

Санкт-Петербургский государственный университет

Кубышкин Ефим Алексеевич

Выпускная квалификационная работа

Разработка прототипа специализированного процессора, основанного на Interaction Nets

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Григорьев С. В.

Рецензент:
инженер-исследователь, «Лаборатория YADRO» СПбГУ, Косарев Д. С.

Санкт-Петербург
2025

Saint Petersburg State University

Efim Kubyshkin

Bachelor's Thesis

Development of a prototype of a specialized processor based on Interaction Nets

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc., docent S.V. Grigorev

Reviewer:
research engineer at YADRO Lab SPbSU D.S. Kosarev

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	5
2. Обзор	6
2.1. INTERACTION NETS	6
2.2. Альтернативы	9
2.3. CLASH	10
2.4. Выводы	10
3. Общая структура проекта	11
4. Генерация вычислителя	13
4.1. Микроархитектура ядра вычислителя	13
4.2. Технические особенности	15
5. Тестирование стенда	19
5.1. Проверка работоспособности	19
5.2. Анализ аппаратных характеристик	19
5.3. Результаты	20
Заключение	22
Список литературы	23

Введение

Современные вычислительные системы сталкиваются с растущей потребностью в параллельной обработке данных [9]. Однако традиционные архитектуры процессоров часто оказываются неэффективными при работе с нерегулярным параллелизмом [5], характерным для множества научных и инженерных задач. В таких случаях специализированные аппаратные решения могут значительно повысить производительность. Эти проблемы пытаются решать с помощью реализации аппаратных ускорителей.

Одним из перспективных подходов к организации вычислений является использование крупно-гранулярных реконфигурируемых архитектур (CGRA) [12], которые сочетают гибкость программных моделей с высокой производительностью специализированных аппаратных решений. Они находят применение в задачах обработки сигналов [4], машинного обучения [13, 15] и научных вычислений, где требуется адаптация структуры вычислений под специфику задачи [11, 1].

Одной из математических моделей, естественным образом поддерживающих нерегулярные параллельные вычисления, являются INTERACTION NETS. Этот формализм был предложен Ивом Лафонтон в 1989 году [8] и обладает рядом свойств, делающих его удобным для параллельных и параметризуемых вычислителей. Однако на данный момент существуют только программные реализации INTERACTION NETS [7], которые испытывают ограничения традиционных архитектур и не используют полный потенциал этого формализма. В данной работе рассматривается возможность аппаратной реализации INTERACTION NETS. Основная цель — разработка специализированного вычислителя на основе данной модели, который позволит исследовать её жизнеспособность в «железе». Для этого используется язык CLASH [2], позволяющий описывать цифровые схемы в терминах функционального программирования, что упрощает разработку и тестирование.

Таким образом, работа направлена на исследование возможности аппаратной реализации INTERACTION NETS.

1. Постановка задачи

Целью работы является реализация параметризуемого вычислителя на основе INTERACTION NETS. Для её выполнения были поставлены следующие задачи.

1. Спроектировать общую структуру проекта.
2. Спроектировать архитектуру ядра вычислителя.
3. Реализовать одноядерный прототип вычислителя.
4. Провести тестирование работоспособности вычислителя.

2. Обзор

В этом разделе будет описана модель INTERACTION NETS, которая послужила основой вычислителя. Также будет обзор языка CLASH, на котором реализован проект.

2.1. INTERACTION NETS

Будет использоваться вариант определений, данных И. Лафонтом в 1989 году [8].

Определение 1 (Агент). *Агентом называется символ фиксированного алфавита Σ , для которого*

- *определена **арность** — функция $Ar : \Sigma \rightarrow \mathbb{N} \cup \{0\}$;*
- *есть один **главный порт** и равное его арности количество **дополнительных портов**.*

Определение 2 (Сеть). *Сетью над алфавитом Σ называется помеченный неориентированный граф, в котором*

- *каждый узел помечен агентом;*
- *рёбра могут быть только между портами;*
- *в каждый порт входит не более одного ребра.*

$\mathcal{N}_\Sigma$ — множество всех возможных сетей над алфавитом Σ .

Будем говорить, что порт **свободен**, если из него не идёт ребро. В качестве сети также может выступать просто набор рёбер, тогда каждый из концов ребра будет также считаться свободным портом.

Определение 3 (Интерфейс). *Интерфейсом $\mathcal{I}(n)$ называется множество всех свободный портов сети n .*

Будем говорить, что порт p_a **подключен** к порту p_b , если между ними есть ребро.

