Федеральное агенство по образованию Российской Федерации
Московский Государственный Технический Университет
имени Н.Э.Баумана

Факультет «Робототехника и комплексная автоматизация»
Кафедра «Системы Автоматизированного Проектирования»

Пояснительная записка
к дипломному проекту на тему:

Инструментарий статистического анализа эффективности исполнения программ.

Студент–дипломник _____Панков М.К.

Научный руководитель _____Карпенко А.П.

Рецензент _____XXX

Зав. кафедрой РК-6 _____Карпенко А.П.

Москва,
2013

Оглавление

Введение

Разработка современного математического обеспечения САПР — сложная задача, требующая больших вложений материальных и временных ресурсов. Зачастую для достижения оптимальности по таким критериям, как производительность, объём занимаемой памяти, и энергопотребление требуется ручная настройка компилятора — выбор вариантов тонких настроек оптимизации под определённую программно-аппаратную платформу с учётом особенностей решаемой задачи. Это плохо формализованная задача, которая часто решается разработчиками МО методом проб и ошибок. В результате этого оптимальность часто не достигается ни по одному из выбранных критериев.

Распространённой проблемой для исследователей и инженеров является то, что параметры компиляции подбираются под конкретный набор входных данных, на конкретной программно-аппаратной платформе [1, 2]. При этом для набора данных, встречающегося в реальных задачах (а не используемого во время подбора оптимальных параметров), оптимальные параметры могут оказаться иными.

Для увеличения возможностей по тонкой оптимизации математического программного обеспечения для рабочих станций и суперкомпьютеров необходима формализация поисковой области. С помощью построения модели производительности программ возможно достичь лучшего понимания воздействия оптимизаций компилятора на интересующие разработчика критерии эффективности программы. Таким образом можно сделать поиск оптимальных настроек более направленным и локализованным, сокращая цикл разработки математического обеспечения, стоимость разработки и поддержки.

Цель — разработать систему сбора, систематизации, формализации данных о производительности компилируемых программ в зависимости от настроек компилятора и программно-аппаратной платформы, а также выполняющую функции поддержки базы знаний и обучения с целью моделирования и предсказания эффективности программ.

Примером критерия эффективности является производительность программы на данной программно-аппаратной платформе.

Конструкторская часть

1 Статистический анализ быстродействия программ

1.1 Технологии компиляторов

Современный оптимизирующий промышленный компилятор — сложная программа, состоящая из нескольких модулей, типично выделяемых в две части [3], описанные ниже.

1) Анализирующая часть.

   1.1) Лексический анализатор. Распознаёт строки («лексемы») и формирует из них типизированные значения («токены»).

   1.2) Синтаксический анализатор. Потребляет токены и анализирует грамматическую правильность сформированных из них конструкций.

   1.3) Семантический анализатор. Производит семантический анализ — оценивает правильность и осмысленность сформированных из токенов конструкций.

2) Синтезирующая часть.

   2.1) Генератор абстрактного синтаксического дерева. Производит построение абстрактного синтаксического дерева из токенов. Дерево отражает связи между различными элементами исходного файла, но уже не привязано к их строковому представлению. Оно содержит множество типизированных узлов со связями между ними.

   2.2) Генератор промежуточного представления. Создаёт промежуточное представление (такое как трёхадресный код или форма с статическим единственным присваиванием) из абстрактного синтаксического дерева. Также может конвертировать одно промежуточное представление в другое для удобства произведения оптимизаций.

2.3) Оптимизатор. Производит трансформацию промежуточного представления с сохранением семантики и увеличением эффективности. Типичными оптимизациями являются уменьшение сложности операции и разворачивание циклов. Обычно использует набор эвристик, определяющих правила применения оптимизаций и их параметры, т.к. многие оптимизации имеют настройку того или иного рода, определяющие её агрессивность.

2.4) Генератор исполняемого файла. Производит окончательное создание исполняемого файла в двоичном формате: перевод промежуточного представления в набор инструкций целевой машины, оптимизации на объектном файле и связывание объектных файлов в исполняемый.

Стоит отметить, что данное деление является довольно условным: так, синтаксический анализатор, семантический анализатор и генератор абстрактного синтаксического дерева типично объединяются в одну единицу, называемую «парсер». Однако, оно довольно показательно с точки зрения объёма и разнообразия задач, решаемых компилятором.

1.2 Сложность разработки компиляторов

Разработка современного оптимизирующего промышленного компилятора — сложная задача, требующая огромных вложений человеческих ресурсов. Больша́я часть этой сложности обусловлена необходимостью тонкой настройки эвристических методов оптимизации исполняемого кода (которые в общем случае решаются NP-сложными алгоритмами), производимой вручную путём просмотра генерируемого кода и состояний промежуточного представления компилятора. Не говоря о том, что само по себе создание таких эвристик представляет собой зачастую нетривиальную задачу [4–6], эвристики часто имеют противоположные цели — например, одни оптимизации увеличивают производительность кода ценой размера исполняемого файла (например, разворачивание циклов), а другие уменьшают размер кода, уменьшая производительность. Помимо этого, в компиляторах чаще всего используются упрощённые модели аппаратного обеспечения, не учитывающие многие эффекты даже первого порядка, не говоря уже о побочных эффектах. Более того, сами правила обнаружения случаев, в которых можно оптимизировать код, создаются человеком, что также трудоёмко и является далеко не исчерпывающим способом увеличения производительности результирующего кода. Помимо этого, от порядка произведения оптимизаций часто зависит итоговая производительность и размер кода исполняемого файла.

В то же время, у настроек эвристик есть фокальные точки — такие настройки оптимизации, при которых достигается оптимум по какому-либо критерию. Эти фокальные точки могут быть найдены с помощью функции цели. Примером задачи, для которой существуют достаточно хорошие целевые функции, является задача распределения регистров.

