Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Архитектура вычислительных систем

К защите допустить:

И.О. Заведующего кафедрой информатики
_____ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

СРАВНЕНИЕ ЭФФЕКТИВНОСТИ ВЫПОЛНЕНИЯ ПРЕОБРАЗОВАНИЙ ФУРЬЕ НА ПРОЦЕССОРАХ *AMD RYZEN 5 5500U* И *AMD RYZEN 7 4800H*

БГУИР КП 1-40 04 01 028 ПЗ

Студент Е. В. Савончик

Руководитель А. Н. Марков

Нормоконтролёр А. А. Калиновская

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современной вычислительной технике преобразование Фурье играет ключевую роль в обработке сигналов, компьютерной графике, криптографии и других областях. Этот математический инструмент позволяет анализировать и преобразовывать данные в частотную область, что имеет важное значение для многих прикладных задач.

Преобразование Фурье выделяется своей выдающейся значимостью в сигнальной обработке. Этот метод позволяет извлекать частотные компоненты из данных, что является неотъемлемой частью анализа сигналов в разнообразных областях – от обработки аудиосигналов до распознавания образов в изображениях.

Цель курсовой работы – сравнить эффективность выполнения преобразований Фурье на процессорах *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*. Эти процессоры предназначены для ноутбуков и представляют собой современные вычислительные устройства с различной архитектурой и характеристиками, что делает их интересными объектами для сравнительного анализа. Архитектурные особенности данных процессоров могут оказать влияние на эффективность выполнения алгоритмов, включая преобразование Фурье.

Для дотижения цели необходимо выполнить следующие задачи:

1 Разработка программы для выполнения преобразований Фурье.

2 Реализовать программу, способную проводить преобразования Фурье над входными данными с различными характеристиками.

3 Интеграция измерения времени выполнения.

4 Настройка *Linux*-среды на флеш-накопителе.

5 Подготовить флеш-накопитель с *Linux*-средой для обеспечения устойчивых условий сравнения процессоров.

6 Запуск программы и сбор данных.

7 Анализ и сравнение данных.

Эти задачи составляют ключевой этап в выполнении исследования по сравнительному анализу производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* при выполнении преобразований Фурье.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Современные компьютерные системы предоставляют широкий спектр вычислительных возможностей для решения различных задач. В данной главе будет рассмотрена структура и архитектура двух вычислительных систем, на которых будет проводиться сравнение производительности преобразований Фурье: *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*.

1.1 Структура и архитектура вычислительной системы

AMD Ryzen 7 4800H базируется на архитектуре *Zen 2* и представляет собой процессор высокой производительности. Этот процессор оснащен 8 вычислительными ядрами и 16 потоками, что обеспечивает возможность эффективной работы с широким спектром приложений. Кроме того, у него 8 МБ кэш-памяти *L3*, что способствует более быстрой обработке данных.

Кроме восьми процессорных ядер, *4800H* имеет встроенный видеопроцессор *Radeon RX Vega 7* с 7 вычислительными блоками. Вероятная частота работы: До 1600 МГц. Встроенный в *4800HS* двухканальный контроллер памяти поддерживает *DDR4-3200* и *LPDDR4-4266*. [1]

AMD Ryzen 5 5500U – производительный шестиядерный процессор для тонких ноутбуков. Входит в семейство *Lucienne*, задействует архитектуру *Zen 2*; был представлен в первой половине 2021 года. Рассматриваемый *Ryzen* изготавливается на 7-нанометровом производстве *TSMC*, его процессорные ядра работают с частотой 2.1...4.0 ГГц и располагают удваивающей потоки функцией *SMT*.

AMD Ryzen 5 5500U использует процессорные ядра *Zen 2*, а не *Zen 3*. *Zen 2* крайне компетентная архитектура, с достойной производительностью на ватт и достойной производительностью на МГц. Также он совместим с памятью *DDR4-3200* или *LPDDR4-4266*. Количество кэша третьего уровня = 8 МБ, что аналогично с количеством кэша третьего уровня в *AMD Ryzen 7 4800H*, но по количеству кэша первого и второго уровня *AMD Ryzen 5 5500U* уступает. Этот процессор располагает несколькими линиями *PCI-Express 3.0* для подключения дискретных видеокарт, накопителей данных и устройств других типов; в отличие от пятитысячных *Ryzen* для настольного компьютера, поддержки *PCI-Express 4.0* нет. Максимальная производительность *SSD* будет ограничена 3.9 гигабайтами в секунду. [2]

Сопоставление основных технических характеристик *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* можно найти в таблице 1.1.

Таблица 1.1 – Сравнение основных технических характеристик *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*

Параметр	*AMD Ryzen 5 5500U*	*AMD Ryzen 7 4800H*
Количество ядер процессора	6	8
Количество потоков	12	16
Частота	До 4,0 ГГц	До 4,2 ГГц
Базовые часы	2,1 ГГц	2,9 ГГц
Кэш *L1*	384 КБ	512 КБ
Кэш *L2*	3 МБ	4 МБ
Кэш *L3*	8 МБ	8 МБ
TDP по умолчанию	15 ВТ	45 В
Процессорная технология для ядер ЦП	*TSMC 7nm FinFET*	*TSMC 7nm FinFET*
Максимальная рабочая температура	105°C	105°C
Технологический процесс	7 нм	7 нм
Тип архитектуры ядра	*Zen 2*	*Zen 2*
Тип архитектуры	x86	x86
Поддержка 64 бит	+	+

Архитектура *Zen 2* (рисунок 1.1) от компании *AMD* используется в обоих процессорах. Некоторыми ее ключевыми особенностями по сравнению с другими архитектурами являются [3]:

1 **7-нм техпроцесс**. *Zen 2* была первой архитектурой от *AMD*, созданной с использованием 7-нанометрового техпроцесса. Это позволило значительно увеличить количество транзисторов на кристалле и повысить энергоэффективность.

2 **Многопоточность и многоядерность**. *Zen 2* поддерживает симметричную многопоточность (*SMT*), что позволяет каждому физическому ядру обрабатывать две потоковые задачи. Это дает выигрыш в многозадачности и улучшает общую производительность.

3 **Улучшенная кэш-память**. Каждое ядро *Zen 2* имеет больше кэш-памяти, что улучшает кэш-попадание и снижает задержки при доступе к данным.

4 **Микроархитектурные улучшения**. *Zen 2* внесла множество микроархитектурных изменений, таких как улучшения в предикторах ветвлений и повышенная эффективность исполнения инструкций.

5 **Повышенная *IPC***. Улучшения в архитектуре *Zen 2* привели к значительному повышению инструкций в секунду (*IPC*) по сравнению с предыдущими архитектурами.

6 **Интегрированная графика *Vega***. Процессоры *Zen 2*, включая *Ryzen 5 5500U*, могут включать интегрированный графический ускоритель *Vega*, что делает их подходящими для систем, не требующих дискретной графической карты.

7 **Поддержка *PCI Express 4.0***. Эта архитектура предоставила поддержку новейшего стандарта интерфейса *PCI Express*, что повышает пропускную способность для графических и расширительных карт.

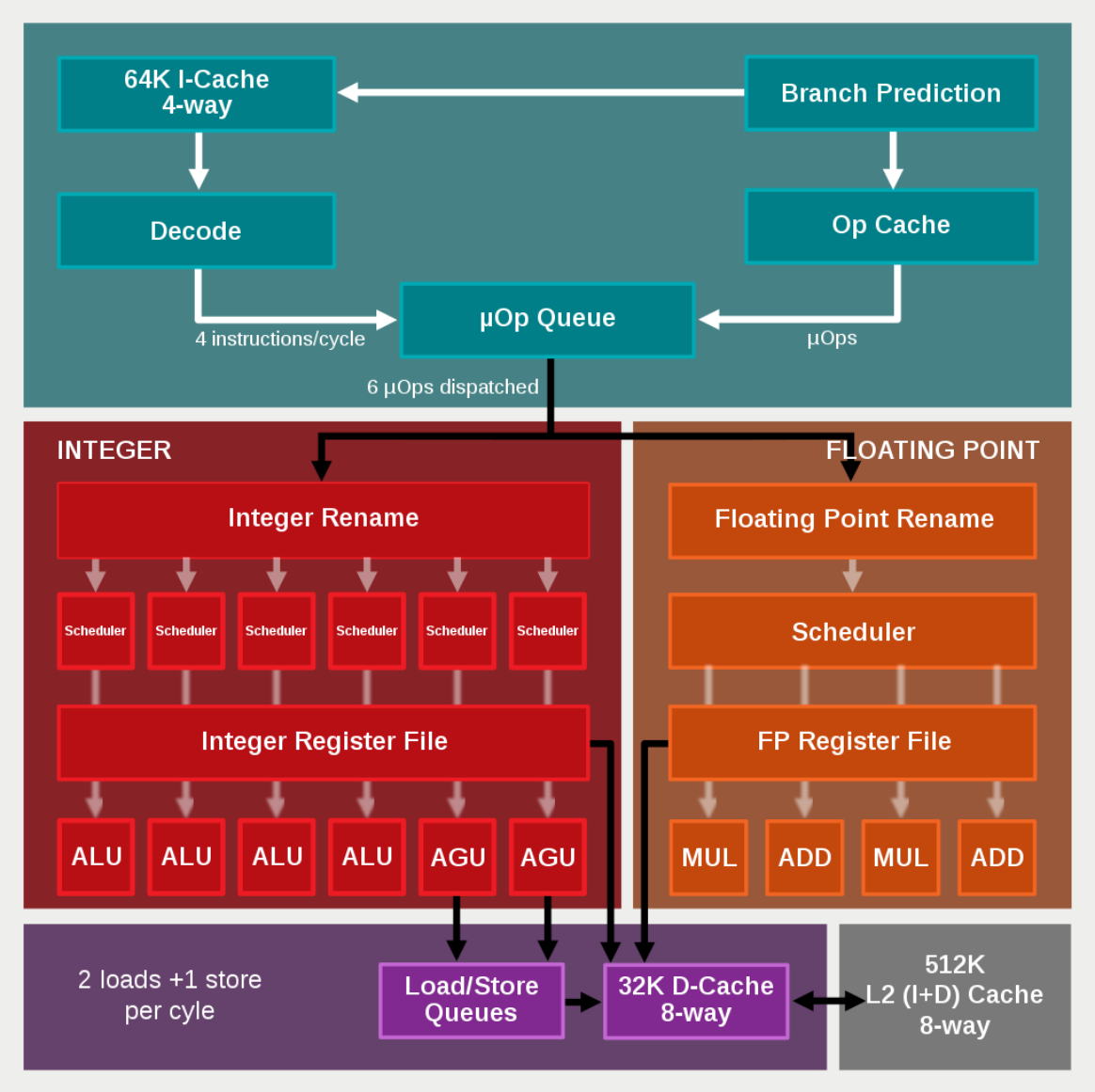

Рисунок 1.1 – Микроархитектура *Zen-2* вычислительных ядер процессоров *AMD Ryzen 7 4800H* и *AMD Ryzen 5 5500U*

В целом, архитектура *Zen 2* представляет собой солидный шаг вперед в развитии процессоров *AMD*, обеспечивая выдающуюся производительность и энергоэффективность.

