Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Разработка транслятора модельного функционального языка в Interaction Nets

Пономарев Николай Алексеевич

Отчёт по производственной практике
в форме «Решение»

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

Санкт-Петербург
2025

Оглавление

Введение **3**

1. Interaction Nets **4**

2. Обзор существующих решений **7**

3. Постановка задачи **9**

4. Общая архитектура проекта **10**

5. Подробности реализации **12**

6. План экспериментов **16**

Заключение **17**

Список литературы **18**

Введение

Искусственный интеллект и анализ графов — одни из наиболее привлекательных областей науки в данный момент [5, 6]. Многие алгоритмы, используемые в этих областях, основаны на линейной алгебре или могут быть переформулированы в её терминах, позволяя использовать развитую экосистему для работы с линейной алгеброй. Поскольку вычисления в линейной алгебре часто независимы друг от друга, разумно использовать возможности параллельного программирования для ускорения работы алгоритмов. А для больших объемов данных разумно использовать разреженную линейную алгебру.

К сожалению, распараллеливание разреженной линейной алгебры — сложная задача для традиционных архитектур вычислителей таких, как CPU или GPU, из-за нелокальных обращений к памяти и непредсказуемого количества «агентов» [8, 4, 17]. В настоящее время для решения этих проблем всё чаще применяют ускорители на специализированных архитектурах [1, 28, 10, 2, 25].

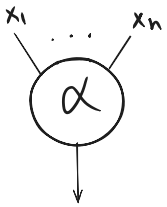
Interaction Nets — модель вычислений, которая была описана Yves Lafont в 1990 году. В этой модели программа представляется в виде графа, и, в силу свойств модели, вычисления происходят только локально и между конечным множеством вершин за шаг, поэтому в данной модели легко достигается параллельность.

Для Interaction Nets был написан не один интерпретатор, например [15, 20], однако попыток реализовать ускоритель на его основе не предпринималось, кроме того существующие интерпретаторы используют собственные языки программирования далёкие от распространённых. Поэтому в рамках проекта Lamagraph исследуются возможности по разработке параметризуемого многоядерного сопроцессора для разреженной линейной алгебры на основе Interaction Nets и ML-подобного функционального языка программирования для программирования сопроцессора.

1. Interaction Nets

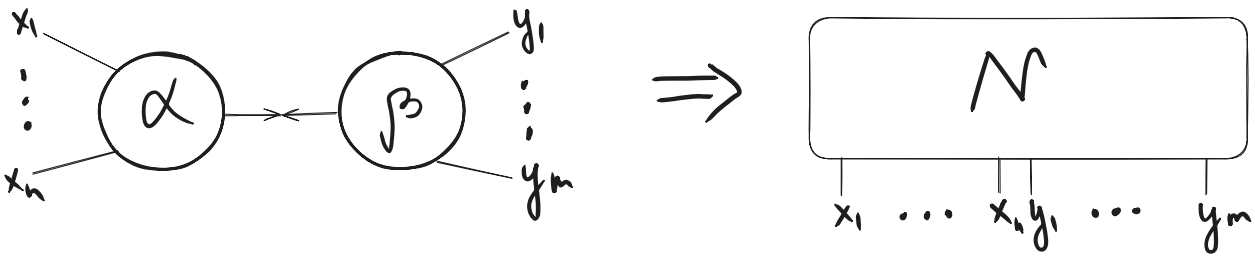
Данный раздел является кратким описанием системы Interaction Nets, более подробно про неё можно узнать в [11, 21, 19].

Описание системы. Interaction Nets представляет собой систему переписывания графов. Зафиксируем множество символов Σ, которым будем обозначать узлы графа. Для каждого символа зафиксируем его арность ar, которая будет означать количество *дополнительных* портов символа. Так, если символ $\alpha \in \Sigma$ имеет арность $\mathrm{ar}(\alpha) = n$, где $n \in \mathbb{N}$, то у символа имеется $n+1$ портов: n дополнительных и один выделенный — *главный*.



Узлы могут изображаться с помощью кругов, треугольников или прямоугольников. *Сеть*, построенная на Σ, является неориентированным графом с символами из Σ в его вершинах. Ребра соединяют порты вершин так, что в каждый порт приходит не более одного ребра. Порт не соединенный ни с одним ребром называется *свободным*, а множество таких портов называется *интерфейсом*.