Так как сеть является графом, её удобно представлять в графическом виде. В этом представлении есть некоторые особенности.

- Свободные порты рисуются как никуда не идущие прямые.
- Узлы рисуются треугольниками с одним острым углом и двумя скруглёнными.

Рассмотрим пример сети, состоящий из трёх узлов, множество агентов которых $\Sigma = \{\beta, \alpha\}$ с арностями $Ar(\beta) = 0$, $Ar(\alpha) = 2$. Главные порты α -узлов подключены друг к другу, а главный порт β -узла подключен к одному из дополнительных портов левого α -узла. Свободных портов этой сети три: один у левого α -агента и два у правого. Они же и представляют собой интерфейс. Вся эта информация представлена на рис. 1. В большинстве случаев изображение более информативно, и в документации проекта часто можно найти рисунки в приведённом стиле.

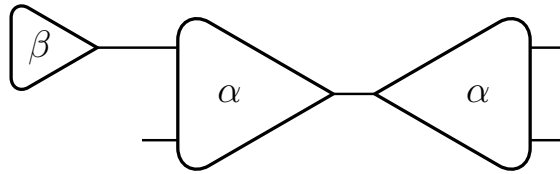


Рис. 1: Пример сети из трёх узлов

Вычисления в INTERACTION NETS происходят с помощью **редукций**: переписыванием активной пары в сеть.

Определение 4 (Правило редукции). *Правило редукции — это правило переписывания графа. $\mathcal{R}$ — множество правил редукции.*

Более формально, правило редукции это триплет $r \in \Sigma \times \Sigma \times \mathcal{N}_\Sigma$, для которого верны следующие свойства.

- **Симметричность:**

$$(l_1, l_2, n) \in \mathcal{R} \Rightarrow (l_2, l_1, n) \in \mathcal{R}$$

- **Детерминированность:**

$$(l_1, l_2, n_1) \in \mathcal{R} \wedge (m_1, m_2, n_2) \in \mathcal{R} \Rightarrow (l_1, l_2) \neq (m_1, m_2)$$

- **Сохранность интерфейса:**

$$\forall (l_1, l_2, n) \in \mathcal{R}, \text{Ar}(l_1) + \text{Ar}(l_2) = |\mathcal{I}(n)|$$

Например, для того, чтобы реализовать простое λ -исчисление, достаточно определить агенты $\Sigma = \{\lambda, @\}$ и правило редукции, представленное на рис. 2.

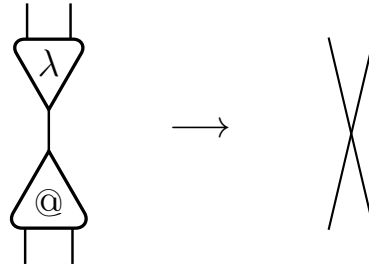


Рис. 2: Правило редукции, представляющее собой применение λ -терма

В той же работе И. Лафонта [8] показано, что через этот формализм выражается λ -исчисление (например, на рис. 2 показано правило для β -редукции). В дополнение ко всему, редукции можно выполнять независимо друг от друга. Что и служит основной причиной естественности INTERACTION NETS для параллельных вычислений.

Также видно и большую степень свободы: можно определять как специфические агенты, так и правила редукции для них.

Программные реализации INTERACTION NETS

На данный момент INTERACTION NETS имеют несколько программных реализаций, позволяющих моделировать их поведение на традиционных архитектурах, например.

- **INPLA** [7] — одна из наиболее известных software реализаций INTERACTION NETS. Она предоставляет инструменты для работы

с графовыми представлениями и оптимизированными правилами редукции.

- **HVM**¹ — высокопроизводительная виртуальная машина, основанная на INTERACTION NETS. HVM оптимизирована для вычислений, использующих оптимальные стратегии редукции, что делает её значительно быстрее традиционных реализаций лямбда-исчисления.

Программные реализации INTERACTION NETS показывают, что этот формализм является мощным инструментом для представления параллельных вычислений.

Они, тем не менее, сталкиваются с фундаментальными проблемами традиционных архитектур с нерегулярным параллелизмом: высокими накладными расходами на иерархию памяти и промахи кэшей.

2.2. Альтернативы

INTERACTION NETS не является ни единственной, ни самой популярной моделью для создания конфигурируемых вычислителей. В этом разделе будет рассмотрен альтернативный подход — transport triggered architecture [3] — а также его реализации.