1.2 История, версии и достоинства

Процессоры *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* являются частью знаменитой линейки процессоров *AMD Ryzen*. Введены они были в продуктовую линейку в разные периоды:

1 *AMD Ryzen 7 4800H* был представлен в начале 2020 года в рамках анонса процессоров для ноутбуков.

2 *AMD Ryzen 5 5500U* начал свой путь в конце 2021 года, продолжая традиции предыдущих моделей.

Версия *4800H* принадлежит к линейке процессоров *Renoir*, основанной на архитектуре *Zen 2*. Эта линейка включает в себя высокопроизводительные процессоры для ноутбуков. [4]

Процессор *5500U* также основан на архитектуре *Zen 2*. Эта модель представляет собой пример эффективного среднебюджетного решения для мобильных устройств. [5]

Достоинствами обоих процессоров можно выделить:

1 Высокая производительность. Оба процессора предназначены для высокопроизводительных систем. *Ryzen 7 4800H* обладает дополнительными ядрами и потоками, что делает его идеальным для ресурсоемких задач.

2 Энергоэффективность. Архитектура *Zen 2* и низкий техпроцесс позволяют снизить энергопотребление, что важно для ноутбуков, где важен баланс между производительностью и длительностью работы от батареи.

3 Интегрированная графика. Оба процессора оснащены интегрированным графическим ускорителем, что позволяет запускать некоторые графические задачи без дополнительного видеоадаптера.

4 Поддержка современных технологий. В обоих случаях присутствуют современные технологии, такие как поддержка *PCI Express 4.0*, что обеспечивает высокую пропускную способность данных.

5 Удобство переноски. Поскольку эти процессоры предназначены для ноутбуков, они обладают хорошей энергоэффективностью, что важно для портативных устройств.

AMD Ryzen 5 5500U и *AMD Ryzen 7 4800H* представляют собой востребованные решения, способные эффективно справляться с широким спектром задач. От повседневных рабочих нагрузок до ресурсоемких

профессиональных вычислений, эти процессоры предоставляют надежную производительность и высокую эффективность.

1.3 Обоснование выбора вычислительной системы

AMD Ryzen 5 5500U и *AMD Ryzen 7 4800H* обладают высокой вычислительной мощностью, необходимой для выполнения сложных математических операций, включая преобразования Фурье. При этом они предлагают разные уровни производительности, что позволяет выбрать оптимальный вариант в зависимости от требований конкретной задачи.

Оба процессора предназначены для мобильных устройств, что подразумевает оптимизацию энергопотребления. Это обеспечивает продолжительную автономную работу и высокую портативность, что является важным аспектом для ноутбуков и мобильных вычислительных систем.

Обе эти процессорные модели основаны на той же архитектуре *Zen 2*, что означает, что они используют аналогичные базовые строительные блоки и принципы функционирования. Это важно для точного сравнения, так как они могут быть оценены в одинаковых условиях.

AMD Ryzen 5 5500U и *AMD Ryzen 7 4800H* совместимы с современными алгоритмами преобразований Фурье, что обеспечивает эффективное выполнение соответствующих вычислительных задач.

Благодаря общей архитектуре *Zen 2*, программное обеспечение и алгоритмы, используемые для выполнения преобразований Фурье, могут быть более точно адаптированы к обоим процессорам. Это снижает возможные искажения, которые могли бы возникнуть из-за различий в архитектуре.

Выбор процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* обоснован, учитывая их высокую производительность, энергоэффективность и совместимость с требуемыми алгоритмами. Эти характеристики делают их подходящими для сравнительного анализа эффективности выполнения преобразований Фурье.

1.4 Анализ выбранной вычислительной системы

AMD Ryzen 5 5500U и *AMD Ryzen 7 4800H* представляют собой обновление по сравнению с предыдущими моделями, обеспечивая выдающуюся вычислительную мощность. В них реализованы передовые технологии, такие как архитектура *Zen 2*, что обеспечивает высокую

эффективность работы, а также поддержка современных интерфейсов, включая *USB 3.0, USB Type-C* и *Ethernet*.

Они базируются на передовой архитектуре *Zen 2*, представляющей собой значительный прогресс в разработке микропроцессоров. Эта архитектура обладает рядом преимуществ, которые делают ее хорошим выбором для анализа.

Важно отметить, что прирост вычислительной мощности и новые технологии, такие как *USB 3.0, USB Type-C* и *Ethernet*, приносят свои преимущества, но могут также потребовать адаптации для некоторого программного обеспечения и устройств, разработанных для предыдущих поколений процессоров.

Однако стоит отметить, что с новыми возможностями приходят и некоторые особенности. Из-за различий в архитектуре *Zen 2*, некоторое программное обеспечение, рассчитанное на предыдущие поколения процессоров, может потребовать адаптации. Кроме того, изменения в выборе портов могут повлечь несовместимость с некоторыми существующими устройствами.

Использование *Zen 2* для анализа обусловлено несколькими преимуществами. Во-первых, эта архитектура предлагает высочайшую эффективность выполнения вычислительных задач благодаря улучшенным характеристикам ядер. Во-вторых, схожие характеристики обоих процессоров на базе *Zen 2* обеспечивают сопоставимые условия для анализа.

Выбор *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* с архитектурой *Zen 2* обеспечивает более надежную и точную основу для сравнительного анализа эффективности выполнения преобразований Фурье, что позволяет более объективно оценить их производительность.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В данной главе рассмотрены выбранные среда разработки и платформа, предназначенные для более точного и объективного сравнения эффективности процессоров в контексте преобразований Фурье. При этом осуществляется обоснование выбора конкретной технологической платформы, которая обеспечивает оптимальные условия для проведения сравнительного анализа процессоров *AMD Ryzen*. Также освещены особенности архитектуры *Zen 2*, которые имеют важное значение в контексте анализа.

2.1 Выбор операционной системы

Сравнивать производительность процессоров представляет определенные трудности. Из-за различий в аппаратном и программном обеспечении, а также в настройках и конфигурациях систем. Эти различия могут привести к искажению результатов тестов и сделать выводы менее достоверными. Поэтому, чтобы дать объективную оценку и сравнить работу процессоров, необходимо создать однородную среду, в которой будут минимизированы внешние факторы, влияющие на результаты тестов.

Может показаться что, самым простым способом создания однородной среды для разных устройств является ручная настройка всех параметров: установка одинаковой версии операционной системы, применение идентичных версий драйверов для аппаратных компонентов, настройка параметров ядра ОС, а также закрытие всех ненужных фоновых приложений и процессов на обоих ноутбуках. Однако этот метод требует значительных усилий и не исключает полностью различия, поскольку на одной и той же ОС различные ноутбуки могут обладать разным аппаратным обеспечением, включая объем оперативной памяти, тип процессора и графическую карту.

Более удобным и эффективным подходом для создания однородной среды является использование виртуальных машин (ВМ). Виртуализация позволяет создать изолированные виртуальные среды, где можно настроить идентичные параметры для проведения сравнительного анализа. Этот метод гораздо более удобен, поскольку виртуальные машины легко создавать, копировать и удалять, а также сохранять и восстанавливать их состояния для последующего использования. Таким образом, можно воспроизвести одинаковые среды на различных физических компьютерах. [6]

Однако даже виртуальная машина (ВМ) может не обеспечивать полной изоляции от хозяйской операционной системы, и в некоторых случаях могут

возникнуть незначительные различия в условиях среды. Виртуализация предоставляет изоляцию на уровне операционной системы (ОС), но не всегда на уровне аппаратного обеспечения. Основные причины, почему ВМ не всегда обеспечивает полную изоляцию, включают:

1 Виртуализация процессора позволяет одному физическому процессору работать с несколькими ВМ. Это может привести к небольшим задержкам и различиям в производительности, поскольку ВМ конкурируют за доступ к ресурсам процессора.

2 Некоторые аппаратные ресурсы, такие как графический процессор (*GPU*), могут быть доступны для нескольких ВМ. Это также может вызвать конфликты и различия в производительности, поскольку ВМ конкурируют за доступ к этим ресурсам.

3 ВМ могут подвергаться влиянию других процессов и задач на хост-системе, что может повлиять на производительность. Например, если на хост-системе выполняется ресурсоемкая задача, это может снизить производительность ВМ.

4 Не всегда легко настроить ВМ точно так, как требуется. Это может привести к различиям в настройках между ВМ, что может повлиять на результаты тестов.

Несмотря на указанные ограничения, использование ВМ для сравнения производительности процессоров является лучшим решением, чем ручная настройка. Это связано с тем, что ручная настройка требует значительных усилий и времени, а также может привести к ошибкам. Использование ВМ позволяет быстро и легко создать однородную среду для сравнения, что повышает точность результатов тестов. Однако всегда существует небольшая степень непредсказуемости, связанная с окружением ВМ.