Пара агентов $(\alpha, \beta) \in \Sigma \times \Sigma$, соединенных своими главными портами называется *активной парой* (редексом). Правило $((\alpha, \beta) \Longrightarrow N)$ заменяет активную пару (α, β) на сеть N. Для каждой пары агентов существует не более одного правила редукции, при этом в процессе редукции интерфейс сохраняется.



Одним из важных свойств системы является *свойство ромба*: порядок переписываний не важен и все последовательности переписывания имеют одну и ту же длину. Из этого следуют практически значимые вещи: преобразовывать граф можно в любом порядке, кроме того, это можно делать параллельно.

Выбор базиса агентов. Поскольку описание Interaction Nets не фиксирует набор агентов, пользователю предоставлены большие возможности по созданию собственных наборов и правил их редукции.

Так, например, в статье [12] Lafont в качестве примера использует набор агентов $\Sigma = \{\mathrm{Cons}, \mathrm{Nil}, \mathrm{Append}\}$ и правила редукции, соответствующие спискам в функциональных языках программирования. А в статье [11] обсуждается базис $\Sigma = \{\gamma, \delta, \varepsilon\}$, который по своим свойствам схож с базисом SKI в λ-исчислении.

Отдельно можно отметить наличие множества базисов для трансляции λ-исчисления в Interaction Nets: [13, 3, 7, 14, 27, 23].

Стратегия вычислений. В отличие от λ-исчисления, Interaction Nets не предполагает наличие нескольких стратегий редукции сама по себе. Тем не менее, поскольку в Interaction Nets возможно кодировать другие формальные системы, то стратегия редукции может проявится в Interaction Nets в различных формах записи агентов и правил их редукции. Так, например, в Interaction Nets можно закодировать как строгую [23], так и ленивую [24] стратегии для λ-исчисления.

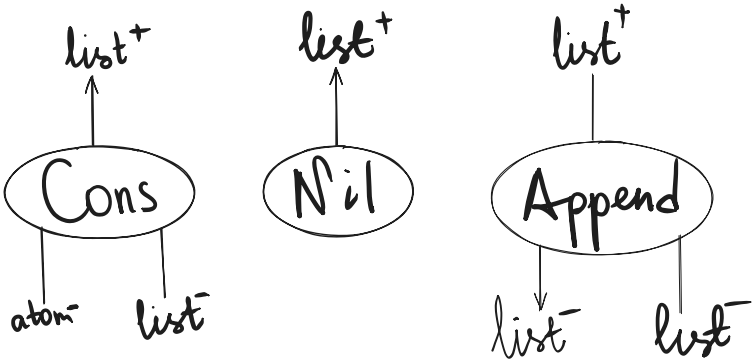
Типизация. Interaction Nets поддерживает типизацию достаточно интуитивным образом: мы будем помечать порт его *типом* и дополнительно *полярностью*. Полярность необходима для обозначения, какой порт является *входным*, а какой *выходным*.

Тогда сеть является правильно типизированной, если входы подключены к выходам одного типа. Правило редукции является правильно типизированным, если

