На правах рукописи

Лось Денис Андреевич

Исследование и оптимизация технологий программирования и методов анализа для мелкозернистого параллелизма задач на современных многоядерных процессорах

Специальность 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание учёной степени
кандидата технических наук

Долгопрудный — 2025

Работа выполнена в федеральном государственном автономном образовательном учреждении высшего образования «Московский физико-технический институт (национальный исследовательский университет)» (МФТИ, Физтех).

Научный руководитель: доктор технических наук, профессор
Плоткин Арнольд Леонидович

Ведущая организация: **Публичное акционерное общество «Институт электронных управляющих машин им. И.С. Брука»**

Защита состоится DD mmmmmmmm YYYY г. в XX часов на заседании диссертационного совета Д 123.456.78, созданного на базе федерального государственного автономного образовательного учреждения высшего образования «Московский физико-технический институт (национальный исследовательский университет)» (МФТИ, Физтех)

по адресу: 141701, Московская область, г. Долгопрудный, Институтский переулок, д. 9

С диссертацией можно ознакомиться в библиотеке МФТИ, Физтех и на сайте организации https://mipt.ru

Автореферат разослан DD mmmmmmmm2026 года.

Ученый секретарь
диссертационного совета Фамилия Имя Отчество

Общая характеристика работы

Актуальность темы. Крупнейшие технологические компании вкладывают значительные ресурсы в оптимизацию сервисов, критичных к задержке (latency-critical). Это оправдано: такие системы ежедневно обслуживают запросы более миллиарда пользователей. Например, инфраструктура биржи NASDAQ обрабатывает свыше миллиона сообщений в секунду.

Стабильная и предсказуемая задержка важна не только для качества пользовательского опыта, но и для безопасного функционирования систем — особенно с развитием беспилотных автомобилей и хирургических роботов.

Жёсткие требования по задержке предъявляются не только в серверных и облачных средах, но и на клиентских устройствах. В 2025 году около 3.6 млрд человек активно играют в видеоигры, из них 1.5 млрд — в Азии. Значительная часть геймплея приходится на мобильные устройства: к середине 2025 года в мире насчитывается более 7 млрд смартфонов в обращении, и их производительность ежегодно растёт, уже сопоставима с настольными системами трёх-пяти поколений назад.

Несмотря на значительные вложения в оптимизацию систем, критичных к задержке, анализ показывает: из-за простоев при исполнении в ряде случаев может быть в среднем задействовано лишь 40% исполнительных устройств для потоков критичных к задержке приложений.

Чтобы повысить утилизацию аппаратных ресурсов, современные высокопроизводительные процессоры используют одновременную многопоточность (Simultaneous Multithreading, SMT), позволяющую на каждом такте отправлять на исполнение на функциональные устройства процессорного ядра инструкции с нескольких потоков исполнения. Параллелизм на уровне инструкций повышает утилизацию ресурсов процессорного ядра, а параллелизм на уровне потоков — общую пропускную способность системы.

Однако технология SMT может снижать однопоточную производительность (single-thread performance), поэтому для ключевых потоков в критичных к задержке приложениях её чаще всего отключают.

В этой работе исследуется применение технологии SMT для повышения производительности ключевых потоков в критичных к задержке приложениях.

Критичные к задержке приложения оптимизируются в том числе за счёт их параллелизации. Крупнозернистые параллельные задачи, как правило, видны и выделяются на этапе дизайна алгоритмов и разработки приложения. Однако мелкозернистые задачи в большинстве случаев остаются невыделенными и неиспользованными при параллелизации. Под

мелкозернистыми задачами в данной работе понимаются задачи, которые выполняют от 500 до 3000 инструкций, что для процессора с частотой 3 ГГц, исполняющего для данного потока в среднем одну инструкцию за такт, соответствует гранулярности от 160 до 1000 наносекунд.

Целью данной работы является исследование и оптимизация технологий мелкозернистой параллелизации на SMT-ядрах современных процессоров для улучшения производительности критичных к задержке приложений.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Исследовать эффективность использования современных технологий параллельного программирования для мелкозернистой параллелизации на SMT-ядрах.
2. Исследовать причины и подходы к минимизации задержек при отправке задач на исполнение в современных технологиях параллельного программирования.
3. Разработать специализированную систему параллельного программирования для мелкозернистого параллелизма задач на SMT-ядрах.
4. Исследовать и разработать полуавтоматическую систему для мелкозернистой параллелизации критичных к задержке приложений промышленного уровня на SMT-ядрах.

Тема и содержание диссертационной работы соответствует паспорту научной специальности 2.3.5. Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей, в частности, пунктам:

п. 1 – Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования.

п. 3 – Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем.

п. 4 – Интеллектуальные системы машинного обучения, управления базами данных и знаний, инструментальные средства разработки цифровых продуктов.

п. 8 – Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования.

Научная новизна:

1. Впервые проведено исследование эффективности использования современных технологий параллельного программирования для мелкозернистой параллелизации на SMT-ядрах.

2. Впервые показана возможность специализированной системы параллельного программирования достичь значительного прироста производительности по сравнению с существующими технологиями параллельного программирования общего назначения при мелкозернистой параллелизации на SMT-ядрах.
3. Впервые представлена полуавтоматическая система параллелизации последовательных программ на основе LLM-агента с использованием процедурного промптинга с инструментальной поддержкой (tool-augmented procedural prompting), в которой встроенная в агента LLM используется для выявления потенциального параллелизма и синтеза параллелизующих преобразований кода, а в качестве инструментов агента могут подключаться такие средства, как средства динамического профилирования и анализа зависимостей, средства обнаружения горячих участков кода (hot spots), а также симулятор производительности SMT-ядер процессора.
4. Впервые полуавтоматическая система параллелизации последовательных программ, использующая LLM, интегрирована в редактор кода с искусственным интеллектом, чтобы улучшить параллелизацию программ за счёт семантической индексации их файлов исходного кода.

Практическая значимость Влияние результатов диссертации заключается в том, что разработанные решения, такие как специализированная система параллельного программирования Relic для мелкозернистого параллелизма задач на SMT-ядрах и полуавтоматическая система Aira для мелкозернистой параллелизации критичных к задержке приложений промышленного уровня на SMT-ядрах, были использованы в исследовательских проектах компании «Huawei».

Результаты данной работы внедрены в кафедральный курс «Микроархитектура современных микропроцессоров» кафедры микропроцессорных технологий в интеллектуальных системах управления МФТИ.