Загрузка Linux с флешки (*LiveCD* или *LiveUSB*) предоставляет еще один метод для создания однородной среды при сравнении производительности различных систем. Вот несколько причин, почему использование *Linux* с флешки может быть предпочтительным способом в сравнении с виртуальными машинами:

1 При загрузке с флешки *Linux* создает виртуальное окружение, которое изолировано от физического оборудования компьютера. Это значит, что вне зависимости от конфигурации физического компьютера, операционная система на флешке видит одно и то же виртуальное окружение. [7]

2 *Linux* на флешке включает в себя единый стандартный набор драйверов, который поддерживает широкий спектр аппаратного

оборудования. Это уменьшает вероятность возникновения проблем совместимости, что может влиять на точность результатов сравнения.

3 *LiveUSB* поставляется с заранее настроенной операционной системой, что гарантирует, что все параметры системы будут одинаковыми при каждом запуске: настройки сети, пользовательские права, окружение и другие.

4 Поскольку *Linux* с флешки работает в режиме «*read-only*» (только для чтения), это также обеспечивает стабильность и непрерывность среды между запусками. Никакие изменения не сохраняются после перезагрузки, и это позволяет избежать накопления временных данных, которые могли бы повлиять на результаты тестов.

В качестве операционной системы для проведения сравнения используется *Linux* в режиме *LiveUSB*. В качестве дистрибутива используется *Debian* без графической оболочки. Использование этого дистрибутива без графической оболочки на флешке при проведении сравнительного анализа процессоров может иметь несколько преимуществ [8]:

1 *Debian* без графической оболочки обычно требует меньше ресурсов системы, таких как процессорное время, оперативная память и дисковое пространство. Это позволяет максимально использовать ресурсы для сравнения производительности процессоров.

2 Отсутствие графической оболочки уменьшает количество процессов и служб, работающих в фоновом режиме, что способствует стабильности системы в процессе тестирования.

3 Отсутствие графического интерфейса минимизирует вероятность вмешательства факторов, связанных с графической подсистемой, в результаты тестирования. Это делает среду более однородной для сравнения различных процессоров.

4 Дистрибутив *Debian* предоставляет удобные инструменты для установки и настройки системы. Выбор минимальной конфигурации без графической оболочки упрощает процесс подготовки к проведению тестов.

5 *Debian* является гибкой и настраиваемой операционной системой, что позволяет легко настроить среду под конкретные потребности сравнительного анализа производительности.

Использование *Debian* без графической оболочки на флешке обеспечивает легкость, эффективность и стабильность при проведении тестов и анализе производительности процессоров.

Архитектура *Zen 2* в сочетании с операционной системой *Linux* предоставляет несколько важных преимуществ для проведения анализа производительности:

1 Эффективное управление ресурсами. Улучшенное управление памятью и распределение ресурсов позволяют оптимизировать вычислительные задачи. Это особенно важно при работе с большими объемами данных и при выполнении параллельных вычислений.

2 Оптимизированные инструкции для *Linux*. Новые наборы инструкций, предназначенные специально для работы под *Linux*, могут ускорить выполнение определенных видов задач. Это может быть критично при анализе производительности, где каждая оптимизация важна.

3 Поддержка аппаратных ускорителей. Если в анализе используются аппаратные ускорители, то *Zen 2* предоставляет расширенные возможности для их эффективной интеграции и взаимодействия с *Linux*.

4 Обработка потоков и параллелизм. Поддержка многопоточности в архитектуре *Zen 2* может существенно ускорить выполнение многопоточных приложений, что особенно важно в анализе больших объемов данных.

5 Энергопотребление и производительность. Управление энергопотреблением позволяет оптимизировать работу процессора под различными нагрузками, что может быть важно при анализе производительности в различных режимах работы.

6 Совместимость и стабильность. Улучшенная совместимость с *Linux* и оптимизированные драйвера обеспечивают стабильную работу системы, что критически важно при проведении анализа производительности.

7 Удобство разработки и отладки. Взаимодействие *Zen 2* с *Linux* предоставляет разработчикам широкие возможности для эффективной разработки, тестирования и отладки программ, что может быть критично для анализа производительности.

Исходя из этих факторов, архитектура *Zen 2* в сочетании с *Linux* предоставляет мощную и эффективную платформу для анализа производительности, позволяя максимально эффективно использовать вычислительные ресурсы и получать точные и надежные результаты.

2.2 История, версии и достоинства

Дистрибутив Debian, созданный на основе одного из старейших дистрибутивов *Linux* в мире, был представлен впервые в 1993 году. *Debian* зародился из идеи объединения лучших качеств существующих операционных систем для создания интуитивно понятной и доступной платформы. Вдохновленный этим стремлением, *Debian* продолжает развиваться и поддерживать идеалы свободного программного обеспечения.

История *Debian* началась в августе 1993 года, когда Ян Мердок, студент Университета Торонто, решил создать собственный дистрибутив операционной системы *Linux*. Вдохновленный идеей объединения лучших качеств существующих дистрибутивов, Мердок ставил перед собой целью создать бесплатную, открытую платформу с устойчивым кодом. Он начал с разработки набора инструментов для сборки *Linux*, а затем собрал команду разработчиков для создания дистрибутива. [9]

Debian стал первым дистрибутивом, который внедрил концепцию свободного программного обеспечения, предоставляя только те пакеты, которые соответствовали определенным стандартам свободы.

Debian, поддерживаемый широким сообществом разработчиков и пользователей, ставит своей целью предоставить стабильную и надежную операционную систему. В отличие от *Ubuntu, Debian* не придерживается строгого графика релизов, но подчеркивает долгосрочную стабильность и надежность в своем развитии.

Основные преимущества *Debian* [9]:

Активное сообщество и поддержка. *Debian* обладает активным сообществом разработчиков и пользователей, готовых оказать помощь и поддержку новичкам.

Широкий выбор программного обеспечения. *Debian* предоставляет обширный выбор приложений, а установка новых программ становится легкой благодаря мощному менеджеру пакетов.

Стабильность и надежность. *Debian* славится своей стабильностью, что делает его отличным выбором для серверов и долгосрочных проектов.

Бесплатное и открытое ПО. *Debian* следует принципам свободного программного обеспечения, предоставляя возможность свободного использования, модификации и распространения программ.

Регулярные обновления. *Debian* предоставляет обновления безопасности и исправления багов, обеспечивая долгосрочную поддержку для некоторых версий.

Безопасность. *Debian* акцентирует внимание на безопасности, предоставляя современные механизмы безопасности и системы обновления.

Минимальные системные требования. *Debian* может работать на относительно слабых компьютерах, обеспечивая доступность для широкого круга пользователей.

Debian является надежным и мощным дистрибутивом *Linux*, который подходит для широкого круга пользователей. Он имеет долгую историю, и его

активное сообщество разработчиков и пользователей продолжает вносить свой вклад в его развитие.

2.3 Анализ среды разработки для написания программы

В качестве языка программирования для написания программы используется *C++*. *C++* имеет несколько преимуществ для работы с процессорами, ядрами и потоками, особенно при использовании библиотеки *OpenMP* [10]:

1 *OpenMP* обеспечивает простой и удобный способ добавления параллельности к коду на *C++*. Директивы *OpenMP* интегрируются в существующий код без необходимости значительных изменений, что делает их легкими в использовании. [11]

2 *C++* известен своей высокой производительностью, что особенно важно при выполнении вычислительно интенсивных задач. В сочетании с *OpenMP*, который предоставляет средства для параллельного выполнения кода, можно эффективно использовать вычислительные ресурсы процессора.

3 *C++* поставляется с обширной стандартной библиотекой, включая контейнеры, алгоритмы, потоки и другие компоненты. Это облегчает написание эффективного и структурированного кода для анализа производительности.

4 Код, написанный с использованием *OpenMP*, может быть легко перенесен на различные платформы, так как большинство современных компиляторов поддерживают стандарт *OpenMP*. Это обеспечивает кроссплатформенность и удобство в поддержке.

5 *OpenMP* предоставляет возможность динамического управления числом потоков во время выполнения программы. Это позволяет более эффективно использовать ресурсы системы в зависимости от текущей нагрузки.

В целом, *OpenMP* в сочетании с *C++* обеспечивает удобные и эффективные средства для работы с многопоточностью, позволяя легко внедрять параллельные решения и повышать производительность программ.

Для проведения исследования эффективности выполнения преобразований Фурье на процессорах *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* необходимо использовать среду разработки, которая соответствует следующим требованиям:

1 Поддержка языка программирования *C/C++*.

2 Поддержка инструментов и библиотек, в том числе библиотеки *OpenMP*.

3 Поддержка инструментов и возможностей для разработки и отладки программ.

Для оценки среды разработки по указанным требованиям были определены следующие критерии:

Поддержка языка программирования *C/C++*. Среда разработки должна поддерживать язык программирования *C/C++*, поскольку он является основным языком для реализации алгоритмов преобразования Фурье.

Поддержка инструментов и возможностей для разработки и отладки программ. Среда разработки должна предоставлять широкий спектр инструментов и возможностей для разработки и отладки программ. Это позволит облегчить и ускорить процесс разработки и отладки программы.

Visual Studio поддерживает языки программирования *C* и *C++*. Это является одним из основных преимуществ среды, поскольку *C/C++* являются основными языками для реализации алгоритмов преобразования Фурье. Также она обеспечивает богатый набор инструментов и библиотек для работы с аппаратным обеспечением. Она интегрирована с различными инструментами для анализа и оптимизации производительности, что позволяет эффективно использовать процессоры *AMD Ryzen*. [12]

Среда разработки – ключевой инструмент в создании программного обеспечения. В данной работе используется *Visual Studio*, одна из наиболее мощных и универсальных сред разработки. Разработанная компанией *Microsoft*, она предоставляет обширный набор инструментов и возможностей для разработки приложений под разные платформы. Также соответствует всем указанным требованиям. Она поддерживает язык программирования *C/C++*, предоставляет инструменты и библиотеки для работы с аппаратным обеспечением, включая процессоры *AMD Ryzen*, и предоставляет широкий спектр инструментов и возможностей для разработки и отладки программ.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ ПРОВЕДЕНИЯ СРАВНЕНИЯ

3.1 Обоснование необходимости анализа

В современном информационном обществе, где данные играют ключевую роль, преобразование Фурье имеет огромное значение. Оно используется в различных областях, начиная от обработки сигналов в телекоммуникациях и медицине, заканчивая анализом данных в финансах, машинном обучении и искусственном интеллекте. Благодаря преобразованию Фурье, можно анализировать данные в частотной области, что позволяет выявлять важные характеристики и закономерности, не всегда очевидные в исходной форме данных.