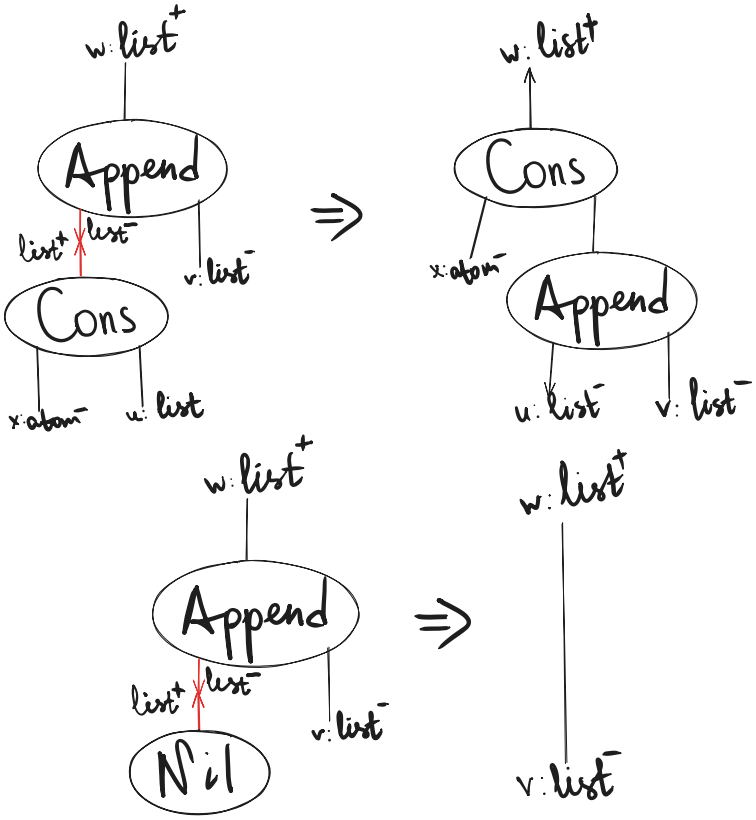
- сеть в левой части правила является правильно типизированной;

- сеть после редукции остаётся правильно типизированной.

Приведём пример из [12, раздел 2]. Пусть имеется множество типов: `atom`, `list`. Обозначим входные порты, как τ^-, а выходные, как τ^+. Тогда набор агентов $\Sigma = \{\text{Cons}, \text{Nil}, \text{Append}\}$ будет типизирован следующим образом:



А типизированные правила редукции будут выглядеть, как



2. Обзор существующих решений

Со времен публикации первых статей был разработан не один исполнитель Interaction Nets. Обзор на момент 2014 года можно найти в работе [21]. Мы же приведём обзор более новых работ в данной области.

Inpla и Train. Inpla[1] — интерпретатор одного из крупных учёных в области Shinya Sato, начатый в его PhD [21], и поддерживающий параллельное исполнение [15].

Inpla предполагает описание сетей на собственном языке программирования, который тем не менее достаточно сложен для использования. Транслятор Train[2] решает данную проблему, конвертируя код из функционального языка программирования в код для Inpla. Весь комплекс реализован на C.

HVM 1, 2, 3 и Bend. Семейство проектов от стартапа Higher Order Company[3]. HVM (Higher-order Virtual Machine) является средой исполнения Interaction Nets, эксплуатирующей параллельность, и существует в нескольких версиях, отличающихся языком реализации, стратегией вычислений и средой исполнения.

HVM1 Написана на Rust, с ленивой стратегией вычисления на CPU. Считается устаревшей.

HVM2 Написана на Rust, со строгой стратегией, поддерживает как CPU через кодогенерацию в C, так и GPU через генерацию в CUDA. На данный момент является стабильной версией.

HVM3 Написана на Haskell; поддерживает как ленивую, так и строгую стратегии вычисления на CPU. Является наследником HVM1 и HVM2 и создается, чтобы их заменить, находится в активной разработке.

[1]Репозиторий проекта: https://github.com/inpla/inpla/ (дата обращения: 15 февраля 2025 г.)

[2]Репозиторий проекта: https://github.com/inpla/train/ (дата обращения: 15 февраля 2025 г.)

[3]GitHub организация проекта: https://github.com/HigherOrderCO/ (дата обращения: 17 февраля 2025 г.)

HVM1 использовал собственный Haskell-подобный синтаксис. В HVM2 и HVM3 используется низкоуровневый функциональный язык. Для облегчения жизни пользователей, с ними предполагает использовать высокоуровневый язык Bend, который будет транслироваться в низкоуровневое представление.

lambda. lambda[4] — интерпретатор λ-исчисления, реализованный на JavaScript, поддерживающий четыре стратегии трансляции в Interaction Nets [20]. Принимает программы на собственном языке программирования, похожем на λ-исчисление.

interact. interact[5] — интерпретатор, написанный на Scala. Имеет свой язык программирования, похожий на OCaml и Python, где каждая функция становится агентом сети.