Transport Triggered Architecture (ТТА) — это класс архитектур, в которых вычисления являются побочным эффектом от перемещения данных между функциональными блоками. Это позволяет достичь параллелизма на уровне инструкций. Существуют реализации данного подхода, например.

- **ТСЕ** [6] — это зрелая среда разработки от Тамперского университета, поддерживающая проектирование, компиляцию и симуляцию ТТА-процессоров. Проект также поддерживает весь пайплайн: от программы на языке высокого уровня до прошивки FPGA.
- **NITTA**² совмещает подходы ТТА и NISC для работы с нерегулярным параллелизмом, поддерживает описание на высокоуровневом

¹GitHub проекта: <https://github.com/HigherOrderCO/HVM>

²GitHub проекта: <https://github.com/ryukzak/nitta>

языке программирования и высокую параметризуемость.

Эти проекты показывают интерес академического сообщества к решению проблемы нерегулярного параллелизма через альтернативные архитектуры. Более того, оба решения делают акцент на реконфигурируемость, параллельность и поддержку описания на высокоуровневом языке программирования.

2.3. CLASH

Описание логических схем на привычных HDL (SystemVerilog и VHDL) бывает затруднительно из-за отсутствия удобных механизмов создания высокоуровневых абстракций и работы с ними. Тестирование также вызывает трудности: писать testbench файлы утомительно. Эти проблемы, активно используя особенности HASKELL, решает CLASH [2].

В CLASH активно используется typelevel programming для абстрагирования низкоуровневых особенностей и статической проверки корректности. Части написанных компонентов часто представляют собой чистые функции, которые удобно тестировать с помощью HASKELL библиотек, например, Hedgehog³ и HUnit⁴. А для сложной последовательностной логики у CLASH есть симулятор, который позволяет писать интеграционные тесты, не выходя из инфраструктуры HASKELL.

2.4. Выводы

INTERACTION NETS является альтернативной моделью вычислений, для которой естественна параллельность и параметризуемость. Для реализации реконфигурируемой архитектуры, основанной на этой модели могут быть полезны высокоуровневые абстракции. CLASH, реализованный на языке HASKELL подходит для этого в силу развитой системы типов и функциональной парадигмы.

³GitHub библиотеки: <https://github.com/hedgehogqa/haskell-hedgehog>

⁴GitHub библиотеки: <https://github.com/hspec/HUnit>

3. Общая структура проекта

Основной целью проекта является создание инфраструктуры для создания параметризуемых специализированных вычислителей на основе INTERACTION NETS. Исходя из этого, были выдвинуты следующие требования.

- Возможность параметризовать компилятор и ядро вычислителя типами агентов и правилами редукции.
- Возможность сбора статистики: размер сети, количество редукций, время исполнения и другой.
- Использование единого стека технологий — гомогенность.
- Получение полнофункционального прототипа, содержащего все компоненты, важнее, чем детальная проработка какого-то отдельного компонента.
- Возможность вносить изменения в любые компоненты.

Общая структура проекта представлена на рис. 3. Пользователю надо написать программу на модельном языке **LAMAGRAN ML**, дизайн и реализация которого подробнее описана в другой работе. А также спецификацию для INTERACTION NETS, которая должна содержать типы агентов, правила редукции и схему компиляции. Имея программу и спецификацию, генерируется код на SYSTEMVERILOG, с помощью которого можно прошить FPGA и получить вычислитель. Далее будет подробнее описана каждая из представленных частей проекта.

Compiler Отвечает за построение абстрактного синтаксического дерева (AST) по коду на языке LML, его интерпретацию и «рассахаривание».

Middle Содержит все абстракции для INTERACTION NETS а так же транслятор из высокоуровневого AST в AST для специализированной сети, для которого тоже есть интерпретатор.

Hardware Отвечает за генерацию «ядра» вычислителя по спецификации INTERACTION NETS.