В существующих компиляторах также стоит проблема подбора оптимальных настроек компиляции для данной конкретной программы. Чаще всего проще ограничиться стандартной настройкой «максимальная производительность», нежели производить трудоёмкий анализ кода, который так же требует квалификации опытного разработчика компилятора и знания о платформе, для которой генерируется исполняемый файл. Эта задача также может решаться автоматически и, в ряде случаев, более эффективно.

1.3 Подходы к решению проблемы сложности эвристик

Попытки настраивать и даже синтезировать [7, 8] эвристики предпринимались неоднократно. Как уже показали работы [9–12], задача подбора самих эвристик и их настройки может решаться автоматически.

Принципиально, классическим подходом к этой задаче является итеративная компиляция. Также, в последнее время получили распространение методы машинного обучения и коллективной оптимизации. Далее мы рассмотрим эти подходы.

1.4 Итеративная компиляция

Итеративная компиляция — методика поиска оптимальных по некоторому критерию настроек компилятора, при которой компилируемая программа собирается много раз с одним набором данных на одной и той же программно-аппаратной платформе с разными, выбираемыми, как правило, случайным образом, настройками, с последующим запуском и замером интересующих факторов во время её выполнения (например, времени выполнения). Очевидно, что такой подход страдает от чудовищной неэффективности — поиск оптимальных настроек для одной программы может занимать месяц машинного времени. Количество необходимых компиляций составляет десятки и сотни запусков [13]. Кроме того, такой поиск является весьма ограниченным, поскольку найденные настройки в итоге подходят только для данного набора данных на данной платформе с данным компилятором (вплоть до конкретной версии), и практически никаких общих рекомендаций извлечь из этого процесса не получается.

Неоднократно производились попытки ускорить этот процесс с помощью применения различных методов оптимизации, изменяющих характер поиска настроек с случайного на более направленный. В этом случае задача

осложняется чрезвычайно большой многомерностью такой оптимизации — признаков, по которым опознаются программы и которые можно считать аргументами функции эффективности, насчитываются десятки (имеются ввиду такие признаки, собираемые с исходного кода программы, как средняя глубина вложенности циклов, среднее число вызовов процедур на блок и т.д.). Таким образом, данный способ решения является крайне плохо приспособленным для реального применения.

1.5 Машинное обучение

Недавние работы в данной области показали возможность увеличить эффективность итеративной компиляции с помощью машинного обучения, позволяя обеспечить компилятор возможностью подстраиваться под сложные архитектуры и повысить его производительность относительно классических статических компиляторов — т.е. таких, которые не были специально модифицированы для работы с инструментарием машинного обучения [14, 15].

Далее для понимания связи машинного обучения с компиляцией приведена классическая постановка задачи этого раздела науки.

Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества.

В нашем случае объектом является исходный код программы с его признаками (такими, как средний размер массивов и среднее число итераций циклов), а ответом — значение интересующего нас фактора во время исполнения программы (например, объём занимаемой памяти).

Стоит отметить, что предпочтительным является обучение с использованием только производственных запусков компилятора, т.е. без специальных обучающих запусков. Для этого предлагается использовать коллективную оптимизацию, позволяющую пользователям обмениваться данными об успешной оптимизации и не проделывать избыточную работу каждый раз изолированно.

1.6 Коллективная оптимизация

Коллективная оптимизация предполагает реализацию некого инструментария, подключаемого к компилятору с целью организации

его прозрачной работы с общим репозиторием настроек компиляции. Этот подход является многообещающим, но не лишён сложностей.

А именно, перед реализацией коллективной оптимизации стоит несколько проблем исследовательского и инженерного характера. Инженерная проблема состоит в создании системы, позволяющей прозрачно переиспользовать накопленные другими пользователями знания о произведенных компиляциях и отправлять данные о них без дополнительных действий со стороны конечного пользователя компилятора. Частью решения этой проблемы является плагин к компилятору GCC, предоставляющий возможности произвольного включения выбранных проходов компилятора и обеспечивающий его взаимодействие с репозиторием кода cTuning.org [16]. В случае более современного компилятора llvm [17] плагин имеет более простую организацию, поскольку данный инструмент имеет встроенные возможности произвольного выбора набора производимых оптимизаций, и нужно обеспечить только взаимодействие с репозиторием cTuning. Исследовательская проблема состоит в изучении воздействия различных оптимизаций на различные факторы получаемых программ, такие, как время исполнения и объём используемой памяти.

Коллективная оптимизация может обеспечить эффективность итеративной компиляции без произведения такого большого количества избыточных запусков компилятора. Одним из успешно применяемых подходов к этой задаче предполагает статистическое сравнение эффективности пар наборов оптимизаций. Недавние работы также обнаружили значительно бóльшую важность обучения на различных наборах данных и на различных архитектурах, нежели обучения на различных программах [18]. Это опровергает доминировавшую до этого точку зрения на применение машинного обучения в компиляторах и предлагает новые перспективы в области автоматической настройки компиляции.

1.7 Выводы

Существующие методы статистического анализа быстродействия программ плохо применимы на практике ввиду отсутствия удобного инструментария для проведения исследований и большого времени, требуемого для проведения запусков программ.

Наиболее перспективными подходами к решению задачи анализа быстродействия являются машинное обучение предсказателей производительности и коллективное принятие решений в сфере анализа и оптимизации быстродействия.

2 Методология моделирования, применяемая в системе Adaptor

2.1 Статистические методы обработки информации

2.1.1 Регрессионный анализ

К регрессионному анализу относятся задачи выявления искажённой случайным «шумом» функциональной зависимости интересующего исследователя показателя Y от измеряемых переменных $X_1, ..., X_m$. Данными служит таблица экспериментально полученных «зашумлённых» значений Y на разных наборах $x_1, ..., x_m$ [19].

Цели регрессионного анализа. Цели проведения регрессионного анализа приведены ниже.

— Определение степени детерминированности вариации критериальной (зависимой) переменной предикторами (независимыми переменными).