Выводы. Таким образом, несмотря на существование множества различных интерпретаторов, большинство из них используют свои собственные языки программирования, часто синтаксически далёкие от массовых, и не декларируют используемые наборы агентов. Кроме того, на данный момент попыток разработать ускоритель на основе Interaction Nets не предпринималось.

[4]Репозиторий проекта: https://github.com/codedot/lambda/ (дата обращения: 17 февраля 2025 г.)

[5]Репозиторий проекта: https://github.com/szeiger/interact/ (дата обращения: 18 февраля 2025 г.)

3. Постановка задачи

Целью работы является разработка транслятора модельного функционального языка в Interaction Nets. Для её выполнения были поставлены следующие задачи:

1. Реализовать интерпретатор модельного ML-подобного языка
   (a) Конкретный синтаксис языка
   (b) AST и синтаксический анализатор
   (c) Алгоритм вывода типов
   (d) Рассахаривание в обогащенное λ-исчисление
   (e) Интерпретатор обогащенного λ-исчисления
2. Реализовать транслятор из обогащенного λ-исчисления в Interaction Nets
   (a) Представление набора агентов и правил редукции
   (b) Сопоставление λ-термам соответствующих агентов
   (c) Готовые наборы агентов и правил редукции
3. Реализовать интерпретатор Interaction Nets
   (a) Последовательные редукции
   (b) Последовательное исполнение параллельных редукций
   (c) Сбор статистики с учётом способа редукции
4. Провести эксперименты с наборами инструкций

В рамках производственной практики планируется выполнение только первой задачи.

4. Общая архитектура проекта

Проект Lamagraph ставит перед собой цель изучить возможности по созданию специализированных ускорителей на основе Interaction Nets.

К проекту выдвинуты следующие требования.

- Возможность параметризовать компилятор и вычислительное ядро типами агентов сети и правилами их редукции.
- Возможность сбора статистики такой, как размер сети, количество редукций, время исполнения и другой.
- Возможность постановки сравнительных экспериментов.
- Использование единого стека технологий — гомогенность.
- Получение полнофункционального прототипа, содержащего все компоненты, важнее, чем детальная проработка какого-то отдельного компонента.
- Расширяемость и модифицируемость. Должна быть возможность вносить изменения в любые компоненты.

Крупномасштабная архитектура проекта изображена на рисунке 1 и состоит из трёх крупных блоков.

Compiler Транслятор ML-подобного языка программирования. Содержит в себе интерпретатор и генерирует промежуточное представление, пригодное к дальнейшей трансляции в Interaction Nets.

Hardware Генератор описания аппаратуры для ПЛИС, параметризуемый базисом агентов сети.

Middle Транслятор из промежуточного представления в байт-код, пригодный для исполнения на процессоре, сгенерированном в блоке Hardware.

