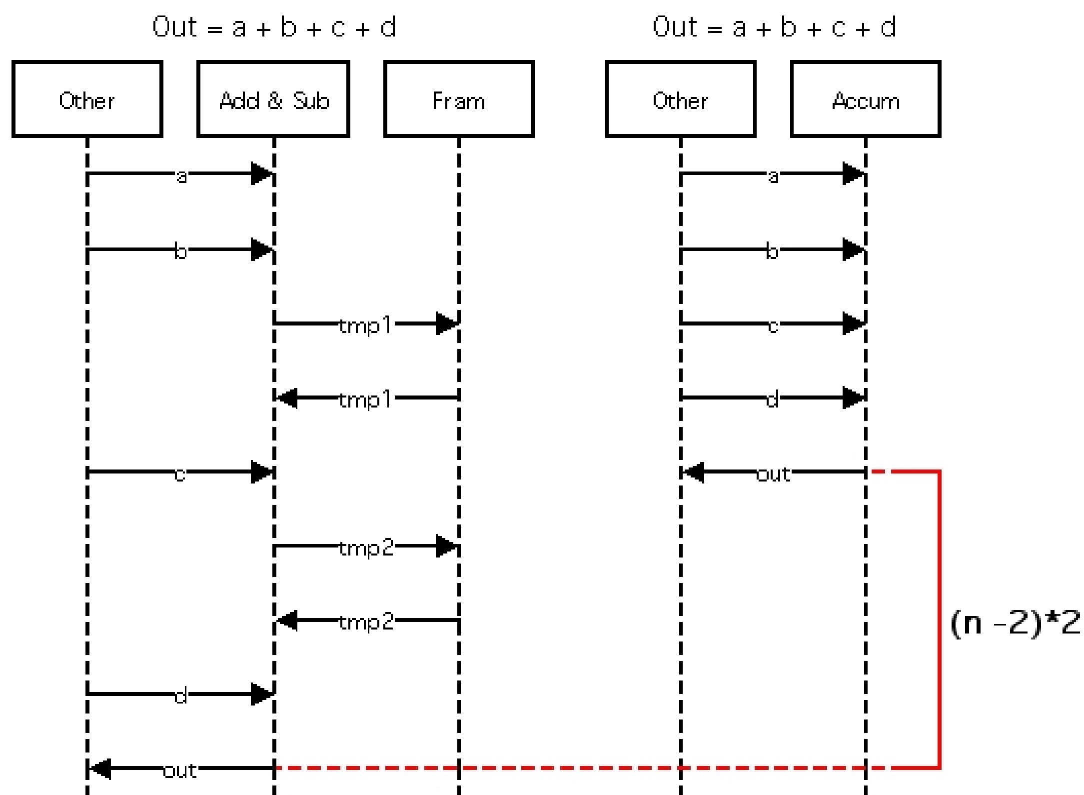


Рисунок 7 – Сравнение количества пересылок с использованием функций сложения и вычитания (*Add & Sub*), и с использованием функции сумматора (*Accum*)

Пример работы механизма оптимизации аккумулятора в процессе синтеза показан в приложении А.

2.2 Свёртка констант

Для уменьшения количества операций во время работы целевого вычислителя, САПР можно вычислять все операции, с константами во время процесса синтеза. Тем самым целевая система будет работать быстрее, но при этом можно будет производить некоторые операции с константами в коде входного алгоритма.

Для реализации свёртки констант требуется выполнить следующие действия:

- вычислить операции, где входные значения являются константами;
- запустить процесс моделирования работы системы в ограниченной среде;
- получить результат моделирования в виде константного значения;
- заменить вычисленные операции на константное значение.

Рисунок 8 – Схема получения опций свёртки констант в процессе синтеза

Функция может быть вычислена с помощью свёртки констант, если её входные значения (*inputs*) являются константами, поэтому все функции фильтруются и выбираются только те, которые удовлетворяют описанным выше требованиям. После чего данные функции разделяются на контейнеры, где каждый контейнер содержит в себе функцию с идентификатором переменных, а также сами значения константных переменных. Для каждой функции в системе автоматического проектирования реализована операция моделирования (*simulate*) [5, 6], которая принимает на вход функцию, которую требуется вычислить, а также контекст, где хранятся значения идентификаторов входных переменных, с помощью чего можно смоделировать работу данной функции во время выполнения САПР, без использования средств моделирования для специализированных вычислителей.

Операция моделирования запускается и выдаёт итоговый результат в виде новой константы, после чего мы можем заменить функцию в графе пересылок данных этой константой. Запускается проверка использования константных значений в графе пересылок данных, потому что при свёртке функции в константу, некоторые константные значения могут больше не требоваться для вычислительного процесса, поэтому следует их удалить, чтобы не занимать лишние объёмы в памяти. Но при этом нельзя удалить все константы, которые используются в данной функции, потому что несколько функций могут ссылаться на одну константу.

Пример работы механизма свёртки констант в процессе синтеза подробно продемонстрирован в приложении Б.

Система автоматического проектирования NITTA состоит из нескольких уровней представления и трансформации данных между этими уровнями, поэтому требовалось выявить наилучшее место для интеграции

такой оптимизации. Проанализировав возможные варианты: во время синтаксического анализа, при инициализации графа пересылок данных, в процессе синтеза; было принято решение интегрировать данную оптимизацию в процесс синтеза, потому что тогда можно гибко (решение о принятии данной оптимизации может как приниматься, так и не приниматься) применять данную оптимизацию с помощью уже имеющихся средств и критериев для оценки полезности данных изменений.

2.3 Вывод

Таким образом, данные механизмы оптимизации целевого вычислителя позволяют существенным образом повысить скорость работы целевого вычислителя. Конечно, данные оптимизации требуют дополнительных действий в процессе синтеза, однако получаемые улучшения перекрывают затраты требуемые на применение данных механизмов. Также был разработан и интегрирован подход для добавления новых методов оптимизации целевого вычислителя в САПР спецвычислителей, поэтому теперь существует понимание и примеры удачных интеграций механизмов оптимизации на основе которых можно добавлять новые механизмы для оптимизации.

3 МЕХАНИЗМЫ ОПТИМИЗАЦИИ ПРОЦЕССА СИНТЕЗА

Процесс синтеза — основная часть САПР NITTA, в которой происходит построение специализированного вычислителя под входной алгоритм. В процессе синтеза, последовательно принимаются решения о выборе дальнейшего пути развития модели вычислительной системы и основная проблема в данном случае, что с увеличением сложности входного алгоритма количество возможных вариантов решений в процессе синтеза

многократно растёт, поэтому проблема оптимизации процесса синтеза не менее актуальная, чем проблема оптимизации целевого вычислителя.

3.1 Эффективное распределение функций прикладного алгоритма между вычислительными блоками

Одной из операций в процессе синтеза является связывание (*Binding*), которая используется для соединения вычислительных блоков с функциями из алгоритма. Например, функция сложения (*Add*) должна быть связана с вычислительным блоком аккумулятора (*Accum)*, функция определяющая значение константы (*Const*) привязывается вычислительному блоку памяти (*Fram)*.

Такое связывание происходит на основе типа данных функций. Каждый вычислительный блок реализует функцию *tryBind*, которая проверяет тип данных входящей функции и если он удовлетворяет этому вычислительному блоку, то функция привязывается к вычислительному блоку.

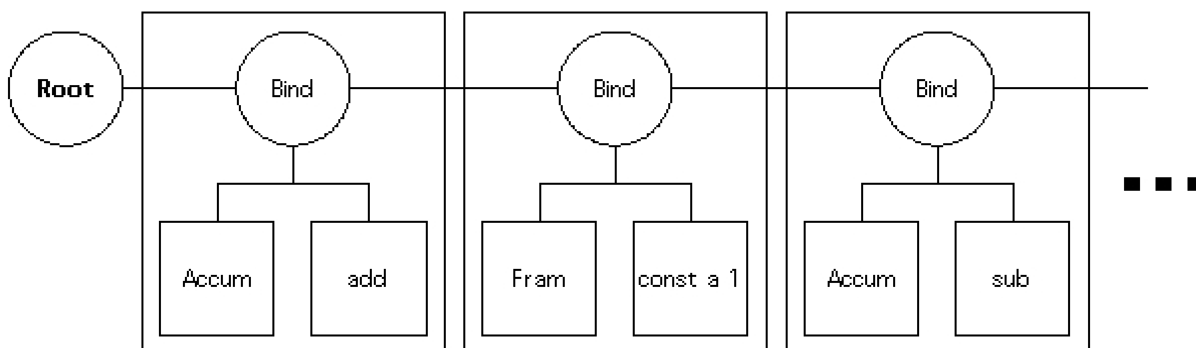


Рисунок 9 – Последовательное связывание функций с вычислительными блоками в процессе синтеза

Все эти операции происходят последовательно в процессе синтеза, что сильно увеличивает количество решений, которые может принять система автоматического проектирования. Такой подход позволяет гибко принимать

решения в процессе синтеза, особенно когда есть несколько одинаковых вычислительных блоков или вычислительных сетей. Но чаще всего эта операция не требует обеспечения такой степени гибкости, при этом сильно замедляя процесс синтеза. Поэтому было принято решение добавить дополнительный вид процесса связывания (групповое связывание), когда связываются все оставшиеся в данный момент функции со всеми вычислительными блоками.

Групповое связывание может использовать разную логику работы в зависимости от вычислительных блоков и функций, которые требуется связать:

- безальтернативное связывание – все вычислительные блоки уникальны,
- полное связывание – существует несколько вычислительных блоков одного типа,
- вычислимое связывание – связываются только те функции с вычислительными блоками, которые можно вычислить прямо сейчас.