— Предсказание значения зависимой переменной с помощью независимой(-ых).

— Определение вклада отдельных независимых переменных в вариацию зависимой.

Регрессионный анализ нельзя использовать для определения наличия связи между переменными, поскольку наличие такой связи и есть предпосылка для применения анализа.

Математическое определение регрессии. Строго регрессионную зависимость можно определить следующим образом. Пусть $Y, X_1, X_2, ..., X_p$ — случайные величины с заданным совместным распределением вероятностей. Если для каждого набора значений $X_1 = x_1, X_2 = x_2, ..., X_p = x_p$ определено условное математическое ожидание

$$y(x_1, x_2, ..., x_p) = E(Y|X_1 = x_1, X_2 = x_2, ..., X_p = x_p)$$

(уравнение регрессии в общем виде), то функция $y(x_1, x_2, ..., x_p)$ называется регрессией величины Y по величинам $X_1, X_2, ..., X_p$, а её график — линией регрессии Y по $X_1, X_2, ..., X_p$, или уравнением регрессии [20]. Зависимость Y от $X_1, X_2, ..., X_p$ проявляется в изменении средних значений Y при изменении $X_1, X_2, ..., X_p$. Хотя при каждом фиксированном наборе значений $X_1, X_2, ..., X_p$ величина Y остаётся случайной величиной с определённым рассеянием. Для выяснения вопроса, насколько точно регрессионный анализ оценивает изменение Y при изменении $X_1, X_2, ..., X_p$, используется

средняя величина дисперсии Y при разных наборах значений $X_1, X_2, ..., X_p$ (фактически речь идет о мере рассеяния зависимой переменной вокруг линии регрессии).

Метод наименьших квадратов (расчёт коэффициентов). На практике линия регрессии чаще всего ищется в виде линейной функции $Y = b_0 + b_1X_1 + b_2X_2 + ... + b_NX_N$ (линейная регрессия), наилучшим образом приближающей искомую кривую [21]. Делается это с помощью метода наименьших квадратов, когда минимизируется сумма квадратов отклонений реально наблюдаемых Y от их оценок $\hat{Y}$ (имеются в виду оценки с помощью прямой линии, претендующей на то, чтобы представлять искомую регрессионную зависимость)

$$\sum_{k=1}^{M} (Y_k - \hat{Y}_k)^2 \to min,$$

где M — объём выборки. Этот подход основан на том известном факте, что фигурирующая в приведённом выражении сумма принимает минимальное значение именно для того случая, когда $Y = y(x_1, x_2, ..., x_N)$. Для решения задачи регрессионного анализа методом наименьших квадратов вводится понятие функции невязки

$$\sigma(\bar{b}) = \frac{1}{2} \sum_{k=1}^{M} (Y_k - \hat{Y}_k)^2.$$

Условие минимума функции невязки выражается системой $N + 1$ линейных уравнений с $N + 1$ неизвестными $b_0...b_n$. В итоге мы получаем получаем матричное уравнение: $A \times X = B$, которое легко решается методом Гаусса. Получаем матрицу, содержащую коэффициенты уравнения линии регрессии:

$$X = \begin{Bmatrix} b_0 \\ b_1 \\ ... \\ b_N \end{Bmatrix}.$$

Для получения наилучших оценок необходимо выполнение предпосылок МНК (условий Гаусса-Маркова). В англоязычной литературе такие оценки называются *BLUE* (*Best Linear Unbiased Estimators*) — наилучшие линейные несмещенные оценки.

Интерпретация параметров регрессии. Параметры b_i являются частными коэффициентами корреляции; $(b_i)^2$ интерпретируется как доля дисперсии Y, объяснённая X_i, при закреплении влияния остальных предикторов, то есть измеряет индивидуальный вклад в объяснение Y. В

случае коррелирующих предикторов возникает проблема неопределённости в оценках, которые становятся зависимыми от порядка включения предикторов в модель. В таких случаях необходимо применение методов анализа корреляционного и пошагового регрессионного анализа [22]. Говоря о нелинейных моделях регрессионного анализа, важно обращать внимание на то, идет ли речь о нелинейности по независимым переменным (с формальной точки зрения легко сводящейся к линейной регрессии), или о нелинейности по оцениваемым параметрам (вызывающей серьёзные вычислительные трудности). При нелинейности первого вида с содержательной точки зрения важно выделять появление в модели членов вида X_1X_2, $X_1X_2X_3$, свидетельствующее о наличии взаимодействий между признаками X_1, X_2 и т. д.

2.1.2 Перекрёстная проверка

Перекрёстная проверка — метод формирования обучающего и тестового множеств для обучения аналитической модели в условиях недостаточности исходных данных или неравномерного представления классов. Для успешного обучения аналитической модели необходимо, чтобы классы были представлены в обучающем множестве примерно в одинаковой пропорции. Однако, если данных недостаточно или процедура сэмплинга при формировании обучающего множества была произведена неудачно, один из классов может оказаться доминирующим. Это может вызвать «перекос» в процессе обучения, и доминирующий класс будет рассматриваться как наиболее вероятный. Метод перекрестной проверки позволяет избежать этого [23].

В его основе лежит разделение исходного множества данных на k примерно равных блоков, например $k = 5$. Затем на $k - 1$, т.е. на 4-х блоках, производится обучение модели, а 5-й блок используется для тестирования. Процедура повторяется k раз, при этом на каждом проходе для проверки выбирается новый блок, а обучение производится на оставшихся.

Перекрестная проверка имеет два основных преимущества перед применением одного множества для обучения и одного для тестирования модели. Во-первых, распределение классов оказывается более равномерным, что улучшает качество обучения. Во-вторых, если при каждом проходе оценить выходную ошибку модели и усреднить ее по всем проходам, то полученная ее оценка будет более достоверной. На практике чаще всего выбирается $k = 10$ (10–проходная перекрестная проверка), когда модель обучается на 9/10 данных и тестируется на 1/10. Исследования показали, что в этом случае получается наиболее достоверная оценка выходной ошибки модели.