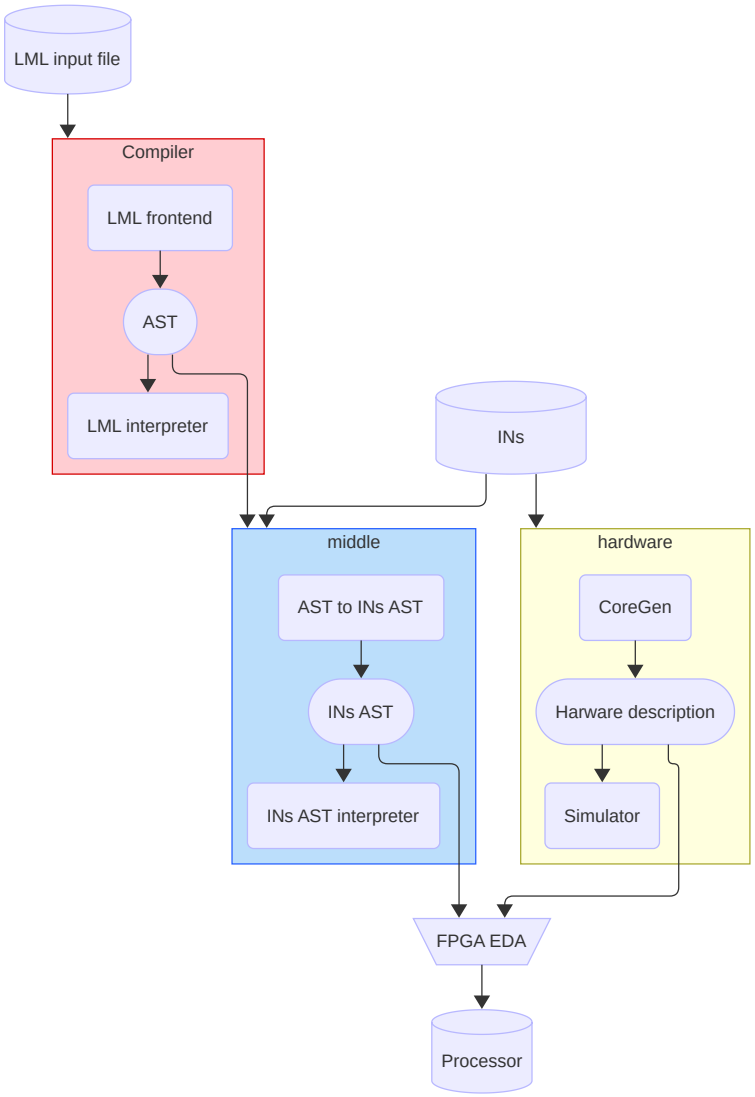


Рис. 1: Общая архитектура проекта Lamagraph.

Проект является командным. Блоки Compiler и Hardware достаточно независимы друг от друга. Блок Middle объединяет два других, поэтому его разработка должна быть скоординирована. В данной работе речь пойдёт о блоке Compiler.

5. Подробности реализации

В качестве языка реализации проекта был выбран Haskell. Предполагаемая архитектура изображена на рисунке 2.

Для сборки проекта и версионирования зависимостей используется Stack[6], каждая часть, описанная в разделе 4, является отдельным пакетом. Для тестирования используется фреймворк Tasty[7], по умолчанию все части компилятора покрываются golden[8] тестами.

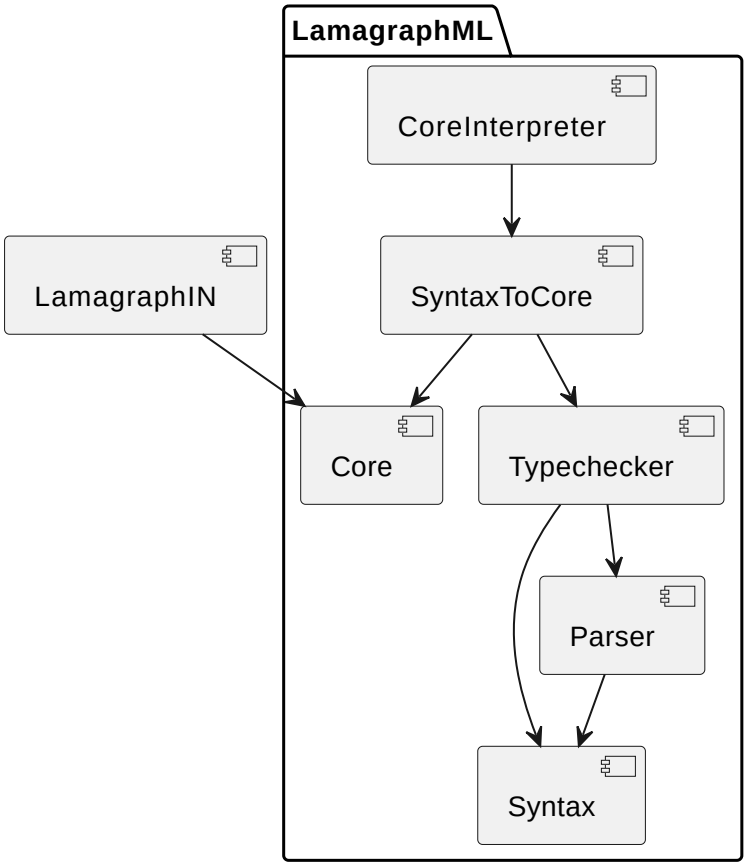


Рис. 2: Архитектура транслятора

[6]Сайт проекта: https://docs.haskellstack.org/en/stable/ (дата обращения 25 февраля 2025 г.)