Теоретическая значимость работы заключается в проведении исследования эффективности современных технологий параллельного программирования на мультипроцессорах с общей памятью для мелкозернистой параллелизации на SMT-ядрах процессора; разработке специализированной системы параллельного программирования для мелкозернистого параллелизма задач на SMT-ядер, а также разработке полуавтоматической системы параллелизации последовательных программ на основе LLM-агента и подключении к нему дополнительных средств анализа в качестве инструментов агента; предложении

использовать семантическую индексацию файлов исходного кода анализируемых приложений для улучшения параллелизации программ.

Основные положения, выносимые на защиту:

1. Специализированная система параллельного программирования, которая позволяет достичь прироста производительности до 20% по сравнению с технологиями параллельного программирования общего назначения, такими как LLVM OpenMP и OpenCilk, при мелкозернистой параллелизации на SMT-ядрах.
2. Метод использования семантической индексации файлов исходного кода программы для улучшения точности выявления потенциального параллелизма с помощью больших языковых моделей в системах параллелизации последовательных программ.
3. Метод использования динамического анализа зависимостей для улучшения точности выявления потенциального параллелизма с помощью больших языковых моделей в системах параллелизации последовательных программ.
4. Метод определения целесообразности параллелизации на SMT-ядрах с помощью симулятора производительности процессора по трассам исполнения в системах параллелизации последовательных программ.

Достоверность. Достоверность и обоснованность результатов и выводов диссертации подтверждаются проведёнными экспериментами, методология которых детально описывается в диссертационной работе. Достоверность также подтверждается публикациями по материалам диссертации, в том числе, в рецензируемых научных изданиях. Теоретическую и методологическую основу проведённых разработок и исследований составили труды зарубежных авторов в области технологий параллельного программирования, разработки компиляторов и машинного обучения.

Апробация работы. Результаты диссертационной работы докладывались на:

1. 66-й Всероссийской научной конференции «Московского физико-технического института (национального исследовательского университета)», Москва, апрель 2024 г.

Личный вклад. Основные результаты диссертационного исследования получены лично автором. Постановка задач и анализ полученных результатов осуществлялись непосредственно автором. В совместных работах вклад автора заключался в техническом руководстве всем процессом работы, разработке алгоритмов, проведении измерений и анализе результатов.

Публикации. Основные результаты по теме диссертации изложены в 3 печатных изданиях, 2 из которых изданы в журналах, рекомендованных ВАК, 1 — в тезисах докладов.

Содержание работы

Во **введении** обосновывается актуальность исследований и разработок, проводимых в данной диссертационной работе, формулируется цель работы и определяются задачи, которые необходимо выполнить для её достижения. Кроме того, излагаются научная новизна и практическая значимость представленной диссертационной работы.

Первая глава посвящена обзору исследований применения технологии одновременной многопоточности при параллелизации программ, реализации метода вспомогательного потока (helper threading) и спекулятивной параллелизации. Кроме того, в первой главе проводится исследование эффективности современных технологий параллельного программирования при мелкозернистой параллелизации на SMT-ядрах.

В **параграфе 1.1** рассматривается устройство современных высокопроизводительных процессоров, а **параграф 1.2** посвящён рассмотрению технологий многопоточности на ядрах процессора.

В **параграфе 1.3** проводится обзор исследований применения технологии одновременной многопоточности при параллелизации программ. Рассматриваются существующие работы по исследованию применения SMT-технологии совместно с различными технологиями параллельного программирования, например реализациями OpenMP, Intel oneAPI Threading Building Blocks (oneTBB), OpenCilk, Taskflow. Обзор показал, что существующие технологии параллельного программирования не имеют специализированных интерфейсов, позволяющих направлять задачи именно на SMT-ядра. Кроме того, в предыдущих работах не проводился анализ эффективности использования SMT-технологии при мелкозернистой параллелизации в сочетании с существующими технологиями параллельного программирования.

В **параграфе 1.4** описывается исследование эффективности современных технологий параллельного программирования при мелкозернистой параллелизации на SMT-ядрах.

Исследование проводилось на разработанных мелкозернистых графовых бенчмарках и бенчмарке обработки документа в формате JSON. На рисунке 1 показан прирост производительности по сравнению с последовательным исполнением бенчмарка при использовании различных технологий параллельного программирования и при различных гранулярностях входного графа для бенчмарка алгоритма поиска в ширину (Breadth-First Search, BFS). Исследование было проведено на вычислительной машине с процессором Intel Core i7-12700, поддерживающим технологию Hyper-Threading (реализацию SMT-технологии компании Intel).

Результаты исследования показали, что для большинства рассмотренных бенчмарков при гранулярности задач до 1000 наносекунд

происходит падение производительности на 20% и более при использовании технологий параллельного программирования общего назначения. Так, например, как показано на рисунке 1, при обработке графа Кронекера с 64 вершинами изменение производительности при использовании LLVM OpenMP, OpenCilk, Taskflow и oneTBB составило соответственно -14%, -10%, -67%, -56%.

В **параграфе 1.5** представлен обзор исследований применения SMT-технологии при реализации метода вспомогательного потока (helper threading), а в **параграфе 1.6** — при использовании спекулятивного параллелизма. Обзор показал, что, несмотря на большое количество предложенных аппаратных оптимизаций для повышения эффективности исполнения мелкозернистых задач на SMT-ядрах, значительная часть этих оптимизаций не нашла отражения в коммерческих процессорах.

Исследование, описанное в первой главе, показало необходимость изучить причины задержек и подходы к их минимизации при отправке задач на исполнение в существующих технологиях параллельного программирования, а также разработать решение, позволяющее эффективно отправлять мелкозернистые задачи на исполнение на SMT-ядрах.

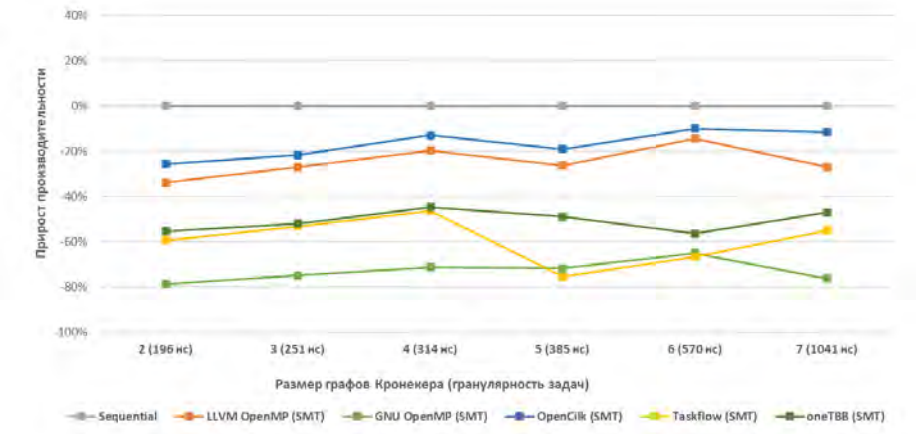

Рисунок 1 — Прирост производительности при мелкозернистой параллелизации на SMT-ядре с использованием различных технологий параллельного программирования для бенчмарка BFS

Вторая глава посвящена разработке специализированной системы параллельного программирования Relic для мелкозернистой параллелизации на SMT-ядрах.