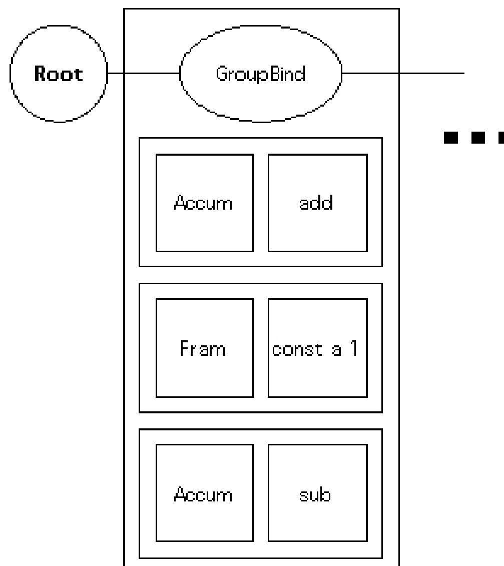


Рисунок 10 – Групповое связывание функций с вычислительными блоками

Безальтернативное связывание является самым простым в описании и реализации, потому что все вычислительные блоки уникальны, а значит в процессе синтеза не будет сложности в принятии верного решения.

Полное связывание предполагает предоставление всех возможных вариантов для связывания всех вычислительных функций со всеми удовлетворяющими по типу вычислительными блоками, тем самым в процессе синтеза можно выбрать наиболее подходящее распределение.

Вычислимое связывание выделяет операции, которые можно рассчитать во время процесса синтеза, например инициализировать значение констант в вычислительном блоке оперативной памяти, и привязывает их к соответствующим по типам вычислительным блокам. В данном случае также требуется принимать решение о выборе конкретной опции, потому что может быть несколько вычислительных блоков, которые соответствуют одной функции.

Для реализации группового связывания был расширен алгебраический тип данных [7] связывания (*Bind*), добавлением ещё одного конструктора с

групповым связыванием (GroupBinding), которые принимает параметр типа группового связывания и список операций связывания. Таким образом получается достичь результата без сложной модификации системы, повторно используя операции связывания для группового связывания.

Групповое связывание может применятся на любом этапе, когда есть функции, которые не привязаны к вычислительным блокам, так например, в процессе синтеза может быть привязано несколько функций к определенным вычислительным блокам, после чего применяется групповое связывание и связываются все остальные функции с вычислительными блоками. Таким образом можно гибко использовать групповое связывание, применяя его там где это требуется и будет эффективно, но при этом позволяя сделать все привязяки последовательно за несколько операций процесса синтеза.

3.2 Формирование наборов данных для машинного обучения

Сейчас процесс синтеза происходит автоматически на основе собираемых метрик во время процесса синтеза, которые влияют на определённые правила, из которых формируется вес операции, после чего операции в процессе синтеза применяются в порядке, где операция с наибольшим весом будет иметь преимущество над операцией с меньшим весом. Данные правила реализуются человеком на основе опыта проектирования подобных систем, поэтому появилась идея добавить возможность использовать методы машинного обучения для теоретического улучшения процесса синтеза за счёт использование решений, которые человек не стал бы пробовать, но при этом они могли бы быть эффективными.

Также, с помощью методов машинного обучения САПР может работать эффективнее в процессе синтеза, потому что будет меньше обходиться ненужных узлов в дереве синтеза, за счёт информации,

содержащейся в весах нейронной сети на основе обучения на «хороших» и «плохих» данных.

В данном случае сразу возникает сложность оценивания качества работы процесса синтеза, для понимания возможных улучшений, после чего сравнив результаты процесса синтеза с использованием методов машинного обучения и обычного автоматизированного процесса синтеза.

Поэтому были выделены следующие метрики для анализа работы процесса синтеза:

- количество известных узлов в процессе синтеза;
- количество известных успешно синтезированных узлов;
- количество не изученных поддеревьев;
- количество узлов, в которых процесс синтеза не был завершён;
- зависимость числа успешно синтезированных узлов от длительности работы целевой вычислительной системы;
- зависимость числа успешно синтезированных узлов от количества шагов в процессе синтеза.

В процессе синтеза строится дерево синтеза, где каждый узел – применение одной из возможных операций из множества операций доступных в процессе синтеза.

Узел считается известным, если мы можем в него пойти в процессе синтеза, т.е. существует такая опция и в процессе синтеза был отправлен запрос на получение информации об этой опции. Если запрос на получение опции не отправлен, и данные не получены, то в процессе синтеза неизвестно есть есть ли какие-то операции, которые можно было бы применить.

Узел считается успешно синтезированным, если он является листом в дереве синтеза (последним элементом), все функции привязаны к вычислительным блокам, все переменные в графе пересылок данных участвуют в пересылках между вычислительными блоками.

Количество неизученных поддеревьев – количество узлов, в которые не были рассчитаны.

Количество узлов в которых процесс синтеза не завершён – узлы, которые являются листьями, но при этом не выполняется одно из условий для успешного синтеза.

Последние два пункта представляются в формате списка кортежей, где первый элемент кортежа, это длительность работы алгоритма или количество шагов в процессе синтеза, а второй элемент – количество успешно синтезированных узлов, которые соответствуют первому параметру. После чего из этих данных можно построить график зависимости между данными свойствами.

САПР NITTA использует подход *RESTful* [8] *API* для обмена серверной части системы с клиентской частью. Обмен и изменение данных происходит посредством HTTP запросов к серверу. Поэтому для формирования метрик о процессе синтеза был добавлен ресурс *TreeInfo* представляющий по GET всю информацию в формате JSON со структурой представленной на листинге 1.

Листинг 1 – Структура данных содержащая метрики для оценивая работы процесса синтеза

```
TreeInfo
  { nodes :: Int
  , success :: Int
  , failed :: Int
  , notProcessed :: Int
  , durationSuccess :: [(Int, Int)]
```

```
, stepsSuccess :: [(Int, Int)]
}
```

3.3 Вывод

Для реализации механизмов оптимизации процесса синтеза, пришлось разобраться в деталях работы процесса синтеза, выяснить как принимаются решения, из каких параметров формируется вес опции в процессе синтеза. Произведённый анализ позволил интегрировать в САПР спецвычислителей дополнительные механизмы, которые позволяют как оптимизировать работу процесса синтеза по времени, затраченному на синтез целевого алгоритма, так и дать оценку алгоритму принятия решений в данном процессе синтеза, чтобы выявить недостатки текущего алгоритма, или использовать внешние средства для обработки информации о принятии решений в процессе синтеза.

4 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

После практической реализации соответствующих механизмов для оптимизации внутри САПР, требовалось провести испытания для проверки теоретических расчётов, работоспособности итогового решения, отладки, поиска недостатков для дальнейшего развития и модернизации.

4.1 Анализ эффективности целевого вычислителя при наличии механизма оптимизации сумматора в процессе синтеза

Проверка производится на основе тестовой программы (Листинг 2), где основная операция – сложение 3 элементов.

Для анализа эффективности целевого вычислителя при наличии механизма оптимизации сумматора в процессе синтеза можно использовать различные метрики, например количество пересылок, пример с которыми был показан на рисунке 7.

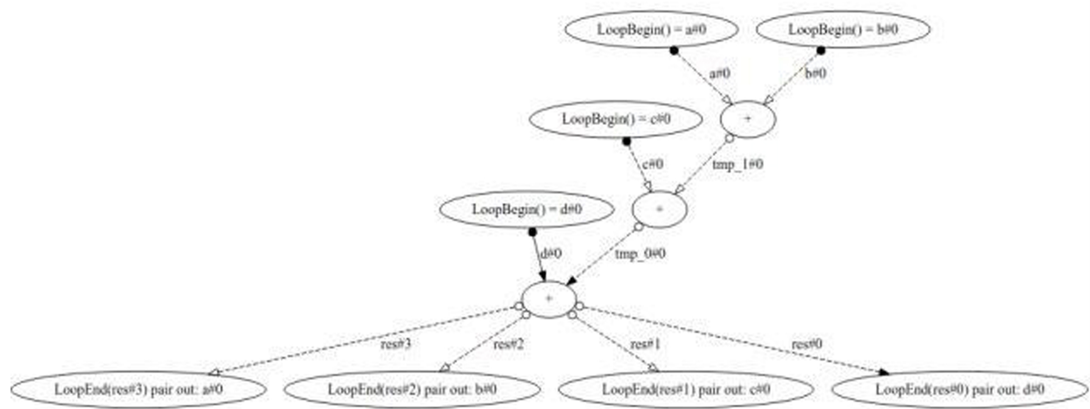


Рисунок 11 – Граф пересылки данных без оптимизации сумматора

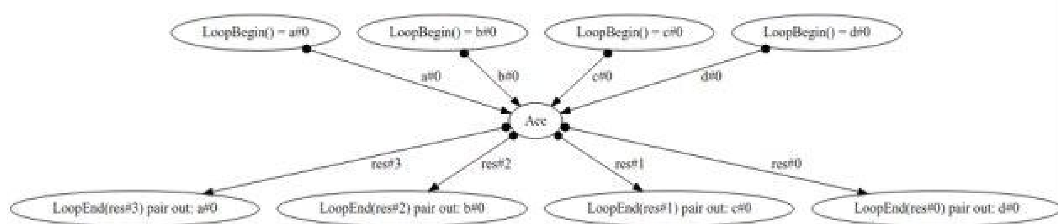


Рисунок 12 – Граф пересылки данных после оптимизации сумматора

Также можно сравнить граф пересылки данных до применения оптимизации сумматора (Рисунок 11) и после применения оптимизации сумматора (Рисунок 12). Можно заметить, что после оптимизации сумматора три последовательных операции сложения трансформируются в одну операцию суммирования, уменьшая количество пересылок данных.