Преобразование Фурье – это математический метод, который позволяет перейти от представления данных во временной области к их представлению в частотной области. Иными словами, это процесс разложения сложного сигнала на сумму элементарных частотных компонент. [13]

Преобразование Фурье особенно полезно в анализе и обработке сигналов, так как оно позволяет выделить частоты, присутствующие в сигнале, и оценить их амплитуду. Это дает возможность выявлять различные характеристики сигнала, такие как доминирующие частоты, амплитуды колебаний, фазовые отношения и др.

Кроме того, преобразование Фурье имеет широкое применение в обработке изображений, где оно используется, например, для компрессии данных, фильтрации и улучшения качества изображений.

В области науки и инженерии, преобразование Фурье является неотъемлемой частью многих алгоритмов и методов анализа данных. Его применение простирается от физики и инженерии до медицины и биологии. [14]

Процессоры *AMD Ryzen* получили признание и популярность на рынке вычислительной техники. Исследование и сравнение эффективности выполнения преобразований Фурье на таких процессорах становится важным в контексте выбора оборудования для различных задач.

Кроме того, *AMD* – компания, которая постоянно внедряет инновации в свои процессоры. Сравнение производительности моделей двух разных поколений позволит наглядно проследить, как эволюционировала архитектура процессоров с течением времени. Это может быть интересно для тех, кто следит за новейшими технологическими достижениями.

Сравнительный анализ процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* предоставит важную информацию о том, как различные архитектурные особенности влияют на эффективность выполнения преобразований Фурье. Эта исследование имеет не только практическую значимость для разработчиков и инженеров, выбирающих оборудование для своих проектов, но также представляет интерес в контексте анализа производительности процессоров разных поколений. Сравнение между *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* позволит выявить изменения в архитектуре и производительности, а также, проведение анализа преобразований Фурье поможет выявить дополнительные аспекты, которые не всегда заметны при общем рассмотрении технических характеристик процессоров. Этот метод позволит получить более подробное представление о том, как эти две модели процессоров различаются и в чем заключаются их особенности.

Сравнительный анализ предполагает детальное сопоставление эффективности выполнения преобразований Фурье на двух различных процессорах: *AMD Ryzen 5 5500U и AMD Ryzen 7 4800H*. Для проведения сравнительного анализа, необходимо выполнить следующие шаги:

1 **Подготовка данных**. Сначала необходимо подготовить одинаковые наборы данных для анализа. Эти данные могут представлять собой сигналы разной природы, от аудио до изображений.

2 **Выбор алгоритмов**. Выбрать алгоритмы преобразования Фурье, которые будут использоваться для сравнения.

3 **Разработка исследовательских программ**. Написать программу для выполнения преобразования Фурье на обоих процессорах. Убедиться, что они используют аппаратные возможности каждого процессора.

4 **Измерение времени выполнения**. Запустить программы на обоих процессорах с одинаковыми входными данными и замерить время, необходимое для выполнения преобразования.

5 **Анализ результатов**. Сравнить времена выполнения на обоих процессорах. Оценить, какая архитектура процессора более эффективно обрабатывает преобразование Фурье.

6 **Интерпретация данных**. По результатам сравнительного анализа сделать выводы о том, какой процессор более подходит для выполнения преобразования Фурье в данном. Рассмотреть как аспекты производительности, так и возможные преимущества каждого процессора.

7 **Дополнительные аспекты**. При необходимости, рассмотреть иные факторы, такие как энергоэффективность, тепловыделение и прочее, которые могут иметь важное значение в конкретной среде применения.

8 **Формулирование выводов**. На основе проведенного сравнительного анализа сделать обоснованные выводы о том, какой процессор является более эффективным для выполнения преобразования Фурье в рассматриваемых условиях.

Результаты сравнительного анализа эффективности выполнения преобразований Фурье на процессорах *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* обладают значимостью в контексте анализа различий между этими двумя процессорами. Анализ направлен на выявление конкретных характеристик, аспектов и особенностей каждого процессора в процессе выполнения преобразований Фурье. Полученные данные позволяют глубже понять, какие именно аспекты архитектуры и производительности влияют на результаты преобразований, а также помогают выделить ключевые отличия между *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в данном контексте.

Полученные результаты могут служить отправной точкой для дальнейших исследований в области оптимизации преобразования Фурье и адаптации его к различным архитектурам процессоров.

3.2 Технологии программирования, используемые для решения поставленных задач

Для эффективной реализации алгоритмов преобразования Фурье на процессорах *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* был выбран набор современных технологий программирования.

Основой реализации алгоритмов преобразования Фурье послужил мощный и гибкий язык программирования *C++*. Этот выбор обусловлен высокой производительностью и поддержкой множества библиотек, что важно при обработке больших объемов данных. [15]

C++ имеет богатую историю, начиная с создания в 1979 году как расширения языка C. Он был разработан для обеспечения высокоуровневых возможностей абстракции данных и структурного программирования, сохраняя при этом близость к языку *C*. В последующие десятилетия *C++* стал одним из наиболее распространенных языков программирования, привнося в разработку множество инноваций.

Для упрощения и ускорения разработки были активно задействованы стандартные библиотеки *C++*, предоставляющие широкий спектр функций для работы с массивами, математическими операциями и другими компонентами, необходимыми для преобразования Фурье.

Для решения задач курсовой работы используются следующие библиотеки:

1 *Vector*. Предоставляет контейнер *std::vector*. Вектор представляет собой динамический массив, который может изменять свой размер во время выполнения программы. Используется для удобного и эффективного хранения и манипулирования динамическими массивами данных.

2 *Cmath*. Используется для выполнения математических вычислений, таких как вычисление корней, логарифмов, тригонометрических функций и прочих.

3 *Iostream*. Предоставляет потоковые операции ввода/вывода. Используется для взаимодействия с пользователем через консоль, вывода результатов выполнения программы и получения входных данных.

4 *Chrono*. Предоставляет возможности для работы с временем. Используется для измерения времени выполнения различных участков программы и оценки производительности. [16]

5 *OpenMP* (*Open Multi-Processing*). Используется для добавления параллельных конструкций и управления многопоточностью в программе. *OpenMP* обеспечивает средства для создания параллельных участков кода, распределения задач между потоками.

Программный код был подвергнут оптимизациям, предоставляемым компилятором. К ним относятся векторизация и оптимизации циклов, что позволяет эффективнее использовать вычислительные ресурсы. [17]

Выбор данных технологий обеспечивает высокую производительность программы и её адаптацию к особенностям архитектур процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*, что является ключевым аспектом при проведении сравнительного анализа.

3.3 Роль архитектуры в сравнительном анализе

Для начала стоит дать определение сущностям, с которыми тесно связана работа программного продукта данной курсовой работы: физическое и логическое ядро процессора, процесс и поток.

Физическое ядро – это непосредственное вычислительное устройство на чипе процессора. У современных процессоров физических ядер несколько на одном процессоре. Каждое физическое ядро способно выполнять независимые вычисления и имеет свои ресурсы, такие как арифметические блоки и кэш-память.

Логическое ядро – это виртуальное вычислительное устройство, предоставляемое процессором на основе физического ядра. Логическое ядро используется для увеличения параллельности выполнения задач. У процессоров с поддержкой *SMT*, каждое физическое ядро может иметь два или более логических ядра. Логические ядра разделяют ресурсы физического ядра, и это позволяет выполнять несколько потоков одновременно. Это увеличивает общую производительность процессора. [18]

SMT (*Simultaneous Multithreading*) – аналог технологии *Intel Hyper-Threading*, созданный *AMD*. Это технология, которая позволяет одному физическому ядру поддерживать выполнение нескольких потоков одновременно. Это достигается путем предоставления нескольких логических ядер на каждом физическом ядре. *SMT* увеличивает эффективность использования ресурсов процессора, так как разные потоки могут запускать вычисления, даже если один из них заблокирован, например, из-за ожидания данных из памяти. [19]

Процесс в операционной системе – это изолированная программа или задача, которая выполняется в оперативной памяти компьютера. Каждый процесс имеет собственное адресное пространство, в котором хранятся его переменные и данные. Процессы обычно изолированы друг от друга и не могут напрямую взаимодействовать с памятью других процессов.

Поток (*thread*) – это более легковесная единица выполнения внутри процесса. Он использует общее адресное пространство процесса и может взаимодействовать с другими потоками внутри того же процесса. Потоки внутри одного процесса могут делиться данными и ресурсами.

Связь между этими сущностями заключается в том, что процессор состоит из физических ядер, и каждое физическое ядро может поддерживать одно или несколько логических ядер. Внутри каждого процесса можно создавать потоки для выполнения различных задач. Эти потоки могут выполняться на доступных логических ядрах (как физических, так и виртуальных) процессора, что позволяет распараллеливать задачи и улучшать производительность в многозадачных системах.

Понимание данной связи позволяет рассмотреть, как именно используемые в курсовой работе технологии программирования взаимодействуют с архитектурой вычислительной системы.

Архитектура вычислительной системы напрямую определяет производительность и эффективность программного обеспечения. Исследование фокусируется на сравнении двух процессоров *AMD Ryzen: AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*. Эти процессоры обладают различными

характеристиками и архитектурными особенностями, которые влияют на их способность обработки вычислительных задач, включая операции преобразования Фурье.

Физические и логические ядра процессоров, а также применение технологии *SMT* (*Simultaneous Multithreading*), оказывают значительное воздействие на их производительность. Например, различия в количестве ядер и потоков могут повлиять на параллельную обработку задач, что критически важно для операций, таких как преобразования Фурье. Это позволяет эффективнее использовать ресурсы процессора при выполнении множества вычислительных задач одновременно.

Кроме того, архитектурные особенности, такие как размер кэш-памяти, тактовая частота и особенности инструкций, могут оказать влияние на производительность при выполнении вычислительных операций. Анализ этих параметров помогает понять, как процессоры обрабатывают данные, в том числе и при преобразованиях Фурье.