2.2 Метод моделирования эффективности исполнения программ *Velocitas*

Метод *Velocitas* удовлетворяет требованиям к практически применимому решению задачи моделирования быстродействия программ. Основные положения метода описаны ниже.

В первую очередь устанавливается, какие атрибуты объектов в наборе данных являются признаками (входными параметрами), а какой — откликом (выходным параметром). Примерами признаков являются характеристики аппаратного обеспечения, на котором запускалась программа, а отклика — время исполнения программы. Выбор выходного параметра, то есть критерия эффективности исполнения программы, из присутствующих в наборе данных, осуществляется исследователем.

После этого создаются дополнительные признаки из уже присутствующих в наборе данных. Например, может быть создан признак «размер входных данных программы», определяемый как произведение числа строк и столбцов в матрице, обрабатываемой программой. Число и содержание дополнительных признаков определяется исследователем на основании личного опыта.

Затем производится фильтрация данных с целью удаления заведомо некорректных измерений. Так, шумом являются все опыты, в результате которых для времени исполнения программы получено неадекватно маленькое значение — менее 0,001 с (это минимально измеримое системой время исполнения). Критерии фильтрации также определяются исследователем. Обычно это является элементарной задачей.

Следующим этапом является выполнение ранжирования признаков, присутствующих в наборе данных. Ранжирование выполняется с помощью метода *Earth Importance* [24], который является свободной реализацией метода *MARS — Multivariate Adaptive Regression Splines*. С помощью метода производится оценка важности признаков, после чего для использования при обучении выбираются только признаки с ненулевой важностью.

Построение модели заканчивается выполнением обучения регресионного предсказателя на базе *k Nearest Neighbours*. Квази-оптимальным значением параметра является $k = 30$, однако оно может быть и во многих случаях должно быть настроено индивидуально с учётом имеющегося набора данных. Это можно выполнить с помощью оптимизации гиперпараметров путём сеточного поиска (англ. *grid search*, [25]).

2.3 Выводы

Предложен метод моделирования быстродействия программ на различных программно-аппаратных платформах при различных входных

данных. Он является удобным для практического применения в инструментарии моделирования и отличается высокими показателями эффективности (см. раздел 4).

Технологическая часть

3 Программная реализация инструментария *Adaptor*

3.1 Выбор языка программирования

3.1.1 Рассмотрение альтернатив

Ниже приведены возможные варианты выбора языка, разделённые по классам, и оценка их применимости в разработке системы.

— Относительно низкоуровневые компилируемые языки (*C*, *C++*). Применимость этих языков в основном ограничена сложностью и низкой скоростью разработки на них в связи с недостаточно высоким уровнем абстрактности этих языков. Высокая производительность, которой можно достичь с помощью них, не является серьёзным преимуществом, поскольку реализация работающей системы на таких языках занимает значительно большее время, делая разговор о конечной производительности преждевременным.

— Более высокоуровневые компилируемые языки (*Java*, *C#*). Эти языки работают на виртуальных машинах, что ограничивает их переносимость. В случае *C#*, полная современная реализация существует только для *Windows*, что является неприемлемым в связи с тем, что конечная система должна работать и на *Unix*–подобных операционных системах. Ещё одним недостатком можно назвать их компилируемость, поскольку это опять же снижает скорость разработки и значительно усложняет реализацию интерактивного окружения для проведения экспериментов.

— Современные императивные интерпретируемые языки (*Ruby*, *Python*). Эти языки позволяют вести разработку с высокой скоростью и хорошо приспособлены для разработки через тестирование (*Test-Driven Development*, [26]). Они хорошо переносимы и позволяют легко реализовать интерактивное окружение для работы с системой.

Недостатком является относительно низкая скорость выполнения, что компенсируется более высокой скоростью разработки.

Другие семейства языков не рассматриваются ввиду низкой популярности. Это обусловлено тем, что язык реализации системы должен быть знаком большому количеству специалистов — так будет проще вести разработку.

3.1.2 Итоговое решение

В качестве основного языка реализации системы был выбран *Python* [27]. Это простой и удобный, популярный и хорошо поддерживаемый язык с большим количеством библиотек. Он более распространён [28] и имеет большее количество библиотек, чем *Ruby*. Он хорошо поддерживает современные методики разработки программ, такие, как разработка через тестирование [26], за счёт встроенных модулей для работы с автоматическими тестами.

В случае необходимости можно достигнуть высокой производительности с помощью интерпретатора *PyPy* [29]. Для реализации сложной обработки данных можно использовать модуль *scipy* [30] и для визуализации графиков — модуль *matplotlib* [31]. Для реализации статистической обработки данных и машинного обучения можно воспользоваться одним из нескольких популярных модулей, среди которых *Orange* [32] и *sklearn* [33].

3.2 Репозиторий исходных кодов экспериментальных программ

Инструментарий выполняет эксперименты по запуску программ, исходный код которых доступен экспериментатору. Экспериментатору должна быть доступна большая база исходных кодов для обеспечения возможности производить эксперименты над разнообразными программами, среди которых должны быть и похожие между собой. Последнее требование нужно ввиду необходимости обнаруживать похожесть программ в их поведении при изменении аппаратного обеспечения и настроек сборки.

В качестве репозитория предлагается использовать сервер распределённой системы контроля версий [34]. Такая архитектура позволяет удобно работать с локальными копиями репозитория при необходимости произвести незначительные изменения для проверки какой-либо гипотезы относительно оптимизационного поведения программ. Она также позволяет отправлять локальные изменения на центральный сервер, что полезно для обмена исходными кодами между экспериментаторами. В отличие от централизованной системы контроля версий, распределённая система позволяет удалённым участникам разработки легко включиться в

процесс развития системы (во многом за счёт более простого и мощного процесса ветвления версий кода). Кроме того, она позволяет использовать для размещения репозитория бесплатные службы вроде *Github* [35] и *Bitbucket* [36], что устраняет необходимость в выделенном сервере системы контроля версий.