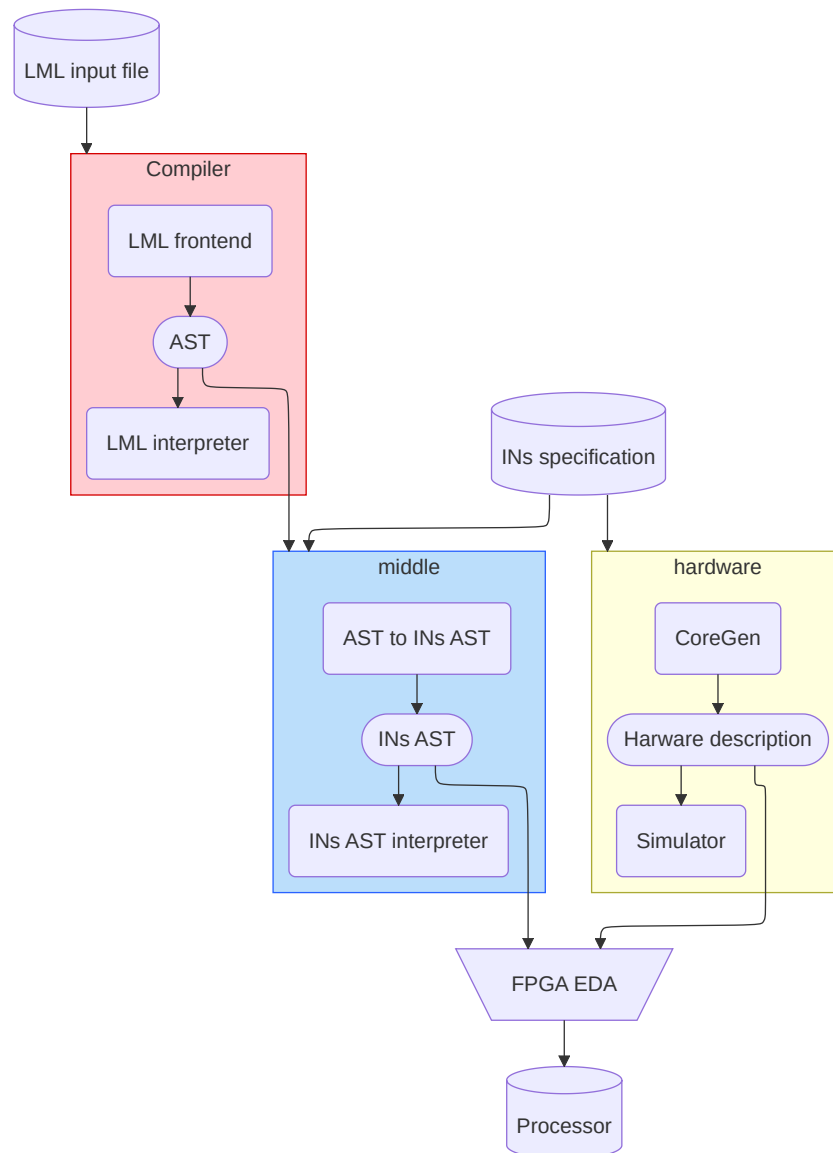


Рис. 3: Высокоуровневая общая архитектура проекта

Интерпретаторы на разных уровнях позволяют поэтапно тестировать компоненты: результат выполнения программы должен совпадать для всех интерпретаторов, симулятора и конкретного вычислителя.

Проект является командным, данная работа посвящена **Hardware** компоненте.

4. Генерация вычислителя

Компонента, которая отвечает за генерацию вычислителя по спецификации INTERACTION NETS состоит из четырёх больших блоков. Пайплайн работы вычислителя является циклом работы трёх из них (один вспомогательный, он отвечает за спецификацию INTERACTION NETS). Первая часть данного раздела будет посвящена описанию этого пайплайна, а вторая — описанию особенностей реализации спроектированного решения.

4.1. Микроархитектура ядра вычислителя

Работа ядра вычислителя происходит за счёт применения абстрактного правила редукции из спецификации к конкретной активной паре. Для этого необходимо

- (a) Работать с памятью: выделять достаточно места под результат редукции, подгружать следующую активную пару, записывать результат.
- (b) Применять абстрактное правило редукции для конкретных узлов, учитывая их локальную топологию.
- (c) Встраивать локально произошедшие изменения в глобальную сеть: интерфейсные узлы должны иметь корректные указатели на порты новых узлов или других интерфейсных узлов.

На рис. 4 показана микроархитектура ядра вычислителя. Каждая компонента, кроме **INSpec**, отвечает за одну из перечисленных выше функций. Далее будет подробнее показан пайплайн работы ядра.

MemoryManager

Вначале необходимо подгрузить из оперативной памяти активную пару. Список адресов активных пар сначала даётся компилятором, а далее поддерживается в актуальном состоянии при помощи **Scheduler**.