Параграф 2.1 посвящён исследованию причин задержек и подходов к их минимизации при отправке задач на исполнение в системах параллельного программирования.

В **параграфе 2.2** описывается общее устройство разработанной системы параллельного программирования Relic.

Система параллельного программирования Relic призвана дополнить существующие технологии параллельного программирования общего назначения (например, LLVM OpenMP), чтобы обеспечить более эффективную отправку на исполнение мелкозернистых задач на SMT-ядрах. Крупнозернистая параллелизация и отправка задач на исполнение на различные физические ядра должна осуществляться технологиями параллельного программирования общего назначения, тогда как мелкозернистая параллелизация на одном физическом SMT-ядре должна осуществляться с помощью специализированной системы Relic.

Поскольку большинство коммерческих процессоров реализуют технологию одновременной многопоточности с двумя логическими потоками, в системе Relic было принято поддержать только такой вариант. Поток, который исполняется на SMT-ядре и создаётся самим приложением или технологией параллельного программирования общего назначения, называется главным (main) потоком, тогда как поток, создаваемый системой Relic для исполнения мелкозернистых задач на SMT-ядре, называется вспомогательным (assistant) потоком. Поскольку приложение может быть многопоточным, главных и вспомогательных потоков может быть несколько. Вспомогательные потоки привязаны к конкретным физическим SMT-ядрам.

Иллюстрация практического применения технологий параллельного программирования общего назначения и системы Relic представлена на рисунке 2.

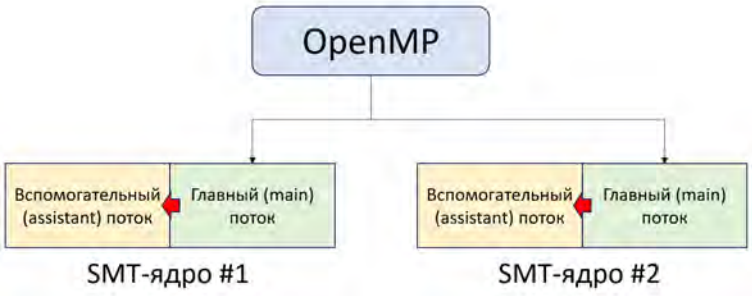

Рисунок 2 — Иллюстрация использования технологии OpenMP совместно со специализированной системой параллельного программирования Relic

Параграф 2.3 посвящён описанию алгоритма отправки задач на исполнение в системе Relic.

В системе Relic отправка задач на исполнение возможна лишь со стороны главного потока. Данное ограничение приводит к паттерну взаимодействия между потоками, в котором существует один производитель (producer) и один потребитель (consumer). Такой паттерн взаимодействия широко известен и называется Single-Producer Single-Consumer (SPSC). Для построения алгоритма отправки задач на исполнение в системе Relic использовались ранее известные механизмы, такие как безблокировочная (lock-free) SPSC-очередь и активное ожидание в цикле. Новым практическим научным результатом, полученным при разработке системы Relic, является то, что эта специализированная система параллельного программирования позволяет получить прирост производительности до 20% при мелкозернистой параллелизации на SMT-ядрах по сравнению с системами параллельного программирования общего назначения. Это открывает возможность оптимизировать критичные к задержке промышленные приложения за счёт ускорения ранее не параллелизованных участков кода.

Было проведено исследование эффективности применения системы Relic для мелкозернистых бенчмарков, рассмотренных в главе 1. Результаты замеров производительности при использовании системы Relic для бенчмарка BFS представлены на рисунке 3.

При входных графах Кронекера с 16 вершинами, как показано на рисунке 4, все рассмотренные бенчмарки были параллелизованы с использованием системы Relic с положительным приростом производительности.

При входном графе Кронекера с 16 вершинами улучшение производительности при использовании системы Relic составило 20.4% и 20.3% по сравнению с OpenCilk и LLVM OpenMP соответственно, как показано на рисунке 5. На рисунке 5 показаны среднегеометрические (geomean) приросты производительности для различных систем параллельного программирования. Среднегеометрические приросты производительности вычисляются без учёта негативных случаев, так как при практическом применении для оптимизации промышленных приложений случаи ухудшения производительности будут обнаружены и отброшены; то есть будут использоваться последовательные реализации.

Кроме того, было проведено исследование производительности системы Relic и технологий OpenCilk и LLVM OpenMP при использовании SMT-ядра и двух разных физических ядер.

Было показано, что для всех исследуемых бенчмарков существуют множества входных данных, для которых использование системы Relic на SMT-ядре даёт лучший (и при этом положительный) прирост производительности по сравнению с использованием системы Relic на

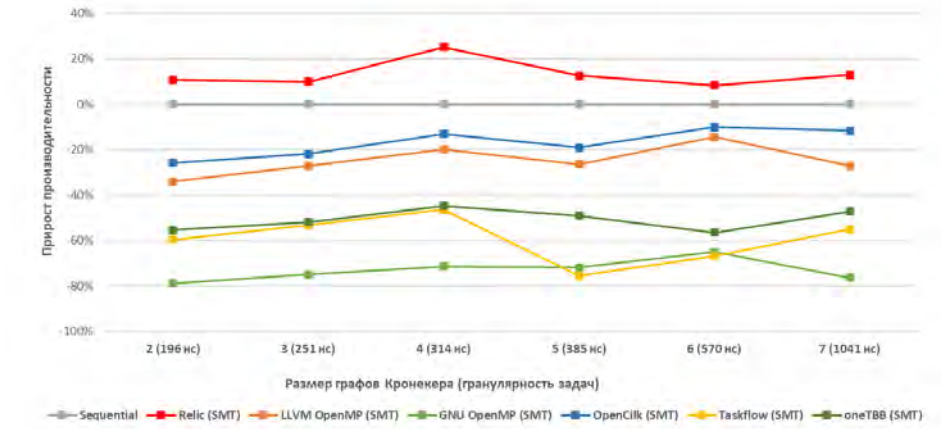

Рисунок 3 — Прирост производительности при мелкозернистой параллелизации на SMT-ядре с системой параллельного программирования Relic для бенчмарка BFS