Дополнительно, можно сравнить волновые диаграммы показанные на рисунках 13, 14; сформированные на основе Verilog кода, позволяющие посмотреть на работу вычислителя на уровне тактов программы. Разница между программой без оптимизации аккумулятора (12 тактов) и с оптимизацией аккумулятора (6 тактов) отличается на 6 тактов, что ровно вполовину меньше, чем без использования оптимизации сумматора. Разница будет увеличиваться с количеством складываемых элементов, потому что без оптимизации сумматора каждая сумма будет занимать 6 тактов, при том, что с применением оптимизации сумматора добавление ещё одного элемента для суммирования потребует лишь 2 такта на каждый дополнительный элемент.

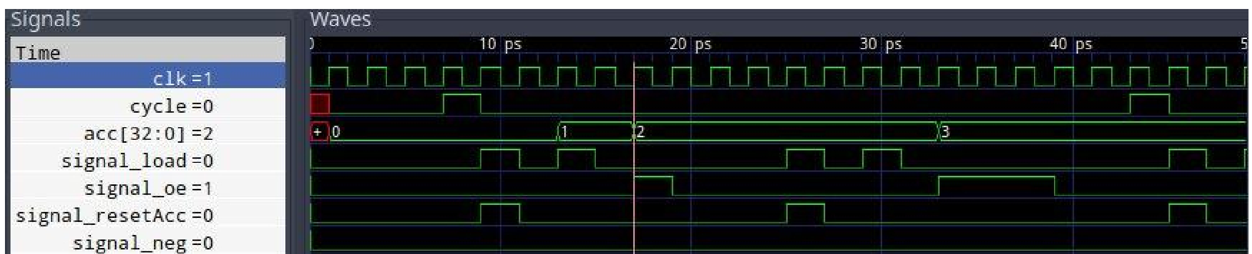


Рисунок 13 – Волновая диаграмма вычислительного блока аккумулятора после применения оптимизации

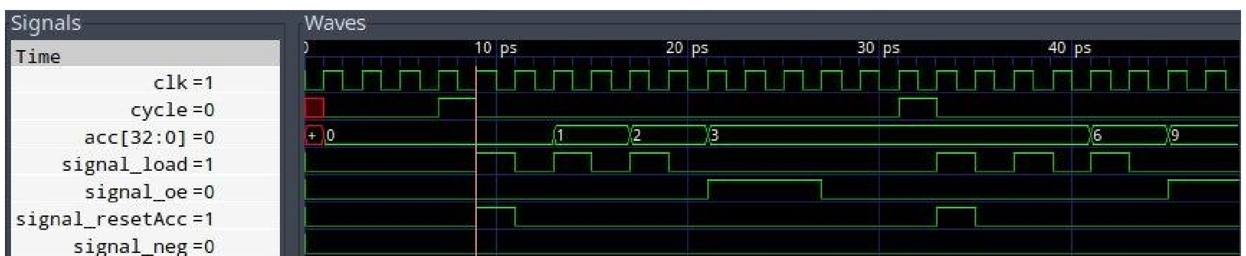


Рисунок 14 – Волновая диаграмма вычислительного блока аккумулятора после применения оптимизации

Таким образом, данная опция в процессе синтеза будет всегда иметь наилучшие результаты с учётом использования одного блока аккумулятора. Однако чтобы добиться максимальной производительности суммирования большого количества элементов стоит использовать много вычислительных

блоков сумматора, потому что тогда накладные расходы на пересылку данных между вычислительными блоками будут много меньше, чем процесс последовательного суммирования большого числа элементов.

Листинг 2 – Программа на языке программирования Lua для оценивания работы оптимизации сумматора

```
function sum(a, b, c)
    local d = a + b + c
    sum(d, d, d)
end
sum(1, 1, 1)
```

4.2 Анализ эффективности целевого вычислителя при наличии механизма свёртки констант в процессе синтеза

При использовании механизма свёртки констант в процессе синтеза, все функции в которых входные переменные это константы, заменяются на вычисленные заранее константные значения, тем самым конечный вычислитель работает ровно на столько тактов быстрее, сколько потребовалось для вычисления данных операций.

Для проведения тестирования эффективности использовалась тестовая программа содержащая несколько операций сложения, представленная на листинге 3.

Листинг 3 – Программа на языке программирования Lua для оценивания работы свертки констант

```
function constantFolding(i)
    local v = 1 + 2 + 3
    local res = i + v + 3
    constantFolding(res)
end
constantFolding(0)
```

Наиболее наглядно результат применения свёртки констант можно рассмотреть в клиентской части системы автоматического проектирования специализированных вычислителей. На рисунке 15 продемонстрирован граф

пересылки данных сформированный по алгоритму из листинга 2, который состоит из суммирования 5 элементов.

После применения оптимизаций граф пересылок данных состоит из одной операции сложения (Рисунок 16). Таким образом, если пересчитать на такты специализированного вычислителя, то в данном конкретном случае разница будет составлять 6 тактов, если применять оптимизацию аккумулятора и много больше, если суммировать переменные попарно с пересылкой временных переменных между вычислительными блоками.

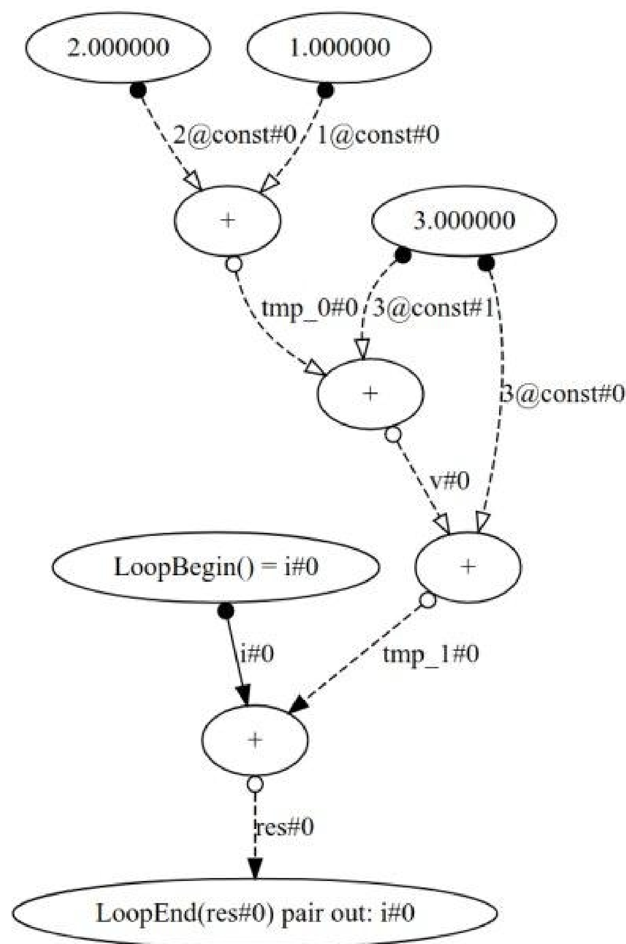


Рисунок 15 – Граф пересылок данных без применения свёртки констант

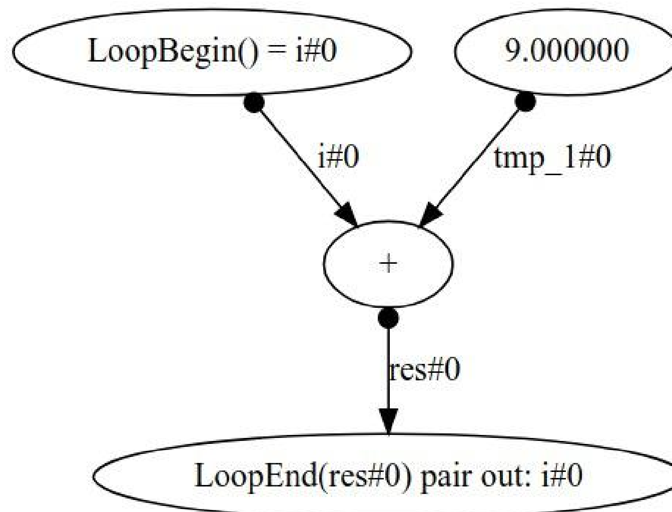


Рисунок 16 – Граф пересылок данных после применения свёртки констант

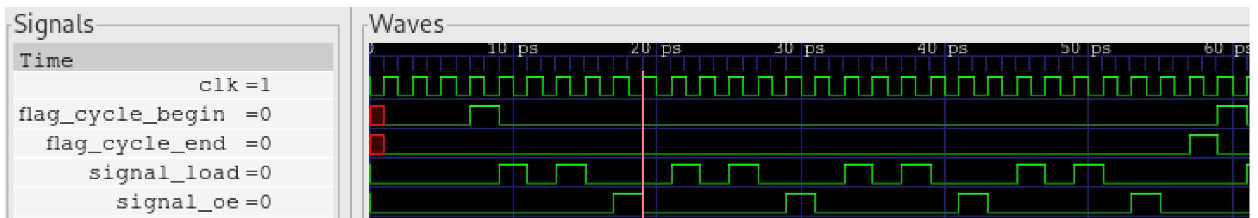


Рисунок 17 – Волновая диаграмма до свёртки констант

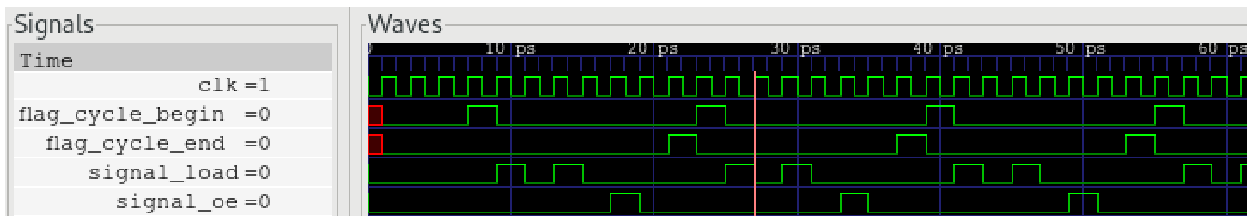


Рисунок 18 – Волновая диаграмма после свёртки констант

Для более подробного изучения можно рассмотреть волновые диаграммы на рисунках 17, 18. В данном случае лучше всего обратить внимание на строчку с параметром `flag_cycle_begin`, потому что если без свёртки констант один цикл программы длится 24 такта, то после применения свёртки констант длительность одного цикла составляет 7 тактов.