3.2.1 Рассмотрение альтернатив

Далее рассматриваются основные распределённые системы контроля версий с оценкой их применимости в разработке системы. Основным источником является веб-страница [37].

— *Mercurial*. Эта система контроля версий более проста в изучении и лучше поддерживает Windows. Вместе с тем, она в большей степени ограничивает разработчика — например, не позволяет переписывать историю изменений.

— *Git*. Эта система версионирования является более мощной, чем *Mercurial* и позволяет производить многие необходимые в процессе работы действия более быстро и удобно.

3.2.2 Итоговое решение

В связи с большой гибкостью и удобством для разработчика, в качестве репозитория исходных кодов был выбран *Git*.

Стоит отметить, что на данный момент репозиторий исходных кодов тестовых программ является частью общего репозитория исходного кода описываемой системы. Этим также частично обусловлен выбор в пользу *Git*, поскольку исторически исходный код инструментария находился в *Git*. Также *Git* более знаком автору.

3.3 База данных экспериментов

Для хранения данных об экспериментах необходима база данных. Традиционная реляционная база данных плохо подходит на эту роль, поскольку она требует наличия жёсткой схемы, которой следуют все таблицы, и все записи должны иметь одинаковый формат. Это представляет проблему в случае плохо определённой предметной области (такой, как наша исследовательская задача статистического анализа производительности программ), поскольку заранее неизвестно, какие свойства сущностей, сохраняемых в БД, являются важными, а какие — нет. Это приводит к тому, что структура БД часто меняется по мере необходимости введения новых свойств, например, при добавлении в модель нового признака эксперимента по запуску программы.

Рассмотрим документо-ориентированные базы данных. Они позволяют хранить документы в каком-либо текстовом формате (обычно *JSON* [38]) и не требуют одинаковой структуры всех документов. Поля документов могут добавляться и удаляться непосредственно по запросу пользователя. Это свойство особенно важно для нашей задачи. Также эти БД лучше масштабируются (т. е. лучше приспособлены к использованию в крупных распределённых системах).

3.3.1 Рассмотрение альтернатив

Далее приведен список документо-ориентированных БД с оценкой их применимости в нашей задаче. Основой для сравнения служит источник [39].

— *MongoDB*. Это хранилище во многом похоже на реляционные БД. Оно использует свой бинарный протокол передачи данных и ориентировано на получение высокой производительности.

— *CouchDB*. Это хранилище акцентировано на целостности хранимых данных и простоте в использовании. Оно поддерживает двустороннюю репликацию данных и представления данных через применение и свёртку (англ. map-reduce [40]). Также имеется возможность построения приложения на *JavaScript*, которое обеспечивает весь необходимый функционал БД (например, просмотр документов с определёнными свойствами). Хорошо подходит для ситуаций, когда данные накапливаются, но не изменяются.

— *Redis*. Хранилище в памяти для быстро-изменяющихся данных с протоколом, похожем на *Telnet*.

Другие альтернативы (*HBase*, *Cassandra*) не рассматриваются ввиду их специализации на экстремально больших объёмах данных (типичных для банков и поисковых машин) и не лучшей поддержки интерфейса к языку *Python*.

3.3.2 Итоговое решение

В итоге в качестве БД была выбрана *CouchDB*, поскольку она хорошо подходит для сильно распределённых систем (что важно для обеспечения работы отдельных исследователей в различных организациях) и накопления редко меняющихся данных. Поскольку в нашем случае эксперимент по оптимизации программы по определению не может быть изменён после проведения, это подходящий выбор. Это хранилище также удобно в разработке — оно позволяет удобно организовать все функции для доступа к данным в виде отдельного приложения на диске — так называемого

CouchApp [41]. Это позволяет хранить исходный код для работы с БД вместе с остальным исходным кодом инструментария.

3.4 Установка инструментария *Adaptor* на компьютер пользователя

В рекомендуемом окружении — ОС *Ubuntu 12.04* на платформе *x86* или *x86-64* — установка производится в полу-автоматическом режиме посредством использования системного пакетного менеджера `apt-get`.

Шаги, выполняемые в процессе установки в общем случае на любой платформе, приведены ниже. Шаги, отмеченные символом *, не являются необходимыми при выполнении установки в рекомендуемом окружении для проведения исследований.

1) *Установить интерпретатор *Python* версии не ниже 2.7.

2) Установить *CouchDB* версии не ниже 1.1.

3) Установить *CouchApp*.

4) *Настроить переменные окружения и пути для использования установленных программ.

3.5 Расширяемость инструментария *Adaptor*

Инструментарий *Adaptor* обеспечивает расширяемость в смысле возможности добавления новых модулей сборки, запуска и анализа данных.

Стоит отметить базовую поддержку сценариев исследований. Сценарий — это набор действий вида «собрать программу», «запустить программу с измерением времени», «построить график зависимости времени от компилятора» и т. д. Он может задаваться интерактивно или в исходном файле в виде функции языка программирования *Python* с использованием *API* инструментария в том же модуле, что и основной код системы, или в отдельном модуле (предпочтительно последнее). Система также предоставляет интерфейс для использования её в качестве модуля языка *Python* сторонними пользователями. Полный набор доступных для использования в сценариях действий на данный момент описан только в исходном коде, однако это не является проблемой по следующим причинам. Во-первых, исходный код системы достаточно хорошо прокомментирован. Во-вторых, при использовании интерпретатора *ipython* для работы с инструментарием, из комментариев и исходного кода генерируется минимальная документация.

3.6 Платформа для использования инструментария *Adaptor*

Инструментарий *Adaptor* протестирован в ОС *Linux* на *x86*-совместимом процессоре ввиду распространённости данной аппаратной платформы, а также удобства разработки и ведения исследований под Linux.