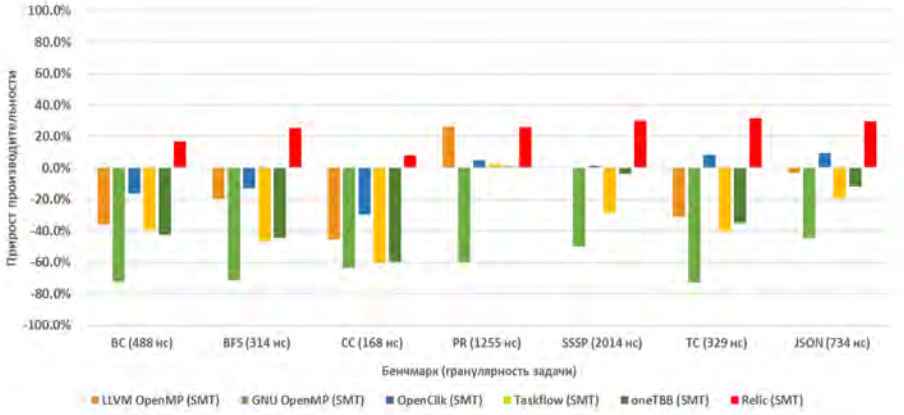

Рисунок 4 — Приросты производительности при мелкозернистой параллелизации на SMT-ядре с использованием различных технологий параллельного программирования для графовых бенчмарков с входным графом Кронекера с 16 вершинами и бенчмарка JSON

двух разных физических ядрах, а также по сравнению с использованием технологий OpenCilk и LLVM OpenMP на одном SMT-ядре или на разных физических ядрах. На рисунках 6 и 7 для бенчмарков BFS и CC такие области выделены красным цветом. Использование SMT-ядра обозначено как «SMT» на рисунках, а использование двух

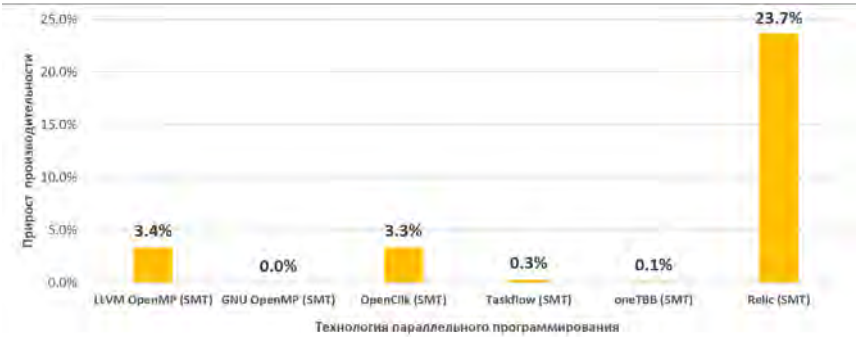

Рисунок 5 — Среднегеометрические приросты производительности (без негативных случаев) при мелкозернистой параллелизации на SMT-ядре с использованием различных технологий параллельного программирования для графовых бенчмарков с входным графом Кронекера с 16 вершинами и бенчмарка JSON

разных физических ядер — как «SMP». При этом для 4 из 6 графовых бенчмарков (BC, BFS, CC и SSSP) положительный прирост производительности в этих областях возможен лишь при использовании системы параллельного программирования Relic. Кроме того, для данных бенчмарков для большинства входных графов Кронекера из указанных областей положительный прирост производительности достигается только при использовании системы Relic на SMT-ядрах.

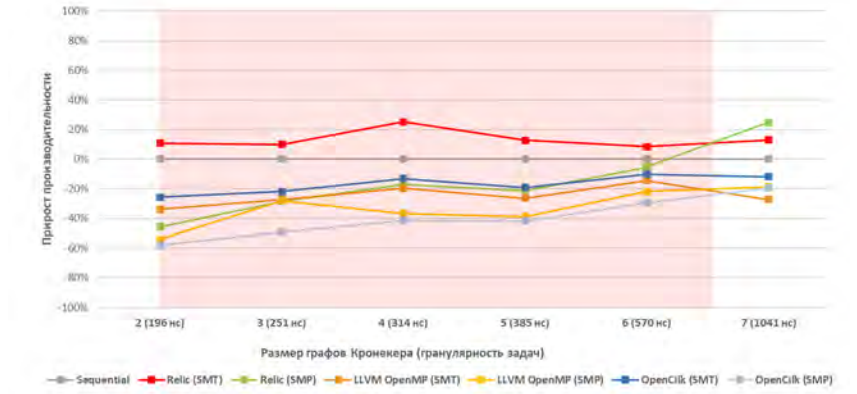

Рисунок 6 — Прирост производительности при мелкозернистой параллелизации на SMT-ядре и на двух различных физических ядрах с использованием различных технологий параллельного программирования для бенчмарка BFS

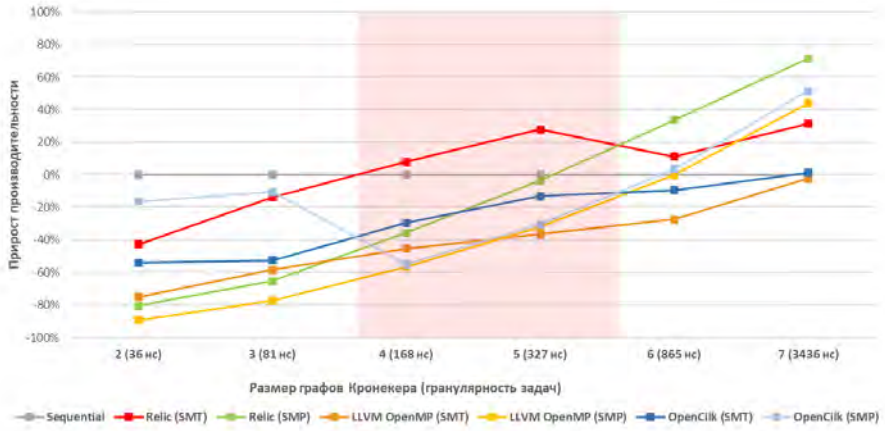

Рисунок 7 — Прирост производительности при мелкозернистой параллелизации на SMT-ядре и на двух различных физических ядрах с использованием различных технологий параллельного программирования для бенчмарка CC

В **параграфе 2.4** рассматривается использование функций-подсказок в системе параллельного программирования Relic для предпробуждения потоков операционной системы.