Reducer

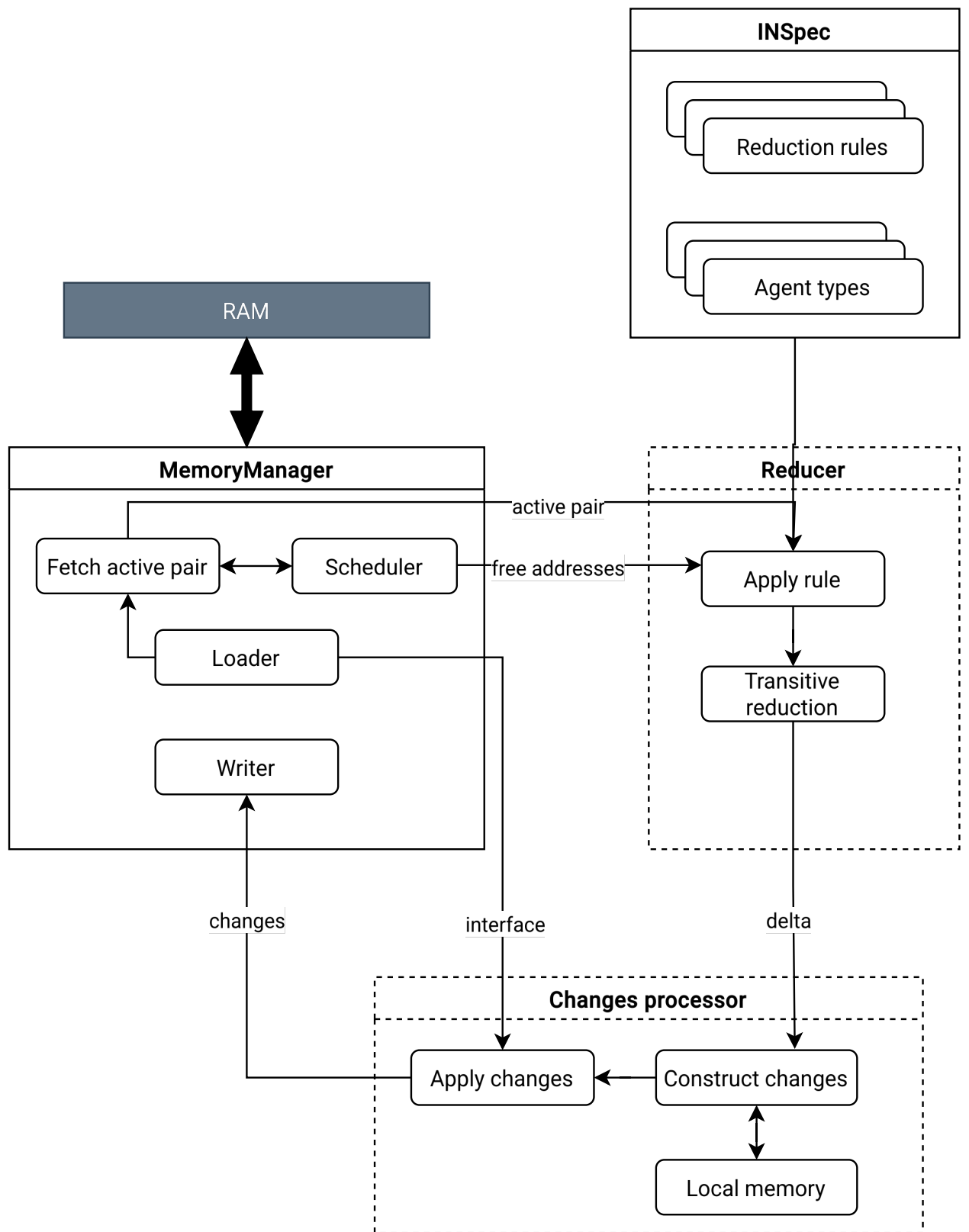


Рис. 4: Микроархитектура ядра вычислителя

Эта пара узлов передаётся в **Apply rule**, которая при помощи **INSpec** определяет правило, и количество новых узлов в результате. После ре-

дукции появляется новая сеть, которая может содержать рёбра, которые можно сократить (пример такой ситуации можно увидеть на рис. 5). За обработку таких ситуаций отвечает **Transitive reduction**.