Отдельно стоит отметить, что свёртка констант может быть применена с любой арифметической операцией доступной в САПР.

4.3 Анализ результатов группового связывания

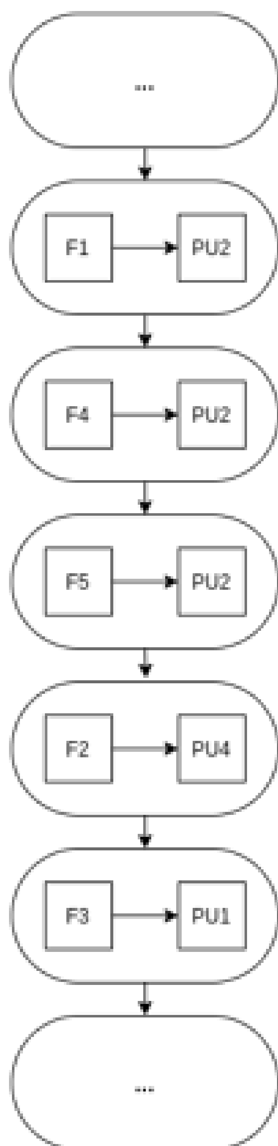
Групповое связывание добавляет новую функциональность в процесс синтеза, что позволяет системе автоматического проектирования гибко подбирать решения в процессе синтеза, изменяя свои параметры в зависимости от входного алгоритма и микроархитектуры, при этом не используя обход всего дерева решений процесса синтеза, а основываясь лишь на самых подходящих в данный момент решениях.

Групповое связывание позволяет ускорить процесс синтеза, в особенности, когда используется микроархитектура с уникальными вычислительными блоками, потому что тогда можно применить безальтернативное связывание, которое привяжет все функции алгоритма к соответствующим по типу вычислительным блокам.

Особенно актуально данное улучшение будет с добавлением множественных сетей в САПР. Тогда процесс синтеза станет ещё сложнее, потому что потребуется рассчитывать взаимодействие между вычислительными блоками как через сеть, так и данные сети между собой, в таком случае станет ещё больше возможностей для синтеза и небольшие операции последовательной привязки вычислительных блоков ощутимо замедлят работу системы автоматического проектирования..

Граф пересылок данных со стандартным методом привязывания функций к вычислительным блокам

Граф пересылок данных с использованием методов одновременного привязывания функций к вычислительным блокам



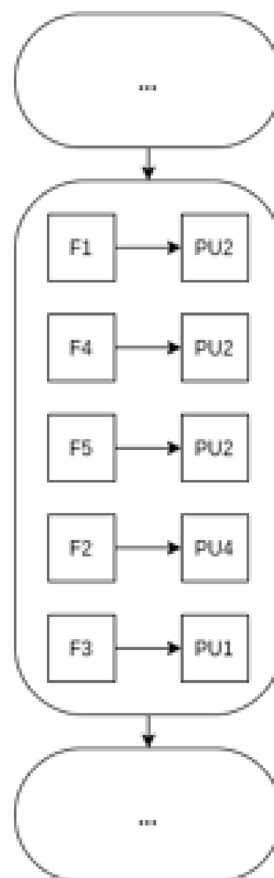


Рисунок 19 – Сравнение последовательных операций связывания и групповой операции связывания в процессе синтеза

4.4 Результаты сбора данных для оценки результата работы процесса синтеза

Для оценки работы процесса синтеза и различных методов его обхода были применены собранные метрики описанные в части 3.2. На рисунке 20 можно увидеть реальный пример данных, полученных после процесса синтеза. Особенно интересны поля с зависимостями успешно синтезированных узлов от параметров длительности работы целевого вычислителя и длительности связывания. Потому что, например, после их анализа можно изменить алгоритм принятия решений в процессе синтеза для оптимизации его работы по времени, которое зависит от количество перебранных узлов в графе синтеза.


```
{
    "notProcessed": 853,
    "durationSuccess": {
        "6": 5,
        "9": 4,
        "10": 5,
        "11": 2,
        "15": 1,
        "25": 1
    },
    "success": 18,
    "stepsSuccess": {
        "10": 5,
        "13": 2,
        "15": 9,
        "19": 1,
        "27": 1
    },
    "nodes": 1112,
    "failed": 10
}
```


Рисунок 20 – Пример результата полученных данных после применения процесса синтеза

Стоит отметить, что в дальнейшем данные поля будут модифицироваться, а также добавятся новые метрики, и в данной работе

было лишь протестировано, что данные метрики являются полезными, их можно собирать и анализировать.

В дальнейшем данные или подобные метрики можно применять для анализа алгоритмов синтеза и степени его успешности, например, для добавления возможности изменять микроархитектуру в процессе синтеза, а данные метрики будут показывать корреляцию между выбранной микроархитектурой, методом процесса синтеза и степенью успешности результата, где успешность может являться разными параметрами: время работы целевого алгоритма, площадь целевого спецвычислителя (количество вычислительных блоков определённого размера), сложность процесса синтеза, скорость работы процесса синтеза и другие.

Одним из недостатков данной реализации сбора метрик, является скорость работы. Потому что для обхода дерева используются однопоточные ленивые алгоритмы, поэтому анализ дерева решений занимает продолжительное время (в среднем запрос длится около 3 секунд). Для предотвращения данной проблемы в будущем, алгоритм будет модифицирован за счёт добавления строгих вычислений, а также распараллеливания сбора данных. Что позволит увеличить скорость анализа дерева, что может быть критично для сбора большого количества данных для машинного обучения.

4.5 Вывод

Анализ результатов проведённых оптимизаций позволяет удостовериться в правильности предположений, сделанных на стадии анализа критических для оптимизации мест. После реализации и интеграции данных методов, в САПР были добавлены новые опции процесса синтеза, применение которых позволяет ускорить работу целевого вычислителя или скорость принятия решений в процессе синтеза.

ЗАКЛЮЧЕНИЕ

В результате выпускной квалификационной работы была достигнута поставленная цель и выполнены все задачи.

Система автоматического проектирования специализированных вычислителей была дополнена новыми опциями процесса синтеза (свёртка констант, оптимизация сумматора), которые позволили оптимизировать работу целевого вычислителя, где критерий эффективности – длительность работы целевого вычислителя.

Также удалось интегрировать новый механизм связывания функций прикладного алгоритма с вычислительными блоками, что в свою очередь позволило оптимизировать работу процесса синтеза, где критерий эффективности – длительность процесса синтеза.

В дополнении была реализована функциональность в виде сбора данных для оценки результата работы процесса синтеза, которая будет актуальна для дальнейшего развития механизмов процесса синтеза и улучшений, связанных с модификацией алгоритма принятия решений в процессе синтеза, где эти данные будут являться критерием для оценки улучшений или ухудшений алгоритма принятия решений в процессе синтеза, позволяя дать измеримую оценку происходящего.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Квиттнер П. Задачи программы вычисления результатов/Пер. с англ. под. ред. В. В. Мартынюка. – 1980.

2. Wells M. B. Donald E. Knuth, The Art of Computer Programming, Volume 1. Fundamental Algorithms and Volume 2. Seminumerical Algorithms //Bulletin of the American Mathematical Society. – 1973. – Т. 79. – №. 3. – С. 501-509.

3. Пенской А. В. Разработка и исследование архитектурных стилей проектирования уровневой организации встроенных систем : дис. – С.-Петерб. нац. исслед. ун-т информац. технологий, механики и оптики, 2016.

4. Пенской А. В. и др. Система высокоуровневого синтеза на основе гибридной реконфигурируемой микроархитектуры //Научно-технический вестник информационных технологий, механики и оптики. – 2019. – Т. 19. – №. 2.