Третья глава посвящена обзору и исследованию существующих средств параллелизации последовательных программ.

В **параграфе 3.1** рассматриваются основные принципы параллелизации программ, приводятся определения зависимостей по данным и по управлению, обсуждаются различия между статическими и динамическими методами анализа зависимостей, а также рассматриваются ключевые паттерны параллельного программирования, которые, как правило, используются для параллелизации и выявляются средствами параллелизации в последовательных программах. В **параграфе 3.2** рассматриваются средства анализа зависимостей по данным и анализ выполнения (execution analysis), которые могут использоваться разработчиками для выявления потенциальных участков кода, пригодных для параллелизации. В **параграфах 3.3 и 3.4** рассматриваются соответственно автоматические и полуавтоматические средства выявления параллелизма. **Параграфы 3.5, 3.6, 3.7** посвящены обзору средств параллелизации, использующих методы машинного обучения. В **параграфе 3.5** обсуждается применение подходов классического машинного обучения, в **параграфе 3.6** рассматриваются глубокие нейронные сети (в том числе архитектура «Трансформер») и большие языковые модели (Large Language Models, LLMs), а в **параграфе 3.7** обсуждается использование LLM-агентов для параллелизации программ.

Обзор существующих средств параллелизации показал большой потенциал использования больших языковых моделей для выявления параллелизма и синтеза параллелизующих преобразований.

В **параграфе 3.8** рассматриваются проблемы применения существующих средств параллелизации к критичным к задержке приложениям промышленного уровня. Было показано, что применение решений на основе LLM для параллелизации критичных к задержке приложений промышленного уровня может быть ограничено. В ряде случаев требуется информация о динамических зависимостях и расширенный контекст, включающий, например, определения функций из других файлов, документацию проекта либо исходный код и документацию сторонних библиотек.

Параграф 3.9 посвящён исследованию точности выявления потенциальных параллельных участков кода существующими средствами параллелизации.

Исследование проводилось для следующих решений:
- ИИ-агент Cursor с моделью Claude Sonnet 4;
- большая языковая модель Grok 4;
- большая языковая модель GPT-5 High Reasoning;
- большая языковая модель GigaChat 2 Max;
- большая языковая модель Claude Sonnet 4;
- специализированная модель OMPify;
- система параллелизации DiscoPoP.

Исследование проводилось на разработанном наборе бенчмарков ParaSem, включающем 70 отдельных бенчмарков (программных проектов), в которых было выделено 399 `for`-циклов. Для каждого выделенного цикла средства параллелизации должны были определить возможность параллелизации с использованием OpenMP. Набор ParaSem разделён на три части: «статическую», «динамическую» и «семантическую». В «статической» части для определения параллелизуемости достаточно статического анализа. В «динамической» части параллелизуемость зависит от входных данных, при этом поведение считается стабильным: бенчмарки принимают только фиксированный (разрешённый) набор входов, так что параллелизуемость выбранного цикла остаётся неизменной на этом наборе. В «семантической» части для корректного вывода о параллелизуемости требуется либо документация (в директории `docs` соответствующего бенчмарка), либо дополнительный контекст из других файлов исходного кода.

Сводные результаты на всём наборе бенчмарков ParaSem представлены в таблице 1. Приведены значения метрик точности, прецизионности, полноты и F1-метрики.

Наилучшую точность выявления параллельных циклов показал ИИ-агент Cursor с моделью Claude Sonnet 4. Использование Claude Sonnet

4 в составе ИИ-агента повышает точность на 14.8% по сравнению с использованием той же модели без агентного режима. В режиме «без агента» модель Claude Sonnet 4 ведёт себя консервативно и демонстрирует лишь 37.8% по метрике полноты.

С точки зрения полноты наилучший результат (74.4%) показала специализированная модель OMPify.

Наихудшие значения по всем рассмотренным метрикам показало средство DiscoPoP, что подчёркивает потенциал применения больших языковых моделей для выявления параллелизма.

Результаты по каждой категории представлены в таблицах 2, 3, 4.

Как и ожидалось, для «статической» категории точность выявления параллельных циклов достигла 90.2% для ИИ-агента с моделью Claude Sonnet 4. Все решения на основе больших языковых моделей в «статической» категории показали полноту выше 80%.

Однако для «динамической» категории Cursor с Claude Sonnet 4 показывает точность лишь 67.4%. При этом значения полноты для всех решений на основе больших языковых моделей не превышают 30%.

Для «семантической» категории максимальная точность (59.3%) была достигнута большой языковой моделью GigaChat 2 Max.

Таблица 1 — Сводные результаты выявления потенциально параллельных циклов на наборе бенчмарков ParaSem

Средство параллелизации	**Точность**	**Прецизионность**	**Полнота**	**F1-метрика**	**TP**	**FP**	**TN**	**FN**
Cursor с Claude Sonnet 4	72.4%	79.2%	52.8%	63.3%	95	25	194	85
Grok 4	64.4%	62.5%	52.8%	57.2%	95	57	162	85
GPT-5 High Reasoning	61.7%	60.2%	44.4%	51.1%	80	53	166	100
GigaChat 2 Max	59.7%	55.7%	54.4%	55.1%	98	78	140	82
Claude Sonnet 4	57.6%	54.4%	37.8%	44.6%	68	57	162	112
OMPify	50.9%	47.2%	74.4%	57.8%	134	150	69	46
DiscoPoP	42.4%	36.8%	38.9%	37.8%	70	120	99	110

Таблица 2 — Результаты выявления потенциально параллельных циклов на ParaSem для «статической» категории

Средство параллелизации	**Точность**	**Прецизионность**	**Полнота**	**F1-метрика**	**TP**	**FP**	**TN**	**FN**
Cursor с Claude Sonnet 4	90.2%	85.9%	90.2%	88.0%	55	9	83	6
Grok 4	79.1%	67.1%	93.4%	78.1%	57	28	64	4
GPT-5 High Reasoning	77.1%	65.1%	91.8%	76.2%	56	30	62	5
Claude Sonnet 4	66.0%	54.8%	83.6%	66.2%	51	42	50	10
GigaChat 2 Max	65.4%	54.6%	86.9%	67.1%	53	44	47	8
OMPify	56.9%	46.9%	62.3%	53.5%	38	43	49	23
DiscoPoP	45.8%	35.1%	42.6%	38.5%	26	48	44	35

Таблица 3 — Результаты выявления потенциально параллельных циклов на ParaSem для «динамической» категории