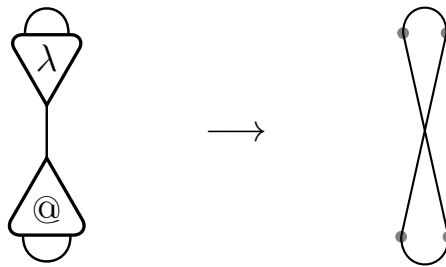


Рис. 5: Пример, когда рёбра соединены сами с собой, поэтому должны исчезнуть

Changes processor

В результате получается сеть (**delta**), которую необходимо встроить в глобальную сеть. Более точно, надо обработать две ситуации: или в портах интерфейсных узлов должны появиться адреса портов только что созданных узлов (см. рис. 6), или же они должны будут ссылаться на порты других интерфейсных узлов (см. рис. 7). Для этого необходимо сконструировать изменения портов во внешних узлах. После эти изменения необходимо применить к интерфейсу, для чего его нужно подгрузить из оперативной памяти, а потом записать изменённый интерфейс.

Этот цикл продолжается до тех пор, пока в сети есть активные пары

4.2. Технические особенности

Работа направлена на создание вычислителя на FPGA, так что в этом разделе будет рассмотрено, как из высокоуровневого функционального языка получить прошивку, и как проект устроен с точки зрения инфраструктуры.

Сборка и синтез

В нашем распоряжении имеется две платы от компании Gowin, так что синтез и прошивание платы актуально для FPGA

этого производителя. Для генерации SystemVerilog файла, содержащего всю логику вычислителя нужно запустить команду `stack run -- clash Core.Core --systemverilog -fclash-clear`. Полученные файлы (файл с логикой, файл с используемыми типами и файл описания временных ограничений) нужно указать в `tcl` скрипте. С помощью него, используя `gw_sh`, можно получить проект и синтезировать прошивку для платы в `gw_ide`.

Тестирование

CLASH позволяет описывать комбинационную логику как обычный код на HASKELL (за исключением отсутствия некоторых функций из Prelude и использования векторов вместо списков). Это позволяет писать как unit тесты для таких функций, так и property based тесты. Последовательностную же логику можно тестировать опять не выходя за пределы инфраструктуры HASKELL с помощью симулятора CLASH. Соответственно, были написаны и интеграционные тесты. Спускаясь на следующий уровень абстракции, на CLASH можно написать test bench файлы на Verilog. И в конце, после прошивки платы можно с помощью средств Gowin IDE отслеживать в режиме дебага конкретные сигналы, проверяя корректность.

Такое тестирование на нескольких уровнях абстракции помогает отслеживать ошибки и эффективно их исправлять.

Параметризуемость

Одним из ключевых требований к работе является высокая параметризуемость вычислителя. В ходе разработки были выделены следующие параметры, которые можно менять.

- Типы агентов.
- Правила редукции.
- Максимальное количество портов узла

- Максимальное количество возникающий в результате редукции новых узлов и рёбер.

Все эти параметры явно вынесены в типы. Более того, некоторые полезные характеристики можно вычислить на уровне типов, например, зная максимальное количество портов, точно известно, что количество узлов в интерфейсе не превосходит удвоенное количество портов. Главными параметрами, от которых по факту зависят все остальные, являются правила редукции и типы агентов. Для реализации их абстрагирования и, как следствие, изменчивости был спроектирован типовой класс **INet** (см. листинг 1). Пользователю необходимо написать типы агентов, для чего предусмотрена возможность задавать их через `TEMPLATE HASKELL` (например, чтобы легко задать агенты для целых чисел). После этого реализовать функцию редукции в зависимости от агентов, в этом же классе задаются максимальное количество новых узлов (`nodesNumber`), рёбер (`edgesNumber`) и максимальное количество портов в узле (`portsNumber`). В дальнейшем планируется написать eDSL для задания всех этих параметров.

```
class INet agentType nodesNumber edgesNumber portsNumber where
makeReduction ::
    Vec nodesNumber (Maybe AddressNumber) -> -- вектор свободных адресов
    LoadedNode portsNumber agentType ->
    LoadedNode portsNumber agentType ->
    ReduceRuleResult nodesNumber edgesNumber portsNumber agentType
```

Листинг 1: Класс для параметризации правил редукции

5. Тестирование стенда

Для оценки работоспособности и эффективности разработанного прототипа специализированного процессора на основе INTERACTION NETS было проведено исследование аппаратных характеристик в сравнении с ближайшими аналогами и тестирование работы прошитой платы.