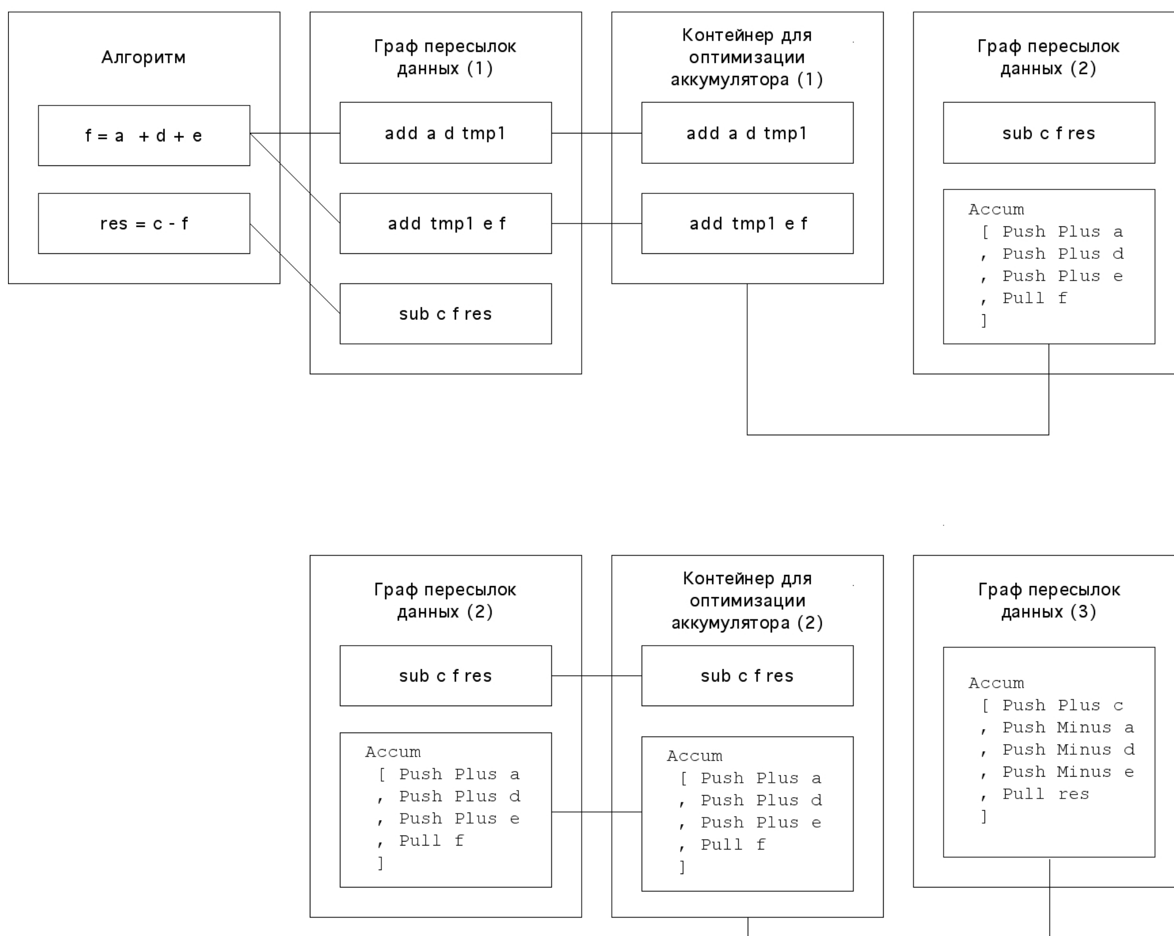
5. Анощенков Д. И. Верификация систем высокоуровневого синтеза аппаратных вычислителей//Альманах научных работ молодых ученых Университета ИТМО. – 2018. – С. 241-244.

6. Prohorov D., Penskoi A. Verification of the CAD System for an Application-Specific Processor by Property-Based Testing //2020 9th Mediterranean Conference on Embedded Computing (MECO). – IEEE, 2020. – С. 1-4.

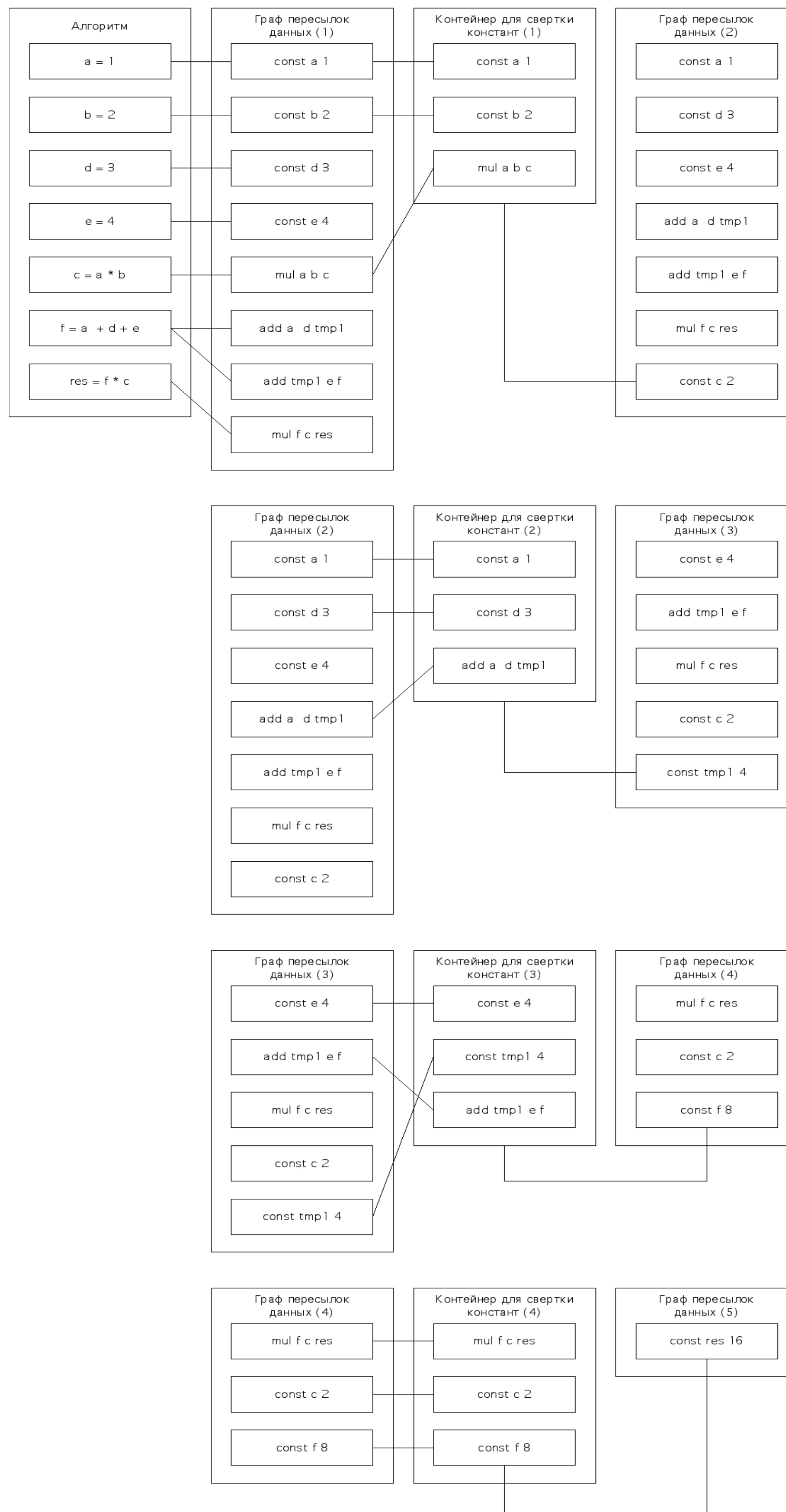
7. Jones S. P. (ed.). Haskell 98 language and libraries: the revised report. – Cambridge University Press, 2003.

8. Rodriguez A. Restful web services: The basics //IBM developerWorks. – 2008. – Т. 33. – С. 18.

ПРИЛОЖЕНИЕ А (ПРИМЕНЕНИЕ ОПТИМИЗАЦИИ АККУМУЛЯТОРА В ПРОЦЕССЕ СИНТЕЗА)



ПРИЛОЖЕНИЕ Б (ПРИМЕНИНИЕ СВЕРТКИ КОНСТАНТ В ПРОЦЕССЕ СИНТЕЗА)