Процессор *AMD Ryzen 5 5500U* представляет собой энергоэффективное решение, разработанное для мобильных устройств. Его архитектура включает в себя несколько вычислительных ядер с поддержкой симметричной многопоточности (*SMT*), что позволяет обрабатывать несколько потоков данных одновременно. Эта архитектура может быть особенно полезной для задач, требующих параллельной обработки.

Процессор *AMD Ryzen 7 4800H*, в свою очередь, представляет собой более производительное решение, также ориентированное на мобильные устройства. Он обладает большим числом вычислительных ядер и поддерживает технологию *SMT*. Это позволяет распределить нагрузку на несколько ядер, что особенно важно для многопоточных приложений.

Архитектура *Zen 2* представляет собой значительный шаг вперёд в разработке микропроцессоров компании *AMD*. Она имеет прямое влияние на разработку программного обеспечения и способствует улучшению производительности приложений. Вот как архитектура *Zen 2* взаимодействует с программным обеспечением и обеспечивает преимущества:

Эффективная многопоточность. *Zen 2* обладает продвинутой многопоточностью, что позволяет эффективно распределить вычислительные нагрузки между ядрами процессора. Это особенно важно для многопоточных приложений, которые могут параллельно выполнять несколько задач.

Улучшенная архитектура кэша. Кэш-память *Zen 2* оптимизирована для ускорения доступа к данным, что сокращает задержки при обращении к памяти. Это значительно повышает скорость выполнения программ.

Расширенные *SIMD*-инструкции. *Zen 2* поддерживает расширенные наборы инструкций *SIMD*, что позволяет одновременно обрабатывать больше данных. Это полезно для приложений, требующих высокой производительности в области обработки сигналов и графики.

Энергоэффективность. Архитектура *Zen 2* разработана с учетом энергосбережения, что делает процессоры данной серии эффективными как в рабочих станциях, так и в ноутбуках. Это важно для мобильных приложений и устройств с ограниченным источником питания.

Возможности адаптации под конкретные задачи. Благодаря гибкости архитектуры *Zen 2*, разработчики имеют возможность оптимизировать программы под конкретные задачи, что позволяет достичь высокой эффективности при решении специализированных задач.

Поддержка новейших технологий. *Zen 2* интегрирует в себя последние технологические достижения, такие как поддержка *PCI Express 4.0*, что обеспечивает высокую пропускную способность данных.

Физические и логические ядра процессоров, а также технология *SMT*, оказывают воздействие на их производительность. Различия в количестве ядер и потоков могут существенно повлиять на параллельную обработку задач, что важно для операций, таких как преобразования Фурье, позволяя эффективнее использовать ресурсы процессора при выполнении множества вычислительных задач одновременно.

Таким образом, архитектурные особенности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* могут влиять на эффективность выполнения алгоритмов преобразования Фурье. Анализ этих особенностей помогает понять, как процессоры обрабатывают данные, в том числе и при преобразованиях Фурье.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

В данной главе рассматривается архитектурная структура разрабатываемой программы, описываются её функциональные возможности и принципы работы.

4.1 Инициализация данных

При запуске программы вводится длина сигнала. Этот этап является важным шагом перед проведением анализа, поскольку позволяет протестировать скорость выполнения алгоритма на различных объемах данных.

Длина сигнала существенно влияет на алгоритмы преобразования Фурье. Алгоритм Фурье имеет временную сложность *O(N^2)*, что означает, что количество операций, необходимых для его выполнения, растет квадратично в зависимости от размера сигнала. Другими словами, при увеличении длины сигнала в два раза, количество операций увеличится в четыре раза.

Это важно учитывать при выборе длины сигнала для анализа. Например, при небольших объемах данных алгоритм может работать очень быстро. Однако, при анализе больших сигналов, скорость работы алгоритма может существенно снижаться из-за высокой вычислительной нагрузки.

Таким образом, подбор оптимальной длины сигнала играет ключевую роль в проведении сравнительного анализа эффективности алгоритмов преобразования Фурье на различных процессорах. Этот аспект позволяет оценить эффективность алгоритма в реальных условиях, что важно для последующего сравнительного анализа производительности на разных процессорах.

4.2 Генерация тестовых данных

Для проведения сравнительного анализа эффективности преобразования Фурье важно иметь тестовые данные, которые позволяют оценить работу алгоритма в различных условиях. Программа предоставляет инструмент для генерации таких данных.

При запуске программы пользователь имеет возможность указать длину сигнала. Это позволяет протестировать алгоритм на различных наборах данных, что важно для получения более точной оценки его эффективности.

В качестве тестовых данных для алгоритма преобразования Фурье используется синусоида с частотой 0,1, но, благодаря гибкости программы, существует возможность варьировать длину сигнала для тщательного тестирования алгоритма в различных условиях.

Это позволяет проводить анализ эффективности алгоритма на различных процессорах и оценивать, как он справляется с разными типами входных данных. Таким образом, программа предоставляет не только средства для анализа, но и возможность генерировать тестовые данные, соответствующие конкретным задачам и условиям эксперимента.

4.3 Реализация алгоритмов преобразования Фурье

Для сравнения производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в программе реализованы алгоритмы дискретного преобразования Фурье (*DFT*). Каждый алгоритм работает в соответствии с формулой, что обеспечивает надежность и точность расчетов. [20]

Дискретное преобразование Фурье является одним из фундаментальных алгоритмов в обработке сигналов и анализе данных. Оно позволяет анализировать сигналы в частотной области, выделяя их частотные компоненты. Этот алгоритм основан на идее разложения сложного сигнала на более простые синусоидальные составляющие разных частот.

Алгоритм *DFT* преобразует последовательность *N* дискретных отсчетов в комплексный спектр, представленный *N* комплексными числами. В данной реализации, для каждого комплексного числа вычисляется вещественная и мнимая части. Эти значения представляют собой действительные и мнимые части комплексного числа, соответствующего частотной компоненте сигнала. [21]

Для этого вычисляется сумма по всем *N* отсчетам сигнала. Внутри этой суммы используется угловая частота, определенная как $2\pi kn/N$, где k – индекс текущей компоненты частотного спектра, а n – индекс текущего отсчета.

Вычисление реальной и мнимой частей позволяет получить комплексное число, представляющее амплитуду и фазу частотной компоненты. После этого, с использованием теоремы Пифагора, вычисляется модуль этого комплексного числа, что представляет собой амплитуду этой компоненты.

Алгоритм дискретного преобразования Фурье (*DFT*) выбран для сравнения скорости двух процессоров по нескольким причинам:

Ресурсоемкость. *DFT* – вычислительно интенсивный алгоритм, который требует большое количество операций с плавающей запятой. Таким образом, он позволяет оценить производительность процессоров в вычислительно сложных сценариях.

Стандартизированный алгоритм. *DFT* имеет стандартную математическую формулировку, что позволяет сравнивать производительность процессоров на одних и тех же математических основах.

Чувствительность к оптимизациям. Алгоритм *DFT* поддается различным оптимизациям. Это позволяет оценить эффективность архитектуры процессора в применении оптимизированных алгоритмов.

Потенциал для распараллеливания. Некоторые варианты алгоритма *DFT* могут быть распараллелены, что позволяет использовать многопоточные процессоры более эффективно.

Таким образом, анализ производительности на алгоритме *DFT* предоставляет важную информацию о способности процессора обрабатывать вычислительно сложные задачи в различных областях применения.

4.4 Измерение времени выполнения

Для оценки производительности алгоритмов преобразования Фурье используется измерение времени выполнения. Это позволяет установить разницу в эффективности выполнения алгоритмов на различных процессорах. Измерение времени выполнения является ключевым инструментом при сравнительном анализе производительности алгоритмов. Оно позволяет оценить, насколько эффективно алгоритм работает на конкретном оборудовании.

Использование времени выполнения позволяет установить разницу в производительности между разными архитектурами процессоров. Более эффективный процессор сможет выполнять алгоритм за меньшее количество времени, что является важным критерием при выборе оборудования для конкретной задачи.

Принцип измерения времени выполнения заключается в фиксации начального и конечного временных моментов выполнения алгоритма. В языке *C++*, для этого используются специализированные библиотечные функции. Эти функции предоставляют точные механизмы для подсчета временного интервала между началом и окончанием работы алгоритма.

При проведении сравнительного анализа с помощью времени выполнения, важно убедиться, что тестовые условия максимально схожи. Это включает в себя одинаковые объемы данных, одинаковые начальные условия и прочие параметры, которые могут повлиять на результаты тестирования.

Конфигурация окружения тестирования играет ключевую роль в достоверности результатов сравнительного анализа. Для обеспечения равных условий для процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* было принято решение использовать флеш-накопитель с предустановленной операционной системой Linux. Это позволяет создать стандартизированное окружение, где обе архитектуры процессоров работают на аналогичных условиях без дополнительного влияния операционной системы на процесс тестирования.

Кроме того, важно учесть такие факторы, как температурные условия и нагрузка на процессоры во время тестирования. Равные условия тестирования помогут минимизировать влияние внешних факторов на результаты анализа производительности, что сделает полученные данные более объективными и надежными для последующего сравнения двух процессоров.

4.5 Общая структура программы

Программа начинается с включения необходимых заголовочных файлов для работы с различными библиотеками. <*vector*> используется для работы с динамическими массивами, <*cmath*> предоставляет математические функции, <*iostream*> используется для ввода и вывода данных, <*chrono*> используется для работы с временем, а <*omp.h*> включает функционал *OpenMP* для параллельного программирования.

Затем определен пользовательский тип данных *RealVector*, представляющий вектор действительных чисел типа double. Этот тип данных используется для хранения сигналов и результатов преобразования Фурье.

Функция *dft_parallel*: Эта функция реализует алгоритм дискретного преобразования Фурье (*DFT*) с использованием параллельных вычислений. На вход она принимает вектор x с исходными данными и вектор *X*, в который записываются результаты преобразования. Параллельная обработка выполняется с использованием директив *#pragma omp*, что позволяет использовать *OpenMP* для распараллеливания циклов.