[7]Описание проекта на Hackage: https://hackage.haskell.org/package/tasty/ (дата обращения 25 февраля 2025 г.)

[8]Описание golden тестов: https://ro-che.info/articles/2017-12-04-golden-tests (дата обращения 25 февраля 2025 г.)

Синтаксис языка. Синтаксис языка основан на OCaml. Однако упрощен для простоты реализации[9]. Так, например в языке остались стандартные для функциональных языков конструкции, такие как сопоставление с образцом, рекурсивные и взаимнорекурсивные функции, а также алгебраические типы данных. Однако в отличие от OCaml отсутствует поддержка классов и функторов, а система модулей максимально упрощена и напоминает систему модулей в F#.

Для представления AST (дерева абстрактного синтаксиса) используется паттерн Trees That Grow (TTG) [22]. Он позволяет с помощью механизма type families [26] гибко параметризовать дерево необходимыми аннотациями, более того аннотации могут различаться в разных узлах дерева, тем самым поддерживая безопасность кода.

Парсер. Для синтаксического анализа используется связка лексера Alex и парсер-генератора Happy, которые являются аналогами flex и bison, написанными на Haskell.

На данном этапе аннотации с помощью TTG не используются.

Для тестирования лексера используются модульные тесты. Для парсера используется property-based тестирование с использованием библиотеки Hedgehog[10]. Данный метод основан на том, что синтаксический анализ в AST и печать AST должны давать тождественное отображение при композиции.

Вывод типов. Поскольку язык ML-подобный, используется система типов Хиндли-Милнера [9, 16].

На данном этапе в аннотациях TTG сохраняется тип каждого узла дерева.

Промежуточное представление. AST получаемый после парсинга и вывода типов получается достаточно сложным — в нём большое

[9]С полной грамматикой можно ознакомиться в репозитории проекта: https://github.com/Lamagraph/interaction-nets-in-fpga/blob/main/lamagraph-compiler/src/Lamagraph/Compiler/Syntax.hs (дата обращения 25 февраля 2025 г.)

[10]Репозиторий проекта: https://github.com/hedgehogqa/haskell-hedgehog/ (дата обращения: 19 февраля 2025 г.)

```haskell
data Literal = LitInt Int | LitChar Char | LitString Text

type DataCon = Name

newtype Var = Id Name

data Expr b
  = Var b
  | Lit Literal
  | App (Expr b) (Expr b)
  | Lam b (Expr b)
  | Let (Bind b) (Expr b)
  | Match (Expr b) b (NonEmpty (MatchAlt b))
  | Tuple (Expr b) (NonEmpty (Expr b))

type MatchAlt b = (AltCon, [b], Expr b)

data AltCon = DataAlt DataCon | LitAlt Literal | TupleAlt | DEFAULT

data Bind b = NonRec b (Expr b) | Rec (NonEmpty (b, Expr b))

type CoreExpr = Expr Var
type CoreMatchAlt = MatchAlt Var
type CoreBind = Bind Var
```

Листинг 1: Представление Core в алгебраических типах данных.

количество различных узлов с аннотациями. Для упрощения дальнейшей работы используется промежуточное представление на основе GHC Core[11], также часто называемое обогащённым λ-исчислением[12]. Его описание представлено на листинге 1.

Важными отличиями от GHC Core являются наличие выделенного конструктора для пар, а также отсутствие типизации.

Наличие выделенного конструктора для пар обусловлено различием между ML и Haskell — в первом пары тоже выделены и могут быть какой угодно арности, во втором же пара является синтаксическим сахаром для алгебраического типа-суммы и ограничена арностью 63.

От типизации пришлось отказаться в угоду простоты реализации, а также в виду отсутствия оптимизаций со стороны компилятора, где наличие типов упрощает и делает более безопасным их применение.

[11]Подробнее можно прочитать по ссылке: https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/core-syn-type (дата обращения: 19 февраля 2025 г.)

[12]Подробнее узнать про способы обогащения λ-исчисления можно в [18, раздел 3.2]

Трансляция высокоуровневого AST в промежуточное представление. Алгоритм трансляции достаточно прямолинеен, наиболее сложной частью является трансляция сложных шаблонов в левой части let-связывания и замена вложенных шаблонов на вложенные match-выражения.