ПРИЛОЖЕНИЕ В (ЛИСТИНГ ВЫЧИСЛИТЕЛЬНОГО БЛОКА СУММАТОРА)

```
{- |
Module      : NITTA.Model.ProcessorUnits.Accum
Description : Accumulator processor unit implementation
Copyright   : (c) Aleksandr Penskoi, 2019
License     : BSD3
Maintainer  : aleksandr.penskoi@gmail.com
Stability   : experimental
-}
module NITTA.Model.ProcessorUnits.Accum (
    Accum,
    Ports (..),
    IOPorts (..),
) where

import Control.Monad (when)
import Data.Default
import Data.List (find, partition, (\\))
import Data.Maybe (fromMaybe)
import Data.Set (elems, fromList, member)
import Data.String.Interpolate
import Data.String.ToString
import qualified Data.Text as T
import NITTA.Intermediate.Functions
import NITTA.Intermediate.Types
import NITTA.Model.Problems
import NITTA.Model.ProcessorUnits.Types
import NITTA.Model.Types
import NITTA.Project
import NITTA.Utils
import NITTA.Utils.ProcessDescription
import Numeric.Interval.NonEmpty (inf, singleton, sup, (...))

{- |Type that contains expression:
@a + b = c@ is exression and it equals:
    @[[(False, "a"), (False, "b")], [(False, "c")]]@
@a + b = c; d - e = f@ is one expression too and it equals:
    @[[(False, "a"), (False, "b")], [(False, "c")], [(False, "d"), (True,
      "d")], [(False, "f")]]@
-}
data Job v x = Job
    { -- |Contains future parts expression to eval (c + d = e)
      tasks :: [[(Bool, v)]]
    , -- |Contain current parts expression (a + b = c)
      current :: [[(Bool, v)]]
    , -- |Func of this expression
      func :: F v x
    , -- |Flag indicates when evaluation ended
      calcEnd :: Bool
    }
    deriving (Eq, Show)

data Accum v x t = Accum
    { -- |List of jobs (expressions)
      work :: [Job v x]
```

```
    , -- |Current job
      currentWork :: Maybe (Job v x)
    , -- |Process
      process_ :: Process t (StepInfo v x t)
    , -- |Flag is indicated when new job starts
      isInit :: Bool
    }

instance (VarValTime v x t) => Show (Accum v x t) where
    show a =
        [__i|
            Accum:
                work                 = #{ work a }
                currentWork          = #{ currentWork a }
                process_             = #{ process_ a }
                isInit               = #{ isInit a }
        |]

instance (VarValTime v x t) => Default (Accum v x t) where
    def =
        Accum
            { work = []
            , currentWork = Nothing
            , process_ = def
            , isInit = True
            }

instance Default x => DefaultX (Accum v x t) x

tryBindJob f@Acc{actions} =
    Job
        { tasks = concat $ actionGroups actions
        , current = []
        , func = packF f
        , calcEnd = False
        }

actionGroups [] = []
actionGroups as =
    let (pushs, as') = span isPush as
        (pulls, as'') = span isPull as'
     in [ map (\(Push sign (I v)) -> (sign == Minus, v)) pushs
        , concatMap (\(Pull (O vs)) -> map (True,) $ elems vs) pulls
        ] :
        actionGroups as''

endpointOptionsJob Job{tasks = []} = []
endpointOptionsJob Job{tasks = (t : _), current = []} = map snd t
endpointOptionsJob Job{tasks = (t : ts), current = (c : _)}
    | null (t \\ c) && null ts = []
    | null $ t \\ c = map snd $ head ts
    | otherwise = map snd $ t \\ c

endpointDecisionJob j@Job{tasks = []} _ = j
endpointDecisionJob j@Job{tasks = tasks@(t : _), current = []} v = j{tasks =
      updateTasks current' tasks, current = current'}
    where
        ([(neg, _)], _) = partition ((== v) . snd) t
        current' = [[(neg, v)]]
endpointDecisionJob j@Job{tasks = tasks@(t : ts), current = (c : cs)} v
    | null $ t \\ c = endpointDecisionJob j{tasks = ts} v
```

```
    | t \\ c /= t && length t > length c = j{tasks = updateTasks
      currentInsert tasks, current = currentInsert}
    | otherwise = j{tasks = updateTasks currentAdd tasks, current =
      currentAdd}
    where
        ([val], _) = partition ((== v) . snd) t
        currentInsert = (val : c) : cs
        currentAdd = [val] : c : cs

updateTasks (c : _) tasks@(t : ts)
    | null $ t \\ c = ts
    | otherwise = tasks
updateTasks [] _ = error "Current is null"
updateTasks _ _ = error "Matching error updateTasks"

toTarget = even . length
toSource = odd . length

instance (VarValTime v x t, Num x) => ProcessorUnit (Accum v x t) v x t where
    tryBind f pu@Accum{work}
        | Just (Add a b c) <- castF f = Right pu{work = tryBindJob (Acc [Push
      Plus a, Push Plus b, Pull c]) : work}
        | Just (Sub a b c) <- castF f = Right pu{work = tryBindJob (Acc [Push
      Plus a, Push Minus b, Pull c]) : work}
        | Just f'@Acc{} <- castF f = Right pu{work = tryBindJob f' : work}
        | otherwise = Left $ "The function is unsupported by Accum: " ++ show
      f

    process = process_

instance (VarValTime v x t, Num x) => EndpointProblem (Accum v x t) v t where
    endpointOptions Accum{currentWork = Just a@Job{tasks, calcEnd}, process_
      = Process{nextTick = tick}}
        | toTarget tasks = targets
        | toSource tasks = sources
        where
            targets = map (\v -> EndpointSt (Target v) $ TimeConstraint (tick
      + 1 ... maxBound) (singleton 1)) (endpointOptionsJob a)
            sources = [EndpointSt (Source $ fromList (endpointOptionsJob a))
      $ TimeConstraint (max tick (tickSource calcEnd) ... maxBound) (1 ...
      maxBound)]
            tickSource True = tick + 1
            tickSource _ = tick + 3
    endpointOptions p@Accum{work, currentWork = Nothing} =
        concatMap (\a -> endpointOptions p{currentWork = Just a}) work
    endpointOptions _ = error "Error in matching in endpointOptions function"

    endpointDecision
        pu@Accum{currentWork = Just j@Job{tasks}, isInit}
        d@EndpointSt{epRole = Target v, epAt}
            | not (null tasks) && toTarget tasks =
                let job@Job{tasks = tasks', current = (((neg, _) : _) : _)} =
      endpointDecisionJob j v
                    sel = if isInit then ResetAndLoad neg else Load neg
                    (_, process_') = runSchedule pu $ do
                        updateTick (sup epAt)
                        scheduleEndpoint d $ scheduleInstruction epAt sel
                 in pu
                        { process_ = process_'
                        , currentWork = Just job{calcEnd = False}
                        , isInit = null tasks'
                        }
```

```
endpointDecision
    pu@Accum{currentWork = Just j@Job{tasks, current, func}, process_}
    d@EndpointSt{epRole = Source v, epAt}
        | not (null current) && toSource tasks =
            let job@Job{tasks = tasks'} = foldl endpointDecisionJob j
  (elems v)
                a = inf $ stepsInterval $ relatedEndpoints process_ $
  variables func
                (_, process_') = runSchedule pu $ do
                    endpoints <- scheduleEndpoint d $ scheduleInstruction
  (epAt -1) Out
                    when (null tasks') $ do
                        high <- scheduleFunction (a ... sup epAt) func
                        let low = endpoints ++ map pID (relatedEndpoints
  process_ $ variables func)
                        establishVerticalRelations high low

                    updateTick (sup epAt)
                    return endpoints
             in pu
                    { process_ = process_'
                    , currentWork = if null tasks' then Nothing else Just
  job{calcEnd = True}
                    , isInit = null tasks'
                    }
endpointDecision pu@Accum{work, currentWork = Nothing} d
    | Just job <- getJob work =
        endpointDecision pu{work = work \\ [job], currentWork = Just
  job{calcEnd = False}, isInit = True} d
    where
        getJob = find (\Job{func} -> d `isIn` func)
        e `isIn` f = oneOf (variables e) `member` variables f
endpointDecision pu d = error $ "error in Endpoint Decision function" ++
  show pu ++ show d

instance Connected (Accum v x t) where
    data Ports (Accum v x t) = AccumPorts {resetAcc, load, neg, oe ::
  SignalTag}
        deriving (Show)

instance IOConnected (Accum v x t) where
    data IOPorts (Accum v x t) = AccumIO deriving (Show)

instance Controllable (Accum v x t) where
    data Instruction (Accum v x t) = ResetAndLoad Bool | Load Bool | Out
  deriving (Show)

    data Microcode (Accum v x t) = Microcode
        { oeSignal :: Bool
        , resetAccSignal :: Bool
        , loadSignal :: Bool
        , negSignal :: Maybe Bool
        }
        deriving (Show, Eq, Ord)

    zipSignalTagsAndValues AccumPorts{..} Microcode{..} =
        [ (resetAcc, Bool resetAccSignal)
        , (load, Bool loadSignal)
        , (oe, Bool oeSignal)
        , (neg, maybe Undef Bool negSignal)
        ]
```

```
    usedPortTags AccumPorts{resetAcc, load, neg, oe} = [resetAcc, load, neg,
      oe]

    takePortTags (resetAcc : load : neg : oe : _) _ = AccumPorts resetAcc
      load neg oe
    takePortTags _ _ = error "can not take port tags, tags are over"

instance Default (Microcode (Accum v x t)) where
    def =
        Microcode
            { oeSignal = False
            , resetAccSignal = False
            , loadSignal = False
            , negSignal = Nothing
            }

instance UnambiguouslyDecode (Accum v x t) where
    decodeInstruction (ResetAndLoad neg) = def{resetAccSignal = True,
      loadSignal = True, negSignal = Just neg}
    decodeInstruction (Load neg) = def{resetAccSignal = False, loadSignal =
      True, negSignal = Just neg}
    decodeInstruction Out = def{oeSignal = True}

instance (Var v) => Locks (Accum v x t) v where
    locks Accum{currentWork = Nothing} = []
    locks Accum{currentWork = Just job, work} = locks' job work
        where
            locks' Job{tasks = []} _ = []
            locks' Job{current = []} _ = []
            locks' Job{tasks = (m : ms), current = (r : _)} other =
                [ Lock{lockBy, locked}
                | locked <- concatMap (map snd) ms
                , lockBy <- map snd (m \\ r)
                ]
                    ++ [ Lock{lockBy, locked}
                       | locked <- concatMap (concatMap (map snd) . tasks)
      other
                       , lockBy <- map snd (m ++ r)
                       ]

instance (VarValTime v x t) => TargetSystemComponent (Accum v x t) where
    moduleName _ _ = "pu_accum"
    hardware _tag _pu = FromLibrary "pu_accum.v"
    software _ _ = Empty
    hardwareInstance
        tag
        _pu
        UnitEnv
            { sigClk
            , sigRst
            , ctrlPorts = Just AccumPorts{..}
            , valueIn = Just (dataIn, attrIn)
            , valueOut = Just (dataOut, attrOut)
            } =
            [__i|
                pu_accum \#
                        ( .DATA_WIDTH( #{ dataWidth (def :: x) } )
                        , .ATTR_WIDTH( #{ attrWidth (def :: x) } )
                        ) #{ tag }
                    ( .clk( #{ sigClk } )
                    , .rst( #{ sigRst } )
                    , .signal_resetAcc( #{ resetAcc } )
```

```
                    , .signal_load( #{ load } )
                    , .signal_neg( #{ neg } )
                    , .signal_oe( #{ oe } )
                    , .data_in( #{ dataIn } )
                    , .attr_in( #{ attrIn } )
                    , .data_out( #{ dataOut } )
                    , .attr_out( #{ attrOut } )
                    );
            |]
    hardwareInstance _title _pu _env = error "internal error"

instance (Ord t) => WithFunctions (Accum v x t) (F v x) where
    functions Accum{process_, work} =
        functions process_ ++ map func work

instance (VarValTime v x t) => Testable (Accum v x t) v x where
    testBenchImplementation prj@Project{pName, pUnit} =
        let tbcSignalsConst = ["resetAcc", "load", "oe", "neg"]

            showMicrocode Microcode{resetAccSignal, loadSignal, oeSignal,
      negSignal} =
                ([i|resetAcc <= #{ bool2verilog resetAccSignal };|] ::
      String)
                    <> [i| load <= #{ bool2verilog loadSignal };|]
                    <> [i| oe <= #{ bool2verilog oeSignal };|]
                    <> [i| neg <= #{ bool2verilog $ fromMaybe False negSignal
      };|]

            conf =
                SnippetTestBenchConf
                    { tbcSignals = tbcSignalsConst
                    , tbcPorts =
                        AccumPorts
                            { resetAcc = SignalTag "resetAcc"
                            , load = SignalTag "load"
                            , oe = SignalTag "oe"
                            , neg = SignalTag "neg"
                            }
                    , tbcMC2verilogLiteral = T.pack . showMicrocode
                    }
         in Immediate (toString $ moduleName pName pUnit <> "_tb.v") $
      snippetTestBench prj conf

instance IOTestBench (Accum v x t) v x

instance BreakLoopProblem (Accum v x t) v x
instance ConstantFoldingProblem (Accum v x t) v x
instance OptimizeAccumProblem (Accum v x t) v x
instance ResolveDeadlockProblem (Accum v x t) v x
```