Система использует функции, специфичные для ОС *Linux*, в ограниченном объёме. Для полнофункциональной работы инструментария необходимо портирование модуля сбора данных об аппаратном обеспечении. Все остальные модули системы независимы от ОС, на которой используется система, и должны также работать под *Windows* или *Mac*. Тестирование инструментария на этих ОС не производилось.

3.7 Архитектура инструметария *Adaptor*

Инструментарий *Adaptor* состоит из нескольких основных компонентов. Они перечислены ниже.

- Компонент загрузки и установки основных переменных. Этот компонент осуществляет инициализацию инструментария и выполняет получение базового пути к каталогу системы, соединяется с базой данных и передаёт управление определённому сценарию исследования или пользователю.
- Компонент поддержания структуры рабочих путей. Этот компонент работает со стеком путей и позволяет пользователю переходить к подкаталогам инструментария различными способами:
  - от корневого каталога, в котором расположен инструментарий;
  - от корневого каталога, в котором расположен репозиторий исходных кодов исследуемых программ;
  - по абсолютному пути в локальной операционной системе.
- Компонент подготовки команд для сборки и запуска программ. Он осуществляет шаблонную подстановку заданных пользователем значений в заготовки команд.
- Компонент запуска подготовленных команд и обмена данными с запущенным процессом. Он осуществляет запуск в контролируемом окружении, измерение времени работы и приём-передачу потоков стандартного ввода, вывода и ошибок.
- Компонент калибровки измерения времени. Его описание можно найти ниже в подразделе с соответствующим названием.

— Компонент сравнения измеренного времени с заведомо известным. Осуществляет вычисление ошибки измерения.

— Компонент создания документов базы данных из полученных в рамках эксперимента сведений. Документы имеют формат JSON [38] и создаются из внутренних структур данных инструментария для возможности последующего сохранения в базу данных.

Все компоненты реализованы в виде функций языка *Python*. Применение объектной модели на данный момент не необходимо и перегружает код ненужными абстракциями.

3.8 Эксперимент по оптимизации программы

Эксперимент состоит в следующем.

1) Система собирает программу с определёнными настройками сборки (см. ниже).

2) Система запускает программу в контролируемом окружении с определёнными настройками запуска и производит измерение интересующих метрик исполнения программы. Список метрик см. ниже.

   На данный момент нас будет интересовать полное время исполнения. Список настроек исполнения см. ниже.

3) Система сохраняет данные о сборке и запуске программы в базу данных для последующего анализа и обработки.

В случае компилятора *gcc* настройки сборки включают в себя [42]:

— компилятор (команда для запуска);

— базовый уровень оптимизации (флаг `-O[n]`, где `n` — уровень оптимизации);

— набор флагов тонких настроек оптимизации (флаги семейства `-f[name]`, где `name` — имя определённого набора оптимизаций). Некоторые из этих флагов также имеют числовые параметры;

— путь к заголовочным файлам (опция `-I`);

— путь к исходным файлам;

— параметры определения макросов (флаги вида `-D[MACRO]`, где `MACRO` — имя определяемого макроса);

— путь к исполняемому файлу, производимому компилятором.

Метрики исполнения могут быть следующими:

— полное время исполнения программы;

— время исполнения программы по функциям;

— полный объём памяти, занимаемой программой и данными;

— объём памяти, занимаемой кодом программы;

Настройки исполнения содержат:

— путь к исполняемому файлу;

— источник ввода для стандартного потока ввода программы (перенаправление `stdin`);

— потоки вывода для стандартного потока вывода и стандартного потока ошибок программы (перенаправление `stdout` и `stderr` соответственно);

— аргументы запуска, в т.ч. путь к обрабатываемому файлу данных (например, путь к файлу изображения для программы сжатия изображений).

3.8.1 Время исполнения программы

Точное измерение времени исполнения программы может представлять сложности ввиду возможных колебаний из-за изменения загрузки системы другими задачами. Принципиально, рекомендуется выполнять эксперименты на системе, не занятой другими задачами. Однако даже в таком случае системные процессы или разница в решениях, принятых планировщиком процессов, могут оказать значительное воздействие на измерение времени. Подробнее о механизмах, применяемых в планировщиках процессов в современных ОС и о возможном воздействии на время выполнения задачи, смотри источник [43].

Для устранения описанных проблем необходимо производить несколько запусков программы. Программа запускается 3 раза и берётся минимальное время её выполнения. Это время будет наиболее точно отражать производительность программно-аппаратной платформы, поскольку увеличение времени выполнения происходит в связи с интерференцией данного процесса с другими и общим состоянием системы. Как показали эксперименты (результаты в этой работе не приводятся), воздействие кэширования на производительность программы из тестового набора

при многократном запуске при текущей реализации инструментария не наблюдается.

В случае внешнего измерения времени исполнения влияние оказывают также накладные расходы, связанные с запуском программы из системы. Для их исключения измерять время исполнения лучше изнутри программы и выводить его, например, в стандартный поток ошибок. Таким образом измерение времени производится в пакете тестирования производительности *Polybench* [44].

Внутреннее измерение времени исполнения обеспечивает большую точность, однако, оно требует поддержки на уровне исходного кода. Очевидно, что для универсального инструментария запуска оптимизационных экспериментов такое решение неприемлемо, поскольку должно быть возможно измерение времени выполнения программ, никак специально не подготовленных для этого.

По этой причине система измеряет время исполнения извне. Невысокая точность измерения при этом устраняется с помощью калибровки (см. ниже). При этом также стоит отметить, что погрешность, вносимая внешним измерением, является систематической и не оказывает влияния на относительное время выполнения разных версий программы, что является наиболее важным в данной задаче.