6. План экспериментов

Как обсуждалось в разделе 1, существует не один способ трансляции λ-исчисления в Interaction Nets — это и будет основой для постановки экспериментов.

Мы планируем измерять для разных наборов агентов и правил редукции следующие метрики:

- размер программы в терминах количества агентов,
- количество редукций.

Кроме того, поскольку Interaction Nets — система, допускающая параллельность нас интересуют такие метрики, как

- максимальное количество одновременных (параллельных) редукций;
- количество шагов до достижения нормальной формы при неограниченном количестве параллельных редукций.

Поскольку Lamagraph исследует ускорение линейной алгебры, то и проводить эксперименты планируется на операциях линейной алгебры, например умножении разреженных матриц в формате дерева квадрантов.

Заключение

В рамках данной производственной практики были достигнуты следующие результаты.

1. Реализована часть интерпретатора модельного ML-подобного языка
   (a) Конкретный синтаксис языка
   (b) AST и синтаксический анализатор
   (c) Алгоритм вывода типов
   (d) Рассахаривание в обогащенное λ-исчисление

Остальные задачи планируется выполнить в течение весеннего семестра.

Исходный код находится в репозитории: https://github.com/Lamagraph/interaction-nets-in-fpga/. Имя коммитера: `WoWaster`.

Список литературы

[1] Accelerating Reduction and Scan Using Tensor Core Units / Abdul Dakkak, Cheng Li, Isaac Gelado et al. // Proceedings of the ACM International Conference on Supercomputing. — P. 46–57. — arXiv : cs/1811.09736.

[2] Akkad Ghattas, Mansour Ali, Inaty Elie. Embedded Deep Learning Accelerators: A Survey on Recent Advances. — Vol. 5, no. 5. — P. 1954–1972. — URL: https://ieeexplore.ieee.org/document/10239336/?arnumber=10239336 (дата обращения: 2025-02-22).

[3] Asperti Andrea, Giovannetti Cecilia, Naletto Andrea. The Bologna Optimal Higher-Order Machine. — Vol. 6, no. 6. — P. 763–810. — URL: https://www.cambridge.org/core/product/identifier/S0956796800001994/type/journal_article (дата обращения: 2025-02-24).

[4] Dedicated Hardware Accelerators for Processing of Sparse Matrices and Vectors: A Survey / Valentin Isaac–Chassande, Adrian Evans, Yves Durand, Frédéric Rousseau. — Vol. 21, no. 2. — P. 1–26. — URL: https://dl.acm.org/doi/10.1145/3640542 (дата обращения: 2025-02-21).

[5] Economic Potential of Generative AI / Michael Chui, Eric Hazan, Roger Roberts et al.

[6] García Roberto, Angles Renzo. Path Querying in Graph Databases: A Systematic Mapping Study. — Vol. 12. — P. 33154–33172. — URL: https://ieeexplore.ieee.org/document/10456906 (дата обращения: 2024-12-18).

[7] Gonthier Georges, Abadi Martín, Lévy Jean-Jacques. The Geometry of Optimal Lambda Reduction // Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '92. — Association for Computing Machinery. — P. 15–26. —

URL: https://dl.acm.org/doi/10.1145/143165.143172 (дата обращения: 2025-02-23).

[8] High-Performance Sparse Linear Algebra on HBM-Equipped FPGAs Using HLS: A Case Study on SpMV / Yixiao Du, Yuwei Hu, Zhongchun Zhou, Zhiru Zhang // Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. — ACM. — P. 54–64. — URL: https://dl.acm.org/doi/10.1145/3490422.3502368 (дата обращения: 2025-02-21).

[9] Hindley R. The Principal Type-Scheme of an Object in Combinatory Logic. — Vol. 146. — P. 29–60. — jstor : 1995158.

[10] In-Datacenter Performance Analysis of a Tensor Processing Unit / Norman P. Jouppi, Cliff Young, Nishant Patil et al. // Proceedings of the 44th Annual International Symposium on Computer Architecture. — ISCA '17. — Association for Computing Machinery. — P. 1–12. — URL: https://dl.acm.org/doi/10.1145/3079856.3080246 (дата обращения: 2025-02-21).