Функция *main*: Программа начинает выполнение с функции main. Пользователю предлагается ввести длину сигнала. Процесс ввода параметров сигнала позволяет задать длину сигнала, что важно для детального анализа

алгоритма преобразования. После ввода, программа генерирует тестовый сигнал в виде синусоиды с частотой 0,1. Этот этап необходим для обеспечения начальных данных для анализа. Синусоида с низкой частотой обычно используется для наглядности и может быть заменена на другие сигналы в зависимости от требований. Далее, вызывается функция *dft_parallel*, которая применяет параллельное преобразование Фурье к сигналу. Замеряется время выполнения этой операции с помощью библиотеки <*chrono*>. И, наконец, результаты анализа выводятся в консоль. Программа выводит время выполнения в миллисекундах. Полученные результаты необходимы для дальнейшего сравнительного анализа производительности процессоров.

Выбор данной программной схемы обладает несколькими преимуществами. Возможность задать длину сигнала делает программу универсальной для анализа различных входных данных. Это позволяет проводить тестирование на различных объемах данных, что важно при сравнении производительности алгоритмов на различных системах. Этот подход позволяет не только сравнивать производительность между разными процессорами, но и проводить анализ влияния изменений в архитектуре процессора на общую эффективность выполнения алгоритма.

В целом, данная схема программы предоставляет удобные инструменты для анализа производительности алгоритма преобразования Фурье на различных конфигурациях системы. Она объединяет гибкость в выборе длины сигнала с эффективным использованием параллельных вычислений, что делает ее подходящей для сравнительного анализа производительности процессоров.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Глава описывает основные компоненты и логическую организацию разрабатываемого программного продукта. В ней раскрываются основные компоненты и их взаимосвязи, предоставляя читателю четкое представление о том, как работает программа и каким образом выполняется сравнение процессоров. были проведены запуски на одном и на всех ядрах с включенным и отключенным *SMT*. Целью этих запусков было оценить производительность программы в различных режимах работы и определить влияние *SMT* на ее производительность.

Запуск программы на одном ядре позволяет оценить ее базовую производительность. Это важно, поскольку позволяет понять, насколько эффективна программа в использовании вычислительных ресурсов. Если программа работает медленно на одном ядре, то это может быть связано с неэффективным использованием вычислительных ресурсов. В этом случае можно попробовать оптимизировать код программы, чтобы улучшить ее производительность.

Запуск программы на всех ядрах позволяет оценить, насколько хорошо она масштабируется на многоядерные процессоры. Если программа работает быстрее на всех ядрах, то это означает, что она эффективно использует многоядерную архитектуру процессора.

Сравнение производительности программы с включенным и отключенным *SMT* позволяет оценить, насколько эффективно используется *SMT*. Если программа работает быстрее с включенным *SMT*, то это означает, что *SMT* позволяет программе выполнять больше работы параллельно.

Влияние *SMT* на анализ производительности программы может быть неоднозначным. С одной стороны, *SMT* позволяет программе выполнять больше работы параллельно, что может привести к повышению производительности. С другой стороны, *SMT* может привести к снижению производительности, если программа не использует все доступные ресурсы.

5.1 Описание функциональной схемы программы

На функциональной схеме (приложение А) представлено описание основных компонентов и их взаимодействия в разрабатываемой программе, предназначенной для сравнительного анализа производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в выполнении алгоритма преобразования Фурье.

Как видно из функциональной схемы, программа состоит из следующих шагов:

Инициализация параметров. В начале программы инициализируются параметры, включая длину сигнала и частоту.

Генерация тестовых данных. Программа генерирует сигналы с различными характеристиками, включая амплитуду, частоту и фазу.

Начало отсчета времени выполнения. Программа фиксирует момент начала выполнения алгоритма для последующего измерения времени.

Выполнение преобразования Фурье. Программа применяет алгоритм преобразования Фурье к входным данным для анализа частотных характеристик сигнала.

Остановка таймера. По завершении алгоритма Фурье программа останавливает таймер для подсчета времени выполнения.

Вывод времени, затраченного на выполнение алгоритма. Программа выводит время, которое было затрачено на выполнение алгоритма преобразования Фурье. Это позволяет сравнить производительность алгоритма на разных процессорах.

Эта схема представляет основные этапы работы программы и описывает взаимодействие ее компонентов для проведения сравнительного анализа производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в выполнении алгоритма преобразования Фурье.

5.2 Описание блок-схема алгоритма программы

Программа для сравнения производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в выполнении алгоритма преобразования Фурье имеет структуру, представленную на блок-схеме алгоритма программы (приложение Б).

В коде используется дискретное преобразование Фурье (ДПФ) с прямыми суммами. ДПФ – это математическая операция, которая преобразует сигнал из временной области в частотную область.

Для каждой частоты вычисляется сумма произведений значений сигнала на косинусы и синусы угловых частот $2\pi kn/N$. Полученные суммы представляют собой амплитуды частотных составляющих сигнала.

Благодаря такой структуре программы обеспечивается надежный и точный сравнительный анализ производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* в выполнении алгоритма преобразования Фурье.

5.3 Запуск на всех ядрах с *SMT*

С помощью команды *lscpu | grep -e Socket -e Core -e Thread* терминала *Linux* можно убедиться в том, что ядер в два раза больше, чем физических, из-за наличия двух потоков в каждом ядре (рисунок 5.1, 5.2) [22]:

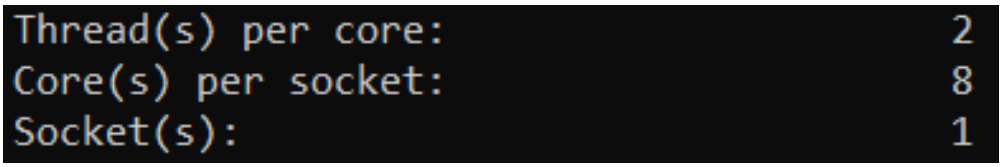

Рисунок 5.1 – Вывод команды для *AMD Ryzen 7 4800H*

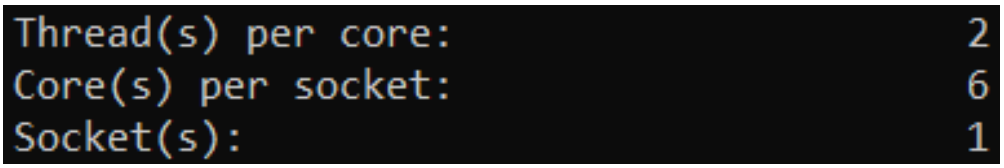

Рисунок 5.2 – Вывод команды для *AMD Ryzen 5 5500U*

Результаты запуска программы на всех ядрах с включенным *SMT* для процессора *AMD Ryzen 5 5500U* отображены в таблице 5.1:

Таблица 5.1 – Время выполнения на всех ядрах с SMT для *AMD Ryzen 5 5500U* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	1,949	1,937	1,938	1,950	1,937	1,942
30000	2,812	2,795	2,800	2,812	2,788	2,801
35000	3,839	3,820	3,818	3,833	3,814	3,824
40000	5,033	5,003	5,009	5,037	4,989	5,014
45000	6,417	6,378	6,386	6,411	6,351	6,388
50000	7,947	7,915	7,917	7,954	7,888	7,924
55000	9,640	9,602	9,610	9,638	9,588	9,615
60000	11,500	11,443	11,455	11,490	11,454	11,468
65000	13,536	13,471	13,475	13,549	13,449	13,496
70000	15,732	15,684	15,674	15,731	15,625	15,689
75000	18,068	17,985	18,017	18,053	18,000	18,024

Результаты запуска программы на всех ядрах с включенным SMT для процессора *AMD Ryzen 7 4800H* отображены в таблице 5.2:

Таблица 5.2 – Время выполнения на всех ядрах с SMT для *AMD Ryzen 7 4800H* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	1,549	1,548	1,555	1,547	1,550	1,549
30000	2,229	2,229	2,238	2,227	2,230	2,230
35000	3,041	3,035	3,047	3,034	3,038	3,039
40000	3,763	3,775	3,765	3,777	3,792	3,774
45000	4,550	4,536	4,565	4,539	4,583	4,554
50000	5,663	5,658	5,682	5,656	5,734	5,678
55000	6,922	6,950	6,917	6,921	6,998	6,941
60000	8,229	8,264	8,254	8,239	8,314	8,260
65000	9,688	9,702	9,684	9,677	9,770	9,704
70000	11,182	11,238	11,205	11,223	11,283	11,226
75000	12,860	12,915	12,901	12,879	12,947	12,900

Для сопоставления значений использовалась формула $\frac{(T_{5500U}-T_{4800H})}{T_{5500U}} * 100\%$, где T_{5500U} – время среднее работы программы для *AMD Ryzen 5 5500U*, T_{4800H} – среднее время работы программы для *AMD Ryzen 5 5500U* (таблица 5.3).

Таблица 5.3 – Коэффициенты эффективности для запуска на всех ядрах с *SMT*

Длина массива	Коэффициент эффективности на заданной длине, %
25000	20,23
30000	20,38
35000	20,52
40000	24,73
45000	28,71
50000	28,34
55000	27,81
60000	27,97
65000	28,09
70000	28,44

Продолжение таблицы 5.3

Длина массива	Коэффициент эффективности на заданной длине, %
75000	28,42
Среднее	25,79

Таким образом, средний коэффициент эффективности составил 25,79%. Графическое представление полученных результатов (график зависимости времени выполнения от длины массива) представлено в графических материалах (рисунок 1 приложение Г).