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
Cursor с Claude Sonnet 4	67.4%	94.7%	29.0%	44.4%	18	1	75	44
Claude Sonnet 4	56.5%	100.0%	3.2%	6.2%	2	0	76	60
Grok 4	55.1%	50.0%	11.3%	18.4%	7	7	69	55
GPT-5 High Reasoning	55.1%	50.0%	4.8%	8.8%	3	3	73	59
GigaChat 2 Max	53.6%	45.0%	14.5%	21.9%	9	11	65	53
DiscoPoP	47.8%	43.6%	54.8%	48.6%	34	44	32	28
OMPify	42.0%	43.1%	90.3%	58.3%	56	74	2	6

Таблица 4 — Результаты выявления потенциально параллельных циклов на ParaSem для «семантической» категории

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
GigaChat 2 Max	59.3%	61.0%	63.2%	62.1%	36	23	28	21
Grok 4	55.6%	58.5%	54.4%	56.4%	31	22	29	26
OMPify	53.7%	54.8%	70.2%	61.5%	40	33	18	17
Cursor с Claude Sonnet 4	53.7%	59.5%	38.6%	46.8%	22	15	36	35
GPT-5 High Reasoning	48.1%	51.2%	36.8%	42.9%	21	20	31	36
Claude Sonnet 4	47.2%	50.0%	26.3%	34.5%	15	15	36	42
DiscoPoP	30.6%	26.3%	17.5%	21.1%	10	28	23	47

Проведённое исследование показало, что точность выявления параллелизма может снижаться на 20–30% в системах, использующих большие языковые модели, при наличии динамических зависимостей и сложных семантических взаимосвязей между файлами исходного кода.

Было показано, что для практического применения к критичным к задержке приложениям промышленного уровня необходимо разработать дополнительные методы, повышающие точность выявления параллелизма в системах параллелизации последовательных программ на основе больших языковых моделей.

Четвёртая глава посвящена разработке полуавтоматической системы параллелизации последовательных программ на основе LLM-агента, которая называется Aira. В Aira предложены два новых метода повышения точности выявления параллелизма с помощью больших языковых моделей в критичных к задержке приложениях промышленного уровня: метод семантической индексации и метод динамического анализа. Кроме того, представлен метод определения целесообразности мелкозернистой параллелизации на SMT-ядрах с помощью симулятора производительности процессора по трассам исполнения.

В **параграфе 4.1** описывается общее устройство системы Aira, схема которой представлена на рисунке 8. Система параллелизации Aira была построена на основе ИИ-агента для программирования Cursor и использует большую языковую модель Claude Sonnet 4. Дополнительные средства анализа, в том числе для новых разработанных методов, подключаются к агенту как инструменты через протокол Model Context Protocol.

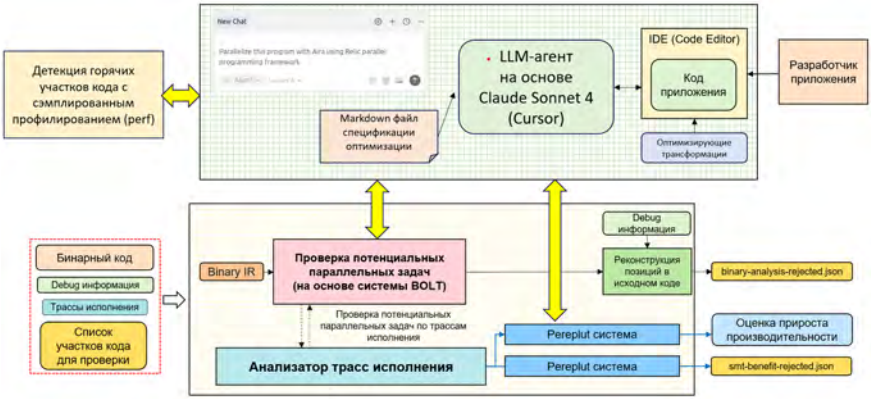

Рисунок 8 — Архитектура системы параллелизации последовательных программ Aira

В **параграфе 4.2** описывается детекция горячих участков кода в системе Aira с помощью сэмплированного профилирования на базе инструмента «perf».

В **параграфе 4.3** описывается общий алгоритм выявления потенциальных параллельных участков кода в системе Aira. Описывается механизм применения функций-аннотаций.

Параграф 4.4 посвящён описанию двух разработанных методов повышения точности выявления потенциального параллелизма с помощью больших языковых моделей в системах параллелизации последовательных программ.

Метод семантической индексации позволяет получить информацию о семантике программы, необходимую для определения того, может ли определённый участок кода быть параллелизован. Дополнительная информация может содержаться в определениях функций и методов, в других файлах исходного кода (например, в виде комментариев) или в документации проекта. Метод основан на семантической индексации файлов исходного кода проекта (в том числе файлов документации).

Общий алгоритм применения семантического анализа состоит из следующих шагов:

1. Получение статической информации об участке кода и его классификация по классам (MAP, FOLD, STENCIL, SCATTER, TASK_SEQUENCE, UNKNOWN) с помощью большой языковой модели.
2. Получение специализированной статической информации об участке кода для соответствующего класса с помощью большой языковой модели.

3. Генерация семантических запросов, специализированных для данного класса.
4. Выделение признаков (сигналов) из полученной семантической информации.
5. Принятие решения о параллелизации участка кода на основе семантического анализа.

Ключевой стадией является генерация семантических запросов. Пример одного из семантических запросов для класса MAP приведён ниже:

> @codebase Show documentation, comments, or tests that mention that the per-element transform used in the MAP loop at file:line (implemented via callee_names) is not thread-safe, relies on mutable global or static state, must be called from a single thread, or otherwise should not be used concurrently.
>
> Prefer explicit statements such as "not thread-safe "must only be used on the main thread "do not call from multiple threads "not safe to parallelize etc. Return the most relevant snippets.

Схема работы разработанного метода динамического анализа представлена на рисунке 9. Была реализована система сбора динамических трасс исполнения с помощью динамической бинарной инструментации (Dynamic Binary Instrumentation, DBI) на основе средства DynamoRIO. Бинарная система анализа динамических зависимостей по трассам исполнения реализована на основе системы BOLT (Binary Optimization and Layout Tool). Для динамического анализа в собранных трассах исполнения выявляются вызовы функций-аннотаций, после чего строятся интервалы обращений к памяти для каждого появления аннотаций в трассе исполнения. На основании этих интервалов обнаруживаются конфликты между регионами, например пересечения адресов обращений к памяти в первой и второй задачах внутри параллельного участка кода.

Для оценки улучшения точности выявления параллелизма при использовании двух разработанных методов применялся разработанный набор бенчмарков ParaSem.