ПРИЛОЖЕНИЕ Г (ЛИСТИНГ ОПТИМИЗАЦИЯ АККУМУЛЯТОРА)

```
module NITTA.Model.Problems.Refactor.OptimizeAccum (
    OptimizeAccum (..),
    OptimizeAccumProblem (..),
) where

import qualified Data.List as L
import qualified Data.Map as M
import Data.Maybe
import qualified Data.Set as S
import GHC.Generics
import NITTA.Intermediate.Functions
import NITTA.Intermediate.Types

{- |OptimizeAccum example:
> OptimizeAccum [+a +b => tmp1; +tmp1 +c => res] [+a +b +c => d]
before:
> [+a +b => tmp1; +tmp1 +c => res]
after:
> [+a +b +c => res]
== Doctest optimize example
>>> let a = constant 1 ["a"]
>>> let b = constant 2 ["b"]
>>> let c = constant 3 ["c"]
>>> let tmp1 = add "a" "b" ["tmp1"]
>>> let res = add "tmp1" "c" ["res"]
>>> let loopRes = loop 1 "e" ["res"]
>>> let fs = [a, b, c, tmp1, res, loopRes] :: [F String Int]
>>> optimizeAccumDecision fs $ head $ optimizeAccumOptions fs
[Acc(+a +b +c = res),const(1) = a,const(2) = b,const(3) = c,Loop (X 1) (O
[res]) (I e)]
-}
data OptimizeAccum v x = OptimizeAccum
    { refOld :: [F v x]
    , refNew :: [F v x]
    }
    deriving (Generic, Show, Eq)

class OptimizeAccumProblem u v x | u -> v x where
    -- |Function takes algorithm in 'DataFlowGraph' and return list of
'Refactor' that can be done
    optimizeAccumOptions :: u -> [OptimizeAccum v x]
    optimizeAccumOptions _ = []

    -- |Function takes 'OptimizeAccum' and modify 'DataFlowGraph'
    optimizeAccumDecision :: u -> OptimizeAccum v x -> u
    optimizeAccumDecision _ _ = error "not implemented"

instance (Var v, Val x) => OptimizeAccumProblem [F v x] v x where
    optimizeAccumOptions fs = res
        where
            res =
                L.nub
                    [ OptimizeAccum{refOld, refNew}
                    | refOld <- selectClusters $ filter isSupportByAccum fs
                    , let refNew = optimizeCluster refOld
```

```
                    , S.fromList refOld /= S.fromList refNew
                    ]

    optimizeAccumDecision fs OptimizeAccum{refOld, refNew} = refNew <> (fs
L.\\ refOld)

selectClusters fs =
    L.nubBy
        (\a b -> S.fromList a == S.fromList b)
        [ [f, f']
        | f <- fs
        , f' <- fs
        , f' /= f
        , inputOutputIntersect f f'
        ]
    where
        inputOutputIntersect f1 f2 = isIntersection (inputs f1) (outputs f2)
|| isIntersection (inputs f2) (outputs f1)
        isIntersection a b = not $ S.disjoint a b

isSupportByAccum f
    | Just Add{} <- castF f = True
    | Just Sub{} <- castF f = True
    | Just Acc{} <- castF f = True
    | otherwise = False

-- |Create Map String (HistoryTree (F v x)), where Key is input label and
Value is FU that contain this input label
containerMapCreate fs =
    M.unions $
        map
            ( \f ->
                foldl
                    ( \dataMap k ->
                        M.insertWith (++) k [f] dataMap
                    )
                    M.empty
                    (S.toList $ inputs f)
            )
            fs

-- |Takes container and refactor it, if it can be
optimizeCluster fs = concatMap refactored fs
    where
        containerMap = containerMapCreate fs

        refactored f =
            concatMap
                ( \o ->
                    case M.findWithDefault [] o containerMap of
                        [] -> []
                        matchedFUs -> concatMap (refactorFunction f)
matchedFUs
                )
                (S.toList $ outputs f)

refactorFunction f' f
    | Just (Acc lst') <- castF f'
      , Just (Acc lst) <- castF f
      , let singleOutBool = (1 ==) $ length $ outputs f'
            isOutInpIntersect =
                any
```

```
                    ( \case
                        Push _ (I v) -> elem v $ outputs f'
                        _ -> False
                    )
                    lst
            makeRefactor = singleOutBool && isOutInpIntersect
         in makeRefactor =
        let subs _ (Push Minus _) (Push Plus v) = Just $ Push Minus v
            subs _ (Push Minus _) (Push Minus v) = Just $ Push Plus v
            subs _ (Push Plus _) push@(Push _ _) = Just push
            subs v _ pull@(Pull _) = deleteFromPull v pull
            subs _ _ _ = error "Pull can not be here"

            refactorAcc _ _ (Pull o) = [Pull o]
            refactorAcc accList accNew accOld@(Push s i@(I v))
                | elem v $ outputs accNew = mapMaybe (subs v accOld) accList
                | s == Minus = [Push Minus i]
                | s == Plus = [Push Plus i]
            refactorAcc _ _ (Push _ (I _)) = undefined
         in [packF $ Acc $ concatMap (refactorAcc lst' f') lst]
    | Just f1 <- fromAddSub f'
      , Just f2 <- fromAddSub f
      , (1 ==) $ length $ outputs f' = case refactorFunction f1 f2 of
        [fNew] -> [fNew]
        _ -> [f, f']
    | otherwise = [f, f']

deleteFromPull v (Pull (O s))
    | S.null deleted = Nothing
    | otherwise = Just $ Pull $ O deleted
    where
        deleted = S.delete v s
deleteFromPull _ (Push _ _) = error "delete only Pull"

fromAddSub f
    | Just (Add in1 in2 (O out)) <- castF f =
        Just $
            acc $
                [Push Plus in1, Push Plus in2] ++ [Pull $ O $ S.fromList [o]
| o <- S.toList out]
    | Just (Sub in1 in2 (O out)) <- castF f =
        Just $
            acc $
                [Push Plus in1, Push Minus in2] ++ [Pull $ O $ S.fromList [o]
| o <- S.toList out]
    | Just Acc{} <- castF f = Just f
    | otherwise = Nothing
```