3.8.2 Калибровка времени исполнения программы

Помимо отмеченных сложностей, есть сложность измерения маленьких интервалов времени. Если время исполнения программы находится на уровне 1 мс (таково временное разрешение системного вызова *Unix* `gettimeofday`), то даже если запускать программу несколько раз, колебания измерений окажутся настолько большими, что сделают результат неточным. Для устранения этой проблемы предлагается специальный алгоритм калибровки, схема которого приведена ниже. Он также позволяет уменьшить влияние колебаний времени исполнения.

1) Положить $n = 0, t = 0, d_{rel} = 1$.

2) Исполнить программу однократно и измерить время её выполнения.

3) Если время выполнения более 1 секунды, положить $n = -1$.

4) Производить следующий цикл, пока $t < 1$ и $d_{rel} > 0,05$. Иначе перейти к шагу 9.

5) Увеличить n на 1: $n = n + 1$.

6) Вычислить число запусков программы как $number = 10^n$.

7) Запустить программу *number* раз, измеряя время выполнения. Повторить это 3 раза. Среди результатов выбрать минимум и присвоить его t. Вычислить относительную дисперсию результатов и присвоить её величине d_{rel}.

8) Перейти к шагу 4.

9) Получить время исполнения программы как $t/number$.

В ряде случаев также используется сравнение измеренного времени исполнения с «реальным». Хотя на практике сложно измерить время реального исполнения (т.е. за исключением времени запуска), время запуска можно вычислить, запустив программу, которая ничего не делает, и измерив время её исполнения. Затем можно вносить поправку на время запуска в каждое измерение (для устранения систематической методической погрешности). Проблема этого решения в том, что требуется очень точное измерение времени запуска пустой программы, но при применении тех же методов это принципиально невозможно — время её исполнения также будет сильно колебаться в зависимости от загрузки системы и практически случайного воздействия планировщика задач в текущем состоянии системы.

Оценку предложенных методов измерения времени можно произвести путём сравнения измеренного времени исполнения с временем, заданным таймером с помощью вызова функции `usleep` стандартной библиотеки языка *C* изнутри программы.

Стоит отметить, что существуют постоянные расходы на запуск программы из нашей системы и их можно вычесть из измеренного времени для повышения точности измерения. Для этого мы вычисляем время выполнения пустой программы, а затем вычитаем его из каждого измеренного результата выполнения реальных программ.

3.8.3 Интерактивное окружение проведения экспериментов

Для удобной работы с инструментарием рекомендуется использовать интерпретатор *ipython* [45]. Он предоставляет графическую оболочку языка *Python*, которая позволяет строить графики в том же окне, сохранять сессии использования интерпретатора и имеет хорошую поддержку автодополнения команд и документации. На рисунке 3.1 приведён снимок экрана этой графической оболочки с встроенным графиком.

3.9 Конвейер обработки данных в системе Orange

На рисунке 3.2 изображён конвейер статистической обработки данных в системе *Orange*. Здесь он приведён целиком, в уменьшенном виде, для понимания общего расположения компонентов.

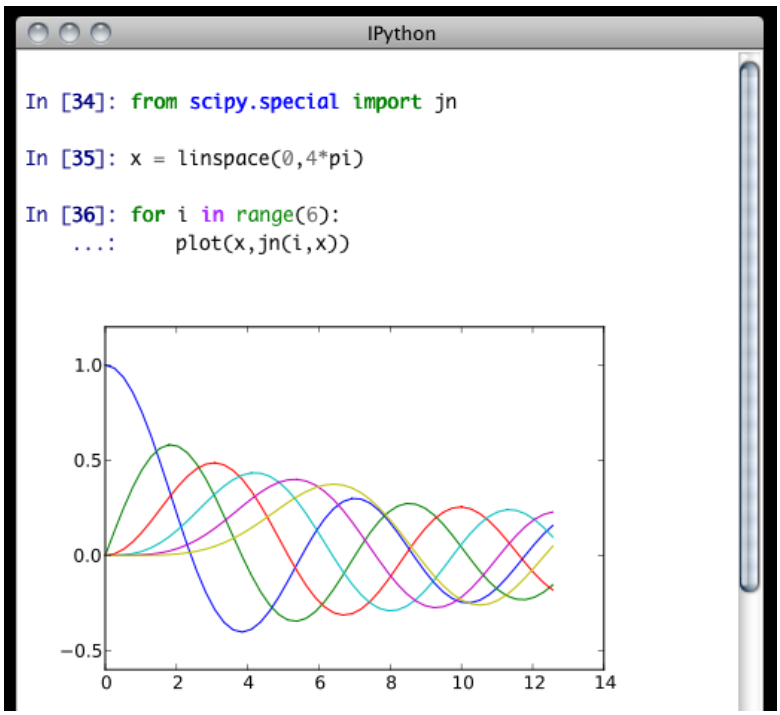


Рисунок 3.1 – Снимок экрана графической оболочки *ipython* с встроенным графиком

Конвейер имеет такой вид для всех трёх серий экспериментов. Далее мы рассмотрим части конвейера по отдельности и опишем каждый компонент, используемый в схеме анализа данных, изображённой на указанном выше рисунке.

Первая часть конвейера — предварительная обработка данных перед построением модели. Она изображена на рисунке 3.3.

Обработка данных в системе начинается с чтения входных данных в формате *CSV* из файла — за это отвечает компонент `1. File`, находящийся в левом верхнем углу схемы. Отчёт по этому компоненту представлен на рисунке 3.4. Опишем формат входного файла.

Входной файл содержит таблицу, в которой столбцы — это названия свойств эксперимента, полученных во время его выполнения в инструментарии, а строки — собственно значения этих свойств. Приведём отрывок файла для пояснения его структуры (рисунок 3.5). Многоточия означают опущенные части файла — поскольку число свойств достигает сорока, представить файл полностью не представляется возможным ввиду ограниченной ширины страницы. Свойство `id` также представлено в сокращённом виде — символы `..` означают опущенные части строки.