5.4 Запуск на всех ядрах без *SMT*

Ожидается, что каждое физическое ядро будет выполнять свой набор инструкций и задач, не поддерживая одновременное выполнение нескольких логических потоков на каждом ядре. Таким образом, процессор будет использовать только физические ядра без создания дополнительных виртуальных ядер. Это может повлиять на общую многозадачность и параллельную обработку, поскольку отсутствие логических потоков на физических ядрах может снизить способность процессора эффективно обрабатывать одновременно несколько задач. [23]

С помощью команды *lscpu | grep -e Socket -e Core -e Thread* терминала *Linux* можно убедиться в том, что ядер столько же, сколько и физических, так как теперь каждое ядро действительно имеет только один поток (рисунок 5.3, 5.4):

```
Thread(s) per core:                    1
Core(s) per socket:                    8
Socket(s):                             1
```

Рисунок 5.3 – Вывод команды для *AMD Ryzen 7 4800H*

```
Thread(s) per core:                    1
Core(s) per socket:                    6
Socket(s):                             1
```

Рисунок 5.4 – Вывод команды для *AMD Ryzen 5 5500U*

Результаты запуска программы на всех ядрах с выключенным *SMT* для процессора *AMD Ryzen 5 5500U* отображены в таблице 5.4:

Таблица 5.4 – Время выполнения на всех ядрах без SMT для *AMD Ryzen 5 5500U* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	2,520	2,598	2,562	2,592	2,556	2,565
30000	3,632	3,730	3,692	3,737	3,688	3,695
35000	4,960	5,082	5,048	5,093	5,032	5,043
40000	6,508	6,648	6,613	6,661	6,609	6,607
45000	8,273	8,416	8,386	8,441	8,383	8,379
50000	10,266	10,396	10,375	10,443	10,378	10,371
55000	12,481	12,597	12,581	12,633	12,587	12,575
60000	14,902	14,999	14,996	15,036	15,002	14,987
65000	17,530	17,618	17,615	17,668	17,629	17,612
70000	20,386	20,445	20,454	20,495	20,447	20,445
75000	23,408	23,475	23,478	23,520	23,480	23,472

Результаты запуска программы на всех ядрах с выключенным *SMT* для процессора *AMD Ryzen 7 4800H* отображены в таблице 5.5:

Таблица 5.5 – Время выполнения на всех ядрах без SMT для *AMD Ryzen 7 4800H* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	2,230	2,230	2,230	2,232	2,233	2,231
30000	3,212	3,211	3,210	3,210	3,214	3,211
35000	4,045	4,109	4,109	4,103	4,105	4,094
40000	4,679	4,827	4,837	4,814	4,814	4,794
45000	5,941	6,143	6,163	6,135	6,127	6,101
50000	7,404	7,699	7,709	7,723	7,689	7,644
55000	8,947	9,323	9,325	9,337	9,297	9,245
60000	10,662	11,097	11,115	11,111	11,062	11,009
65000	12,529	13,015	13,016	13,026	12,962	12,909
70000	14,538	15,102	15,084	15,064	15,051	14,967
75000	16,698	17,334	17,322	17,300	17,311	17,193

Коэффициенты эффективности, рассчитанные для данных результатов представлены на таблице 5.6:

Таблица 5.6 – Коэффициенты эффективности для запуска на всех ядрах без *SMT*

Длина массива	Коэффициент эффективности на заданной длине, %
25000	13,02
30000	13,09
35000	18,81
40000	27,44
45000	27,18
50000	26,29
55000	26,48
60000	26,54
65000	26,70
70000	26,79
75000	26,75
Среднее	23,55

Таким образом, средний коэффициент эффективности составил 23,55%. Графическое представление полученных результатов (график зависимости времени выполнения от длины массива) представлено в графических материалах (рисунок 2 приложение Г).

5.5 Запуск на одном ядре с *SMT*

Результаты запуска программы на одном ядре с включенным *SMT* для процессора *AMD Ryzen 5 5500U* отображены в таблице 5.7:

Таблица 5.7 – Время выполнения одном ядре с SMT для *AMD Ryzen 5 5500U* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	11,221	11,219	11,220	11,199	11,199	11,211
30000	16,154	16,154	16,153	16,123	16,122	16,141
35000	21,982	21,986	21,986	21,945	21,954	21,970
40000	28,716	28,711	28,713	28,658	28,664	28,692

Продолжение таблицы 5.7

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
45000	36,342	36,336	36,337	36,266	36,264	36,309
50000	44,869	44,860	44,860	44,786	44,773	44,829
55000	54,294	54,301	54,256	54,193	54,201	54,249
60000	64,610	64,616	64,585	64,482	64,500	64,558
65000	75,829	75,830	75,823	75,668	75,712	75,772
70000	87,972	87,928	87,965	87,772	87,798	87,887
75000	100,954	100,957	100,977	100,756	100,780	100,884

Результаты запуска программы на одном ядре с включенным *SMT* для процессора *AMD Ryzen 7 4800H* отображены в таблице 5.8:

Таблица 5.8 – Время выполнения одном ядре с SMT для *AMD Ryzen 7 4800H* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	10,573	10,571	10,571	10,572	10,572	10,571
30000	15,219	15,216	15,220	15,219	15,219	15,218
35000	20,713	20,710	20,715	20,712	20,712	20,712
40000	27,052	27,049	27,046	27,054	27,048	27,049
45000	34,238	34,239	34,231	34,242	34,238	34,237
50000	42,281	42,285	42,267	42,290	42,270	42,278
55000	51,164	51,167	51,147	51,171	51,154	51,160
60000	60,895	60,893	60,867	60,890	60,872	60,883
65000	71,473	71,461	71,434	71,559	71,433	71,452
70000	82,883	82,876	82,873	82,880	82,884	82,879
75000	95,142	95,133	95,106	95,122	95,139	95,128

Коэффициенты эффективности, рассчитанные для данных результатов представлены на таблице 5.9:

Таблица 5.9 – Коэффициенты эффективности для запуска на одном ядре с *SMT*

Длина массива	Коэффициент эффективности на заданной длине, %
25000	5,70
30000	5,71

Продолжение таблицы 5.9

Длина массива	Коэффициент эффективности на заданной длине, %
35000	5,72
40000	5,72
45000	5,70
50000	5,69
55000	5,69
60000	5,69
65000	5,70
70000	5,69
75000	5,70
Среднее	5,70

Таким образом, средний коэффициент эффективности составил 5,70%. Графическое представление полученных результатов (график зависимости времени выполнения от длины массива) представлено в графических материалах (рисунок 3 приложение Г).

5.6 Запуск на одном ядре без *SMT*

Результаты запуска программы на одном ядре с выключенным *SMT* для процессора *AMD Ryzen 5 5500U* отображены в таблице 5.10:

Таблица 5.10 – Время выполнения на одном ядре без SMT для *AMD Ryzen 5 5500U* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	11,220	11,219	11,222	11,222	11,219	11,220
30000	16,149	16,155	16,147	16,147	16,142	16,150
35000	21,979	21,980	21,971	21,971	21,985	21,977
40000	28,718	28,698	28,704	28,704	28,708	28,706
45000	36,344	36,332	36,334	36,334	36,334	36,331
50000	44,860	44,859	44,863	44,863	44,831	44,855
55000	54,286	54,290	54,284	54,284	54,270	54,282
60000	64,574	64,617	64,602	64,602	64,591	64,597
65000	75,805	75,845	75,819	75,819	75,824	75,822
70000	87,950	87,941	87,976	87,976	87,932	87,955

Продолжение таблицы 5.10

Длина входного массива	Номер тестового запуска					Среднее
	1	2	1	4	5	
75000	100,951	100,969	100,923	100,923	100,923	100,940

Результаты запуска программы на одном ядре с выключенным *SMT* для процессора *AMD Ryzen 7 4800H* отображены в таблице 5.11:

Таблица 5.11 – Время выполнения на одном ядре без SMT для *AMD Ryzen 7 4800H* (секунды)

Длина входного массива	Номер тестового запуска					Среднее
	1	2	3	4	5	
25000	10,572	10,571	10,578	10,575	10,573	10,573
30000	15,220	15,219	15,216	15,221	15,223	15,219
35000	20,712	20,715	20,717	20,716	20,714	20,714
40000	27,049	27,057	27,066	27,059	27,056	27,057
45000	34,237	34,235	34,257	34,240	34,230	34,239
50000	42,281	42,269	42,290	42,270	42,285	42,279
55000	51,178	51,140	51,174	51,160	51,156	51,161
60000	60,892	60,861	60,890	60,861	60,895	60,879
65000	71,450	71,441	71,466	71,445	71,437	71,447
70000	82,852	82,866	82,877	82,850	82,870	82,863
75000	95,143	95,118	95,143	95,144	95,126	95,134

Коэффициенты эффективности, рассчитанные для данных результатов представлены на таблице 5.12:

Таблица 5.12 – Коэффициенты эффективности для запуска на одном ядре без *SMT*

Длина массива	Коэффициент эффективности на заданной длине, %
25000	5,76
30000	5,76
35000	5,74
40000	5,74
45000	5,75
50000	5,74
55000	5,74

Продолжение таблицы 5.12

Длина массива	Коэффициент эффективности на заданной длине, %
60000	5,75
65000	5,77
70000	5,78
75000	5,75
Среднее	5,75

Таким образом, средний коэффициент эффективности составил 5,75%. Графическое представление полученных результатов (график зависимости времени выполнения от длины массива) представлено в графических материалах (рисунок 4 приложение Г).

5.7 Обоснование полученных результатов

Преобразования Фурье являются фундаментальной математической операцией, часто применяемой в анализе сигналов и изображений. В контексте данного исследования, преобразования Фурье используются для анализа временных сигналов и выделения их частотных компонент.

Выбор операционной системы также оказывает влияние на проведение анализа. *Debian* предоставляет среду без графической оболочки, что обеспечивает эффективное окружение для проведения вычислительных задач. Отсутствие графической оболочки уменьшает нагрузку на процессор и обеспечивает стабильную основу для тестирования производительности. Использование *Debian* как операционной системы для сравнения процессоров позволяет минимизировать факторы, влияющие на результаты.

Применение преобразований Фурье на обоих процессорах в *Debian* позволяет оценить их эффективность в обработке вычислительно интенсивных задач. Результаты тестирования на различных конфигурациях, включая различные числа ядер и использование технологии *SMT*, предоставляют комплексную картину о производительности процессоров в различных сценариях использования.

Связь между преобразованиями Фурье и анализом сигналов становится ключевым аспектом при оценке производительности процессоров. Результаты анализа в различных конфигурациях, подкрепленные использованием преобразований Фурье, позволяют сделать обоснованные выводы относительно производительности *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*. Результаты сравнительного анализа процессоров AMD Ryzen 5 *5500U* и *AMD*

Ryzen 7 4800H в контексте выполнения алгоритма преобразования Фурье на одном и нескольких ядрах выявили ряд интересных тенденций, требующих детального обоснования и анализа.