5.1. Проверка работоспособности

Тестирование на уровне симуляторов (CLASH и Verilator) не даёт гарантии, что плата будет работать правильно хотя бы на проверенных примерах. Для тестирования на самом низком уровне вычислитель после редуцирования сети до нормального состояния отдаёт снимок памяти, что позволяет анализировать результат работы платы.

Чтобы передать данные был использован осциллограф Gowin Analyzer (GAO) — виртуальный инструмент, встроенный в Gowin IDE. Захваченные сигналы записываются и экспортируются в csv файл и разбираются с помощью библиотеки cassava⁵. После бинарные данные дешифруются с помощью CLASH, давая возможность анализировать снова высокоуровневое представление.

Тестирование проводилось на плате tang primer 25k на следующих λ -термах, переведённых в сеть.

- $(\lambda x.x)(\lambda x.x)$
- $(\lambda x.x)\varepsilon$
- $(\lambda x.\varepsilon x)(\lambda y.y)$

5.2. Анализ аппаратных характеристик

Сравнение с аналогами служит для проверки базовых предположений по поводу жизнеспособности системы и оценки её масштабируемости. Для анализа были выбраны следующие вычислители.

⁵GitHub библиотеки: <https://github.com/haskell-hvr/cassava>

- Heron [10] — вычислитель, основанный на редукции графов с открытым исходным кодом. Ближайший аналог разработанного решения. Также реализован на CLASH.
- Vortex [14] — GPU общего назначения с открытым исходным кодом. Проект также нацелен на реализацию массового параллелизма, следовательно, аппаратные затраты на одно ядро могут дать понимание пределов масштабирования.

Оба этих проекта делают акцент на многоядерности, что подходит анализа масштабируемости системы, имеют открытый исходный код и всё ещё находятся в рамках академической среды.

Для изучения были выбраны следующие метрики.

- Максимальная частота работы вычислителя. Важный критерий, во многом определяющий эффективность работы ядра.
- Количество занятых LUT-ов.
- Количество занятых регистров.

Данные по аппаратным затратам Heron были взяты из [10] (раздел «Hardware-Level Design»). Vortex же был синтезирован под плату Alveo U50, и данные были взяты из отчётов vivado. Данные для вычислителя на INTERACTION NETS были получены таким же образом. Все характеристики представлены для одного ядра.

Результаты приведены в сводной таблице 1. Анализируя данные, можно заметить, что разработанный вычислитель значительно уступает в максимальной тактовой частоте обоим аналогам, но занимает гораздо меньше аппаратных ресурсов.

5.3. Результаты

Из проведённого анализа можно сделать следующие выводы.

- Реализация вычислителя на основе INTERACTION NETS с использованием описанных технологий и подходов принципиально возможна.

Вычислитель	Макс. частота (МГц)	LUTs	Регистры
INTERACTION NETS-core	7.5	2900	351
Heron [10]	173	12257	782
Vortex [14]	300	23884	29494

Таблица 1: Сводная таблица использования аппаратных ресурсов разных вычислителей

- Разработанный вычислитель потребляет значительно меньше аппаратных ресурсов по сравнению с аналогами, что даёт надежду на хорошую масштабируемость.
- Максимальная частота значительно уступает аналогам. Это может объясняться плохой оптимизацией работы конвейера.

Однако стоит явно отметить, что оба аналога являются гораздо более «зрелыми» проектами. Разработанный вычислитель не содержит большого количества оптимизаций, которые могут значительно увеличить количество потребляемых ресурсов. Так что из проведённого анализа не следуют (и не могут следовать) выводы по поводу итоговой ресурсоёмкости вычислителя.

Заключение

По итогам годовой работы были выполнены следующие задачи.

1. Спроектирована общую структуру проекта.
 - Сформулированы чёткие требования к проекту
 - Продуман пайплайн: от написания программы на высокоуровневом языке до получения прошивки.
2. Спроектирована архитектуру ядра вычислителя.
 - Выделены этапы выполнения редукции.
 - Продуманы низкоуровневые особенности работы с INTERACTION NETS.
3. Реализован одноядерный прототип вычислителя.
 - Написаны все спроектированные компоненты.
 - Реализованы абстракции для поддержки параметризуемости.
 - Задokumentирован пайплайн для сборки и синтеза прошивки.
4. Проведено тестирование работоспособности вычислителя.

Исходный код находится в репозитории: <https://github.com/Lamagraph/interaction-nets-in-fpga/>. Имя коммитера: KubEF.