ПРИЛОЖЕНИЕ Д (ЛИСТИНГ СВЁРТКА КОНСТАНТ)

```
{-
== Example from ASCII diagram
>>> let a = constant 1 ["a"]
>>> let b = constant 2 ["b"]
>>> let res = add "a" "b" ["res"]
>>> loopRes = loop 1 "e" ["res"]
>>> let fs = [a, b, res, loopRes] :: [F String Int]
>>> constantFoldingDecision fs $ head $ constantFoldingOptions fs
[Loop (X 1) (O [res]) (I e),const(3) = res]
-}
module NITTA.Model.Problems.Refactor.ConstantFolding (
    ConstantFolding (..),
    ConstantFoldingProblem (..),
) where

import Data.Default
import qualified Data.List as L
import qualified Data.Map as M
import qualified Data.Set as S
import GHC.Generics
import NITTA.Intermediate.Functions
import NITTA.Intermediate.Types

data ConstantFolding v x = ConstantFolding
    { cRefOld :: [F v x]
    , cRefNew :: [F v x]
    }
    deriving (Generic, Show, Eq)

class ConstantFoldingProblem u v x | u -> v x where
    -- |Function takes algorithm in 'DataFlowGraph' and return list of
      optimizations that can be done
    constantFoldingOptions :: u -> [ConstantFolding v x]
    constantFoldingOptions _ = []

    -- |Function takes 'ConstantFolding' and modify 'DataFlowGraph'
    constantFoldingDecision :: u -> ConstantFolding v x -> u
    constantFoldingDecision _ _ = error "not implemented"

instance (Var v, Val x) => ConstantFoldingProblem [F v x] v x where
    constantFoldingOptions fs =
        let clusters = selectClusters fs
            evaluatedClusters = map evalCluster clusters
            zipOfClusters = zip clusters evaluatedClusters
            filteredZip = filter (\case ([_], _) -> False; _ -> True)
      zipOfClusters
            options = [ConstantFolding{cRefOld = c, cRefNew = ec} | (c, ec)
      <- filteredZip, c /= ec]
            optionsFiltered = filter isBlankOptions options
            isBlankOptions = not . null . constantFoldingDecision fs
         in optionsFiltered

    constantFoldingDecision fs ConstantFolding{cRefOld, cRefNew}
        | cRefOld == cRefNew = cRefNew
        | otherwise = deleteExtraF $ (fs L.\\ cRefOld) <> cRefNew
```

```
isConst f
    | Just Constant{} <- castF f = True
    | otherwise = False

selectClusters fs =
    let consts = filter isConst fs
        isIntersection a b = not . S.null $ S.intersection a b
        inputsAreConst f = inputs f `S.isSubsetOf` S.unions (map outputs
      consts)
        getInputConsts f = filter (\c -> outputs c `isIntersection` inputs f)
      consts
        createCluster f
            | inputsAreConst f = f : getInputConsts f
            | otherwise = [f]
     in map createCluster fs

evalCluster [f] = [f]
evalCluster fs = outputResult
    where
        (consts, [f]) = L.partition isConst fs
        cntx = CycleCntx $ M.fromList $ concatMap (simulate def) consts
        outputResult
            | null $ outputs f = fs
            | otherwise = map (\(v, x) -> constant x [v]) (simulate cntx f)
      <> consts

deleteExtraF fs =
    L.nub
        [ f1
        | f1 <- fs
        , f2 <- fs
        , f1 /= f2
        , not $ null (variables f1 `S.intersection` variables f2)
        ]
```

ПРИЛОЖЕНИЕ Е (ЛИСТИНГ ГРУППОВОЕ СВЯЗЫВАНИЕ)

```
module NITTA.Model.Problems.Bind (
    Bind (..),
    BindProblem (..),
    GroupBindingT (..),
) where

import qualified Data.Set as S
import Data.String.ToString
import GHC.Generics
import NITTA.Intermediate.Types

data GroupBindingT = NonAlternativeBinds | AllBinds | FirstWaveBinds
    deriving (Show, Generic, Eq)

data Bind tag v x
    = Bind (F v x) tag
    | GroupBinding GroupBindingT [Bind tag v x]
    deriving (Generic)

instance (ToString tag) => Show (Bind tag v x) where
    show (Bind f tag) = "Bind " <> show f <> " " <> toString tag
    show (GroupBinding t bindings) = "GroupBiding (" <> show t <> ") " <>
      concatMap (\(Bind f tag) -> show f <> " " <> toString tag) bindings

class BindProblem u tag v x | u -> tag v x where
    bindOptions :: u -> [Bind tag v x]
    bindDecision :: u -> Bind tag v x -> u

instance (Var v) => Variables (Bind tab v x) v where
    variables (Bind f _tag) = variables f
    variables (GroupBinding _ binds) = S.unions $ map variables binds

instance
    (UnitTag tag, VarValTime v x t) =>
    BindProblem (BusNetwork tag v x t) tag v x
    where
    bindOptions BusNetwork{bnRemains, bnPus} = if not $ null optionsList then
      groupBinding optionsList ++ concat optionsList else []
        where
            optionsFor f =
                [ Bind f puTitle
                | (puTitle, pu) <- M.assocs bnPus
                , allowToProcess f pu
                ]
            optionsList = map optionsFor bnRemains
            compose [] buff = buff
            compose (x:xs) [] = compose xs newbuff
                where
                    newbuff = [ [nvalue] | nvalue <- x ]
            compose (x:xs) buff = compose xs newbuff
                where
                    newbuff = [ values ++ [nvalue] | values <- buff, nvalue
      <- x ]
            groupBinding options
```

```
            | not $ null $ filter (\x -> length x > 1) options = map
  (GroupBinding AllBinds) $ compose options []
            | otherwise = [GroupBinding NonAlternativeBinds $ concat
  options]

bindDecision n@BusNetwork{bnProcess, bnPus, bnBinded, bnRemains} (Bind f
  tag) =
    n
        { bnPus = M.adjust (bind f) tag bnPus
        , bnBinded = registerBinding tag f bnBinded
        , bnProcess = execScheduleWithProcess n bnProcess $
  scheduleFunctionBind f
        , bnRemains = filter (/= f) bnRemains
        }
bindDecision n@BusNetwork{bnProcess, bnPus, bnBinded, bnRemains}
  gp@(GroupBinding NonAlternativeBinds binds) =
    n
        { -- { bnPus = bnPus'
          -- , bnBinded = bnBinded'
          bnPus = L.foldl' (\m (Bind f tag) -> M.adjust (bind f) tag m)
  bnPus binds
        , bnBinded = L.foldl' (\m (Bind f tag) -> registerBinding tag f
  m) bnBinded binds
        , bnProcess = execScheduleWithProcess n bnProcess $
  scheduleGroupBinding gp
        , bnRemains = bnRemains L.\\ map (\(Bind f _) -> f) binds
        }

bindDecision n@BusNetwork{bnProcess, bnPus, bnBinded, bnRemains}
  gp@(GroupBinding AllBinds binds) =
    n
        { -- { bnPus = bnPus'
          -- , bnBinded = bnBinded'
          bnPus = L.foldl' (\m (Bind f tag) -> M.adjust (bind f) tag m)
  bnPus binds
        , bnBinded = L.foldl' (\m (Bind f tag) -> registerBinding tag f
  m) bnBinded binds
        , bnProcess = execScheduleWithProcess n bnProcess $
  scheduleGroupBinding gp
        , bnRemains = bnRemains L.\\ map (\(Bind f _) -> f) binds
        }
```

ПРИЛОЖЕНИЕ Ж (ЛИСТИНГ СБОР ДАННЫХ ДЛЯ АНАЛИЗА ДЕРЕВА СИНТЕЗА)

```
module NITTA.Synthesis.Analysis (
    getTreeInfo,
    TreeInfo (..),
) where

import Control.Concurrent.STM
import qualified Data.HashMap.Strict as HM
import GHC.Generics
import NITTA.Model.TargetSystem (processDuration)
import NITTA.Synthesis.Explore (isComplete, isLeaf)
import NITTA.Synthesis.Types

-- |Metrics of synthesis tree process
data TreeInfo = TreeInfo
    { nodes :: Int
    , success :: Int
    , failed :: Int
    , notProcessed :: Int
    , durationSuccess :: HM.HashMap Int Int
    , stepsSuccess :: HM.HashMap Int Int
    }
    deriving (Generic, Show)

instance Semigroup TreeInfo where
    (<>) synthesisInfo1 synthesisInfo2 =
        let synthesisInfoList = [synthesisInfo1, synthesisInfo2]
            durationSuccessList = map durationSuccess synthesisInfoList
            stepsSuccessList = map stepsSuccess synthesisInfoList
         in TreeInfo
                { nodes = sum $ map nodes synthesisInfoList
                , success = sum $ map success synthesisInfoList
                , failed = sum $ map failed synthesisInfoList
                , notProcessed = sum $ map notProcessed synthesisInfoList
                , durationSuccess = if not $ null durationSuccessList then
      foldl1 (HM.unionWith (+)) durationSuccessList else HM.empty
                , stepsSuccess = if not $ null stepsSuccessList then foldl1
      (HM.unionWith (+)) stepsSuccessList else HM.empty
                }

instance Monoid TreeInfo where
    mempty =
        TreeInfo
            { nodes = 0
            , success = 0
            , failed = 0
            , notProcessed = 0
            , durationSuccess = HM.empty
            , stepsSuccess = HM.empty
            }

getTreeInfo tree@Tree{sID = SID sid, sSubForestVar} = do
    subForestM <- atomically $ tryReadTMVar sSubForestVar
    subForestInfo <- maybe (return mempty) (fmap mconcat . mapM getTreeInfo)
      subForestM
```

```
let isSuccess = isComplete tree && isLeaf tree
let isFail = (not . isComplete) tree && isLeaf tree
let duration = fromEnum $ processDuration $ sTarget $ sState tree
let successDepends value field =
        if not isSuccess
            then field subForestInfo
            else HM.alter (Just . maybe 1 (+ 1)) value $ field
  subForestInfo
return $
    TreeInfo
        { nodes = 1 + nodes subForestInfo
        , success = if isSuccess then 1 else 0 + success subForestInfo
        , failed = if isFail then 1 else 0 + failed subForestInfo
        , notProcessed = maybe 1 (const 0) subForestM + notProcessed
  subForestInfo
        , durationSuccess = successDepends duration durationSuccess
        , stepsSuccess = successDepends (length sid) stepsSuccess
        }
```