На уровне одного ядра процессора *AMD Ryzen 7 4800H* продемонстрировал небольшое превосходство в скорости выполнения алгоритма преобразования Фурье по сравнению с *AMD Ryzen 5 5500U*. Это может быть обусловлено различиями в архитектуре ядер, частотой работы или оптимизацией под конкретные характеристики *AMD Ryzen 7 4800H*. Процессор *AMD Ryzen 7 4800H* имеет более высокую тактовую частоту на одном ядре, что приводит к небольшому преимуществу в однопоточных задачах.

Однако, более заметное различие наблюдается при использовании нескольких ядер. *AMD Ryzen 7 4800H* значительно опережает *AMD Ryzen 5 5500U*. Это обусловлено тем, что алгоритм преобразования Фурье хорошо параллелизуем, и *AMD Ryzen 7 4800H*, обладая большим количеством физических и логических ядер, более эффективно использует параллелизм для обработки данных.

Важным фактором также является поддержка технологии *Simultaneous Multithreading* (*SMT*). Процессор *AMD Ryzen 7 4800H* с *SMT* может эффективно использовать дополнительные потоки для увеличения параллелизма задач и, таким образом, улучшения производительности. В отличие от этого, *AMD Ryzen 5 5500U* может испытывать ограничения в параллелизации из-за меньшего количества физических и логических ядер.

Общий вывод из анализа заключается в том, что при выполнении вычислительно интенсивных задач, таких как преобразование Фурье, процессор с большим числом ядер и поддержкой *SMT* может продемонстрировать заметное преимущество в многозадачных и параллельных сценариях.

ЗАКЛЮЧЕНИЕ

В рамках исследовательской работы было проведено сравнение производительности процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H*, фокусируясь на их способности обработки преобразований Фурье. Данный метод анализа был выбран как важный в вычислительных задачах, требующих эффективной обработки сигналов.

В ходе исследования были рассмотрены технические характеристики обоих процессоров, включая количество физических и логических ядер, тактовую частоту и возможность использования технологии *Simultaneous Multithreading* (*SMT*). Тестовое окружение было подготовлено для проведения серии вычислительных тестов, используя алгоритм преобразования Фурье.

Особое внимание уделено созданию различных конфигураций работы процессоров, включая использование одного и всех доступных ядер, а также работу с включенной и выключенной технологией *SMT*. Проведено пять вычислительных тестов на различных наборах входных данных для каждой из четырех конфигураций, что позволило вычислить разницу во времени выполнения.

Полученные данные были подвергнуты тщательному анализу, вычислены отклонения во времени выполнения между процессорами для каждой конфигурации. Сравнительные графики были составлены для визуализации эффективности процессоров на различных условиях эксплуатации.

Выводы из проведенного анализа указывают на превосходство процессора *AMD Ryzen 7 4800H* при использовании многопоточности и *SMT*, особенно при обработке преобразований Фурье. Это может быть обусловлено как более высокой частотой ядер, так и большим количеством физических и логических ядер, что способствует эффективной параллелизации вычислительных задач.

Таким образом, результатом выполнения данного курсового проекта стал сравнительный анализ процессоров *AMD Ryzen 5 5500U* и *AMD Ryzen 7 4800H* при выполнении преобразований Фурье в различных конфигурациях, что может поспособствовать более глубокому пониманию архитектуры процессоров.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] AMD Ryzen 7 4800H [Электронный ресурс]. – Режим доступа : https://www.notebookcheck-ru.com/AMD-Ryzen-7-4800H.459436.0.html. – Дата доступа : 15.09.2023.

[2] AMD Ryzen 5 5500U [Электронный ресурс]. – Режим доступа : https://www.notebookcheck-ru.com/AMD-Ryzen-5-5500U.542699.0.html. – Дата доступа : 15.09.2023.

[3] Изменения микроархитектуры в Zen 2 [Электронный ресурс]. – Режим доступа : https://pc-01.tech/zen2-2. – Дата доступа : 20.09.2023.

[4] Основные Характеристики AMD Ryzen 7 4800H [Электронный ресурс]. – Режим доступа : https://www.amd.com/en/product/9081. – Дата доступа : 23.09.2023.

[5] Основные Характеристики AMD Ryzen 5 5500U [Электронный ресурс]. – Режим доступа : https://www.amd.com/en/product/10856. – Дата доступа : 25.09.2023.

[6] What is virtualization? [Электронный ресурс]. – Режим доступа : https://www.ibm.com/topics/virtualization. – Дата доступа : 05.10.2023.

[7] Debian linux live usb [Электронный ресурс]. – Режим доступа : https://uchet-jkh.ru/i/debian-linux-live-usb/. – Дата доступа : 07.10.2023.

[8] Причины выбрать Debian [Электронный ресурс]. – Режим доступа : https://www.debian.org/intro/why_debian/. – Дата доступа : 07.10.2023.

[9] Что такое Debian [Электронный ресурс]. – Режим доступа : https://www.nic.ru/help/chto-takoe-debian_11441. – Дата доступа: 10.10.2023.

[10] Multithreading [Электронный ресурс]. – Режим доступа : https://habr.com/ru/companies/otus/articles/549814/. – Дата доступа : 13.10.2023.

[11] Что такое OpenMP? [Электронный ресурс]. – Режим доступа : https://parallel.ru/tech/tech_dev/openmp.html. – Дата доступа : 13.10.2023.

[12] Документация Visual studio [Электронный ресурс]. – Режим доступа : https://visualstudio.microsoft.com/ru/vs/. – Дата доступа : 14.10.2023.

[13] Преобразование Фурье: самый подробный разбор [Электронный ресурс]. – Режим доступа : https://proglib.io/p/fourier-transform. – Дата доступа : 14.10.2023.

[14] Применение преобразования Фурье в цифровой обработке звука [Электронный ресурс]. – Режим доступа : https://shackmaster.narod.ru/fourier.htm. – Дата доступа : 17.10.2023.

[15] Кому и для чего нужен C++ [Электронный ресурс]. – Режим доступа : https://blog.skillfactory.ru/cplus-komu-i-dlya-chego-nuzhen/. – Дата доступа : 17.10.2023.

[16] Библиотека времени Chrono [Электронный ресурс]. – Режим доступа : http://inf-w.ru/?p=8305. – Дата доступа : 17.10.2023.

[17] Compiler Optimizations [Электронный ресурс]. – Режим доступа : https://medium.com/@guannan.shen.ai/compiler-optimizations-46db19221947. – Дата доступа : 19.10.2023.

[18] Логическое ядро процессора [Электронный ресурс]. – Режим доступа : https://uchet-jkh.ru/i/logiceskoe-yadro-processora-cto-eto-i-kak-ono-rabotaet/. – Дата доступа : 21.10.2023.

[19] Технологии многопоточности процессоров [Электронный ресурс]. – Режим доступа : https://club.dns-shop.ru/blog/t-100-protsessoryi/30066-tehnologii-mnogopotochnosti-protsessorov-printsip-rabotyi-i-sferyi-p/. – Дата доступа : 21.10.2023.

[20] Алгоритмы преобразования Фурье [Электронный ресурс]. – Режим доступа : https://moluch.ru/archive/124/34105/. – Дата доступа : 25.10.2023.

[21] Дискретное преобразование Фурье [Электронный ресурс]. – Режим доступа : https://digteh.ru/dsp/DFT/. – Дата доступа : 29.10.2023.

[22] 9 команд для проверки информации о CPU в Linux [Электронный ресурс]. – Режим доступа : https://habr.com/ru/companies/otus/articles/581796/. – Дата доступа : 05.11.2023.

[23] Как отключить SMT на RYZEN [Электронный ресурс]. – Режим доступа : https://te4h.ru/kak-otklyuchit-smt-na-ryzen. – Дата доступа : 10.11.2023.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
#include <vector>
#include <cmath>
#include <iostream>
#include <chrono>
#include <omp.h>

typedef std::vector<double> RealVector;
std::vector<int> testValues { 10000, 15000, 20000, 25000, 30000, 35000,
40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000 };

void dft_parallel(const RealVector& x, RealVector& X, int num_threads) {
    int N = x.size();
    X.resize(N);

#pragma omp parallel for num_threads(num_threads)
    for (int k = 0; k < N; k++) {
        double realPart = 0;
        double imagPart = 0;

#pragma omp parallel for reduction(+:realPart, imagPart)
num_threads(num_threads)
        for (int n = 0; n < N; n++) {
            double angle = 2 * M_PI * k * n / N;
            realPart += x[n] * cos(angle);
            imagPart += x[n] * sin(angle);
        }

        X[k] = sqrt(realPart * realPart + imagPart * imagPart);
    }
}

int main() {
    int n = 0;
    int num_threads = 1;

    std::cout << "Choose the number of threads (1 for single-threaded, " <<
omp_get_max_threads() << " for all threads): ";
    std::cin >> num_threads;

    for (int i = 0; i < testValues.size(); i++)
    {
        n = testValues[i];

        // Генерация массива для тестирования
        RealVector input;
        for (int i = 0; i < n; i++) {
            input.push_back(sin(2 * M_PI * i / 10)); // Пример: синусоида с
частотой 0.1
        }

        // Измерение времени выполнения
        auto start = std::chrono::steady_clock::now();

        // Применение распараллеленного преобразования Фурье с аффинитетом
        RealVector output;
```

```
        dft_parallel(input, output, num_threads);

        auto end = std::chrono::steady_clock::now();
        auto diff = end - start;

        // Вывод времени выполнения
        std::cout << n << " : " << num_threads << " : " <<
std::chrono::duration <double, std::milli>(diff).count()  << " ms" <<
std::endl;
    }

    return 0;
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Функциональная схема алгоритма, реализующего программное средство

ПРИЛОЖЕНИЕ В

(обязательное)

Блок схема алгоритма, реализующего программное средство

ПРИЛОЖЕНИЕ Г

(обязательное)

Сравнительные графики времени выполнения программы

ПРИЛОЖЕНИЕ Д
(обязательное)
Ведомость документов