Дальнейшая работа

Проект планируется развивать, на данный момент был проделан только первый этап. В дальнейшем необходимо выполнить как минимум следующие задачи.

- Добавить eDSL для описания спецификации INTERACTION NETS.
- Реализовать поддержку многоядерности.
- Добавить поддержку HBM.

Список литературы

- [1] Chiu Matt, Herbordt Martin C. Molecular Dynamics Simulations on High-Performance Reconfigurable Computing Systems // [ACM Trans. Reconfigurable Technol. Syst.](#) — 2010. — nov. — Vol. 3, no. 4. — 37 p. — URL: <https://doi.org/10.1145/1862648.1862653>.
- [2] [CλaSH: Structural Descriptions of Synchronous Hardware Using Haskell](#) / Christiaan Baaij, Matthijs Kooijman, Jan Kuper et al. — 2010. — 09. — P. 714–721.
- [3] Corporaal Henk. Design of transport triggered architectures // Proceedings of 4th Great Lakes Symposium on VLSI / IEEE. — 1994. — P. 130–135.
- [4] Flexible video processing platform for 8K UHD TV. / Sukjin Kim, Young-Hwan Park, Jaehyun Kim et al. // Hot chips symposium. — 2015. — P. 1.
- [5] Besta Maciej, Stanojevic Dimitri, Licht Johannes De Fine et al. Graph Processing on FPGAs: Taxonomy, Survey, Challenges. — 2019. — [1903.06697](#).
- [6] [HW/SW Co-design Toolset for Customization of Exposed Datapath Processors](#) / Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, Heikki Berg // Computing Platforms for Software-Defined Radio / Ed. by Waqar Hussain, Jari Nurmi, Jouni Isoaho, Fabio Garzia. — Springer International Publishing, 2017. — P. 147–164. — ISBN: [978-3-319-49679-5](#). — URL: https://doi.org/10.1007/978-3-319-49679-5_8.
- [7] Hassan Abubakar, Mackie Ian, Sato Shinya. An Implementation Model for Interaction Nets // [Electronic Proceedings in Theoretical Computer Science](#). — 2015. — may. — Vol. 183. — P. 66–80. — URL: <http://dx.doi.org/10.4204/EPTCS.183.5>.
- [8] Lafont Yves. [Interaction nets](#) // Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Lan-

- guages. — POPL '90. — New York, NY, USA : Association for Computing Machinery, 1989. — P. 95–108. — URL: <https://doi.org/10.1145/96709.96718>.
- [9] Qing Wu Maksym Spiryagin Colin Cole, McSweeney Tim. Parallel computing in railway research // [International Journal of Rail Transportation](#). — 2020. — Vol. 8, no. 2. — P. 111–134. — URL: <https://doi.org/10.1080/23248378.2018.1553115>.
- [10] Ramsay Craig, Stewart Robert. [Heron: Modern Hardware Graph Reduction](#) // Proceedings of the 35th Symposium on Implementation and Application of Functional Languages. — IFL '23. — New York, NY, USA : Association for Computing Machinery, 2024. — 12 p. — URL: <https://doi.org/10.1145/3652561.3652564>.
- [11] Sparse Matrix-Vector Multiplication on a Reconfigurable Supercomputer with Application / David Dubois, Andrew Dubois, Thomas Boorman et al. // [ACM Trans. Reconfigurable Technol. Syst.](#) — 2010. — jan. — Vol. 3, no. 1. — 31 p. — URL: <https://doi.org/10.1145/1661438.1661440>.
- [12] A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications / Leibo Liu, Jianfeng Zhu, Zhaoshi Li et al. // [ACM Comput. Surv.](#) — 2019. — oct. — Vol. 52, no. 6. — 39 p. — URL: <https://doi.org/10.1145/3357375>.
- [13] Véstias Mário P. A survey of convolutional neural networks on edge with reconfigurable computing // [Algorithms](#). — 2019. — Vol. 12, no. 8. — P. 154.
- [14] Vortex: Extending the RISC-V ISA for GPGPU and 3D-graphics / Blaise Tine, Krishna Praveen Yalamarthy, Fares Elsabbagh, Kim Hye-soon // MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. — 2021. — P. 754–766.

- [15] Vranjković Vuk S, Struharik Rastislav JR, Novak Ladislav A. Reconfigurable hardware for machine learning applications // Journal of Circuits, Systems and Computers. — 2015. — Vol. 24, no. 05. — P. 1550064.