[11] Lafont Yves. Interaction Combinators. — Vol. 137, no. 1. — P. 69–101. — URL: https://www.sciencedirect.com/science/article/pii/S0890540197926432 (дата обращения: 2024-12-17).

[12] Lafont Yves. Interaction Nets // Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '90. — Association for Computing Machinery. — P. 95–108. — URL: https://dl.acm.org/doi/10.1145/96709.96718 (дата обращения: 2024-12-16).

[13] Lamping John. An Algorithm for Optimal Lambda Calculus Reduction // Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '90. — ACM Press. — P. 16–30. — URL: http://portal.acm.org/citation.cfm?doid=96709.96711 (дата обращения: 2025-02-24).

[14] Mackie Ian. An Interaction Net Implementation of Closed Reduction // Implementation and Application of Functional Languages / Ed. by Sven-Bodo Scholz, Olaf Chitil. — Springer Berlin Heidelberg. — Vol. 5836. — P. 43–59. — URL: http://link.springer.com/10.1007/978-3-642-24452-0_3 (дата обращения: 2025-02-22).

[15] Mackie Ian, Sato Shinya. Parallel Evaluation of Interaction Nets: Case Studies and Experiments. — Vol. 73. — URL: https://eceasst.org/index.php/eceasst/article/view/2205 (дата обращения: 2024-12-17).

[16] Milner Robin. A Theory of Type Polymorphism in Programming. — Vol. 17, no. 3. — P. 348–375. — URL: https://www.sciencedirect.com/science/article/pii/0022000078900144 (дата обращения: 2024-12-17).

[17] Mohammed Thaha, Mehmood Rashid. Performance Enhancement Strategies for Sparse Matrix-Vector Multiplication (SpMV) and Iterative Linear Solvers. — arXiv : cs/2212.07490.

[18] Peyton Jones Simon L. The Implementation of Functional Programming Languages. — Prentice Hall Internaltional (UK) Ltd. — URL: https://www.microsoft.com/en-us/research/publication/the-implementation-of-functional-programming-languages-2/.

[19] Salikhmetov Anton. Interaction Nets in Russian. — arXiv : cs/1304.1309.

[20] Salikhmetov Anton. Token-Passing Optimal Reduction with Embedded Read-back. — Vol. 225. — P. 45–54. — arXiv : cs/1609.03644.

[21] Sato Shinya. Design and Implementation of a Low-Level Language for Interaction Nets : thesis / Shinya Sato. — URL: https://sussex.figshare.com/articles/thesis/Design_and_implementation_of_a_low-level_language_for_interaction_nets/23417312/1 (дата обращения: 2025-02-14).

[22] Shayan Najd, Simon Peyton Jones. Trees That Grow. — URL: https://lib.jucs.org/article/22912 (дата обращения: 2024-12-17).

[23] Sinot François-Régis. Call-by-Name and Call-by-Value as Token-Passing Interaction Nets // Typed Lambda Calculi and Applications / Ed. by Paweł Urzyczyn. — Springer Berlin Heidelberg. — Vol. 3461. — P. 386–400. — URL: http://link.springer.com/10.1007/11417170_28 (дата обращения: 2025-02-24).

[24] Sinot François-Régis. Token-Passing Nets: Call-by-Need for Free. — Vol. 135, no. 3. — P. 129–139. — URL: https://www.sciencedirect.com/science/article/pii/S1571066106000934 (дата обращения: 2025-02-22).

[25] Silvano Cristina, Ielmini Daniele, Ferrandi Fabrizio et al. A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. — arXiv : cs/2306.15552.

[26] Type Checking with Open Type Functions / Tom Schrijvers, Simon Peyton Jones, Manuel Chakravarty, Martin Sulzmann. — URL: https://www.researchgate.net/publication/221241290_Type_checking_with_open_type_functions.

[27] Vincent van Oostrom, Kees-Jan van de Looij, Marijn Zwitserlood. Lambdascope Another Optimal Implementation of the Lambda-Calculus.

[28] Zhu Yuhao, Mattina Matthew, Whatmough Paul. Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective. — arXiv : cs/1801.06274.