Как показано в таблице 5, для всего набора бенчмарков использование системы Aira позволило достичь точности 90.5%, прецизионности 91.3%, полноты 87.2% и F1-метрики 89.2%. По сравнению с использованием Cursor с моделью Claude Sonnet 4 улучшения составили 18.1%, 12.1%, 34.4% и 25.9% по метрикам точности, прецизионности, полноты и F1-метрики соответственно.

Значительное улучшение метрик точности и полноты, как показано в таблице 7, достигается при использовании метода динамического анализа для «динамической» категории бенчмарков. Использование системы Aira с динамическим анализом показывает улучшения 21.7% и 51.6% по

Рисунок 9 — Схема работы динамического анализа в системе Aira

сравнению с использованием Cursor с моделью Claude Sonnet 4 по метрикам точности и полноты соответственно.

Для «семантической» категории бенчмарков, как показано в таблице 8, использование системы Aira с семантическим анализом показывает улучшения 29.6% и 49.1% по сравнению с использованием Cursor с моделью Claude Sonnet 4 по метрикам точности и полноты соответственно. По сравнению с использованием большой языковой модели GigaChat 2 Max, показывающей наилучшую точность на «семантической» категории бенчмарков среди существующих решений, улучшения при использовании Aira с семантическим анализом составляют 24.0% и 24.5% по метрикам точности и полноты.

Таблица 5 — Таблица с общими результатами выявления потенциальных параллельных циклов с системой Aira на наборе бенчмарков ParaSem

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
Aira	**90.5%**	**91.3%**	**87.2%**	**89.2%**	157	15	204	23
Aira (только с семантическим анализом)	81.0%	85.6%	69.4%	76.7%	125	21	198	55
Aira (только с динамическим анализом)	80.0%	83.8%	68.9%	75.6%	124	24	195	56
Cursor с Claude Sonnet 4	72.4%	79.2%	52.8%	63.3%	95	25	194	85
Grok 4	64.4%	62.5%	52.8%	57.2%	95	57	162	85
GPT-5 High Reasoning	61.7%	60.2%	44.4%	51.1%	80	53	166	100
GigaChat 2 Max	59.7%	55.7%	54.4%	55.1%	98	78	140	82
Claude Sonnet 4	57.6%	54.4%	37.8%	44.6%	68	57	162	112
OMPify	50.9%	47.2%	74.4%	57.8%	134	150	69	46
DiscoPoP	42.4%	36.8%	38.9%	37.8%	70	120	99	110

В **параграфе 4.5** описывается метод определения целесообразности параллелизации на SMT-ядрах.

Прирост производительности от параллелизации на SMT-ядрах зависит от множества факторов, например семантики исполняемого участка кода, устройства кэшей процессора и гранулярности задачи. Существующие системы параллелизации последовательных программ, в особенности системы, использующие большие языковые модели для

Таблица 6 — Таблица с результатами выявления потенциальных параллельных циклов с системой Aira на наборе бенчмарков ParaSem для «статической» категории

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
Aira	**94.8%**	**93.4%**	**93.4%**	**93.4%**	57	4	88	4
Aira (только с динамическим анализом)	91.5%	90.0%	88.5%	89.3%	54	6	86	7
Aira (только с семантическим анализом)	91.5%	87.5%	91.8%	89.6%	56	8	84	5
Cursor с Claude Sonnet 4	90.2%	85.9%	90.2%	88.0%	55	9	83	6
Grok 4	79.1%	67.1%	93.4%	78.1%	57	28	64	4
GPT-5 High Reasoning	77.1%	65.1%	91.8%	76.2%	56	30	62	5
Claude Sonnet 4	66.0%	54.8%	83.6%	66.2%	51	42	50	10
GigaChat 2 Max	65.4%	54.6%	86.9%	67.1%	53	44	47	8
OMPify	56.9%	46.9%	62.3%	53.5%	38	43	49	23
DiscoPoP	45.8%	35.1%	42.6%	38.5%	26	48	44	35

Таблица 7 — Таблица с результатами выявления потенциальных параллельных циклов с системой Aira на наборе бенчмарков ParaSem для «динамической» категории

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
Aira	**91.3%**	**94.6%**	**85.5%**	**89.8%**	53	3	73	9
Aira (только с динамическим анализом)	89.1%	94.3%	80.6%	86.9%	50	3	73	12
Aira (только с семантическим анализом)	67.4%	90.5%	30.6%	45.8%	19	2	74	43
Cursor с Claude Sonnet 4	67.4%	94.7%	29.0%	44.4%	18	1	75	44
Claude Sonnet 4	56.5%	100.0%	3.2%	6.2%	2	0	76	60
Grok 4	55.1%	50.0%	11.3%	18.4%	7	7	69	55
GPT-5 High Reasoning	55.1%	50.0%	4.8%	8.8%	3	3	73	59
GigaChat 2 Max	53.6%	45.0%	14.5%	21.9%	9	11	65	53
DiscoPoP	47.8%	43.6%	54.8%	48.6%	34	44	32	28
OMPify	42.0%	43.1%	90.3%	58.3%	56	74	2	6

Таблица 8 — Таблица с результатами выявления потенциальных параллельных циклов с системой Aira на наборе бенчмарков ParaSem для «семантической» категории

Средство параллелизации	Точность	Прецизионность	Полнота	F1-метрика	TP	FP	TN	FN
Aira (только с семантическим анализом)	**83.3%**	**82.0%**	**87.7%**	**84.7%**	50	11	40	7
Aira	83.3%	85.5%	82.5%	83.9%	47	8	43	10
GigaChat 2 Max	59.3%	61.0%	63.2%	62.1%	36	23	28	21
Grok 4	55.6%	58.5%	54.4%	56.4%	31	22	29	26
OMPify	53.7%	54.8%	70.2%	61.5%	40	33	18	17
Cursor с Claude Sonnet 4	53.7%	59.5%	38.6%	46.8%	22	15	36	35
Aira (только с динамическим анализом)	51.9%	57.1%	35.1%	43.5%	20	15	36	37
GPT-5 High Reasoning	48.1%	51.2%	36.8%	42.9%	21	20	31	36
Claude Sonnet 4	47.2%	50.0%	26.3%	34.5%	15	15	36	42
DiscoPoP	30.6%	26.3%	17.5%	21.1%	10	28	23	47

выявления потенциальных параллельных участков кода, не имеют механизмов оценки прироста производительности и определения целесообразности параллелизации на SMT-ядрах для найденных участков кода.

Была разработана система Pereplut для определения целесообразности параллелизации на SMT-ядрах с помощью симулятора производительности процессора по трассам исполнения; система Pereplut используется в системе параллелизации Aira. Под целесообразностью параллелизации участка кода на SMT-ядре понимается наличие положительного прироста производительности от параллелизации данного участка кода на SMT-ядре.

В разработанной системе используется симулятор производительности процессора по трассам исполнения Sniper как ключевой элемент. В качестве входных трасс используются трассы исполнения, собранные с помощью динамической бинарной инструментации для применения метода динамического анализа.

Трассы исполнения, собранные при последовательном исполнении программы, трансформируются так, чтобы инициировать параллельное исполнение найденных участков кода в системе параллелизации. Инструкции, исполняемые при вызове функций-аннотаций, заменяются на специальные директивы, добавленные в симулятор Sniper. При встрече таких директив в трассе исполнения симулятор Sniper переключается в SMT-режим и запускает задачи параллельно в качестве логических потоков на SMT-ядре. Кроме того, после окончания исполнения участка кода, определяемого также по встрече специальной директивы, симулятор Sniper переключается обратно в последовательный режим исполнения. Количество тактов, затраченное на исполнение рассматриваемого участка кода, фиксируется. Если сокращение числа тактов превышает 10%, то параллелизация данного участка кода на SMT-ядре считается целесообразной в системе Aira.

Для оценки эффективности использования системы Pereplut было проведено сравнение с большой языковой моделью Claude Sonnet 4. Исследование проводилось на наборе бенчмарков из 159 параллельных участков кода, выбранных из популярных наборов для исследования параллелизма: 103 циклов и 56 случаев параллелизма задач (tasks). Для каждого из 159 параллельных участков кода необходимо было определить, даёт ли его мелкозернистая параллелизация на SMT-ядре положительный прирост производительности.

Проведённый анализ показал, что подход, основанный на использовании системы Pereplut, позволил верно определить 145 параллельных участков кода из 159. С использованием большой языковой модели Claude Sonnet 4 удалось верно определить 108 параллельных участков кода из 159. Значения точности (accuracy) для системы Pereplut и Claude Sonnet 4 представлены в таблице 9. Улучшение точности составило 23.3 процентных пункта (п.п.) по сравнению с использованием большой языковой модели Claude Sonnet 4.

Таблица 9 — Сравнение точности определения целесообразности мелкозернистой параллелизации на SMT-ядрах с помощью Pereplut и модели Claude Sonnet 4

Метрика	Pereplut	Claude Sonnet 4	**Улучшение**
Точность (Accuracy)	91.2%	67.9%	**23.3 п.п.**

В **пятой главе** рассматривается исследование улучшения производительности после применения разработанной полуавтоматической системы параллелизации Aira совместно с разработанной системой параллельного программирования Relic к приложениям, критичным к задержке.

Исследование производительности выполнялось на 10 бенчмарках, представляющих широкий спектр приложений, критичных к задержке, из различных доменов, таких как кибербезопасность, высокочастотная торговля (High-Frequency Trading, HFT), робототехника, социальные сети, рекомендательные системы и аэрокосмическая отрасль.

7 из 10 бенчмарков были успешно параллелизованы с помощью разработанной системы параллелизации Aira совместно с системой Relic. Для них среднегеометрический прирост производительности составил 25.2%. Для двух бенчмарков наблюдалось снижение производительности, поскольку, несмотря на оценку эффективности параллелизации с помощью симулятора Sniper, для них зернистость и характеристики задач были определены как достаточные для отправки на исполнение на SMT-поток. Бенчмарк Fraud не был параллелизован, так как на основе симулятора Sniper было сделано верное предсказание, что отправка на исполнение на SMT-поток этих задач приведёт к падению производительности. Среднегеометрический прирост производительности Aira без учёта негативных случаев составляет 17%, тогда как при использовании технологии LLVM OpenMP среднегеометрический прирост производительности составил лишь 6.2%.

В **заключении** приведены основные результаты работы, которые заключаются в следующем:

1. Проведено исследование эффективности использования современных технологий параллельного программирования для мелкозернистой параллелизации на SMT-ядрах.
2. Определены причины и подходы к минимизации задержек при отправке задач на исполнение в современных технологиях параллельного программирования.
3. Разработана специализированная система параллельного программирования Relic, которая показывает прирост производительности более чем на 20% по сравнению с современными технологиями параллельного программирования при мелкозернистой параллелизации на SMT-ядрах.
4. Показана ограниченная применимость современных систем параллелизации последовательных программ для мелкозернистой параллелизации критичных к задержке приложений промышленного уровня на SMT-ядрах.
5. Разработана полуавтоматическая система Aira для мелкозернистой параллелизации критичных к задержке

приложений промышленного уровня на SMT-ядрах. Система Aira была построена на основе LLM-агента с использованием процедурного промптинга с инструментальной поддержкой (tool-augmented procedural prompting). В системе Aira встроенная в агента LLM используется для выявления потенциального параллелизма и синтеза параллелизующих преобразований кода, а в качестве инструментов агента могут подключаться такие средства, как средства динамического профилирования и анализа зависимостей для улучшения точности выявления потенциального параллелизма, средства обнаружения горячих участков кода (hot spots), а также разработанная система определения целесообразности параллелизации на SMT-ядрах процессора по трассам исполнения программы.

6. Разработан метод использования семантической индексации файлов исходного кода программы для улучшения точности выявления потенциального параллелизма с помощью больших языковых моделей в системах параллелизации последовательных программ.
7. Получен прирост производительности 17% на критичных к задержке приложениях за счет их параллелизации системой Aira с использованием системы параллельного программирования Relic.

Публикации автора по теме диссертации

1. *Los*, *D.* Exploring Fine-grained Task Parallelism on Simultaneous Multithreading Cores [Текст] / D. Los, I. Petushkov // International Journal of Open Information Technologies. — 2024. — Т. 12, № 10. — С. 144—151.
2. *Los*, *D.* Accelerating Latency-Critical Applications with AI-Powered Semi-Automatic Fine-Grained Parallelization on SMT Processors [Текст] / D. Los, I. Petushkov // International Journal of Open Information Technologies. — 2025. — Т. 13, № 9. — С. 129—134.
3. *Хайдари*, *Ф. Г.* Аналитическая система оптимизации трасс исполнения для исследования программной предподкачки данных [Текст] / Ф. Г. Хайдари, Д. А. Лось, И. В. Петушков // Труды 66-й Всероссийской научной конференции МФТИ. Радиотехника и компьютерные технологии. — 2024. — С. 27—28.