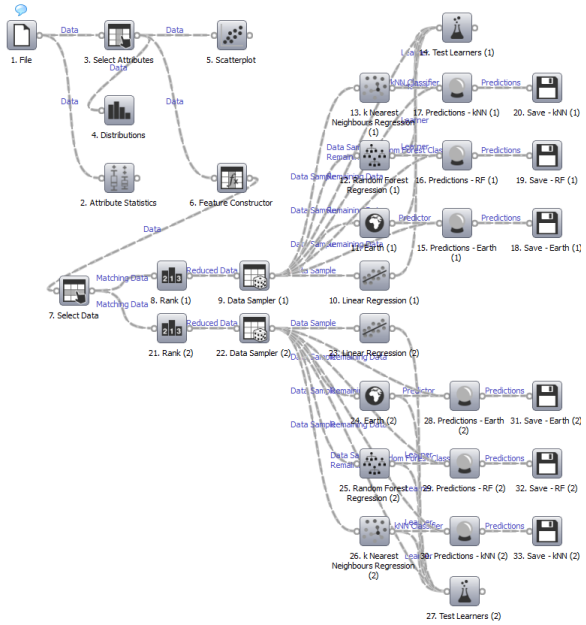


Рисунок 3.2 – Конвейер обработки данных в системе *Orange*

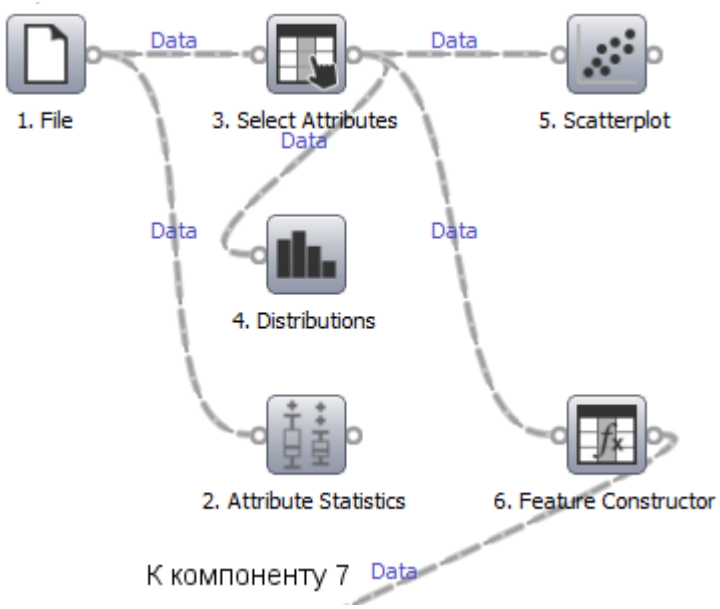


Рисунок 3.3 – Конвейер обработки данных в системе *Orange*. Часть 1 — предварительная обработка

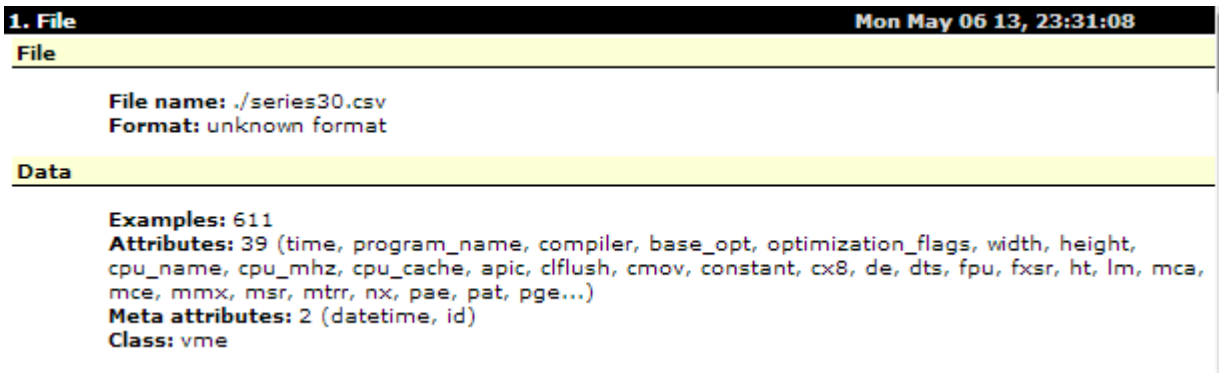


Рисунок 3.4 – Компонент `1. File`. Входные данные в файле *CSV*

```
id datetime time program_name compiler base_opt width height cpu_name cpu_mhz cpu_cache ...
51..16 2013-04-17 22:25:29 0.0041661978 symm gcc None 64 64 Intel(R) Xeon(R) ... 2666.76 6144 ...
51..6d 2013-04-17 22:25:14 1.3440570831 symm gcc None 256 256 Intel(R) Xeon(R) ... 2666.76 6144 ...
51..0b 2013-04-17 22:24:57 0.121064496 symm gcc None 128 128 Intel(R) Xeon(R) ... 2666.76 6144 ...
```

Рисунок 3.5 – Отрывок входного файла

Итак, первая строка файла — названия свойств. `id` — уникальный идентификатор эксперимента, `datetime` — время его проведения в GMT, `time` — время исполнения программы, `program_name` — название программы, `compiler` — название используемого компилятора, `base_opt` — базовый уровень оптимизации программы компилятором, `optimization_flags` — дополнительные настройки оптимизации, `width` — число столбцов в обрабатываемой матрице, `height` — число строк в обрабатываемой матрице, `cpu_name` — название процессора, на котором исполнялась программа, `cpu_mhz` — частота процессора, `cpu_cache` — объём кэша третьего уровня. Все остальные свойства, начиная с `apic` и заканчивая `vme` — двоичные свойства наличия у процессора поддержки определённой возможности, такой как, например, набора инструкций *SSE*.

Отметим, что пунктирные линии на схеме обозначают передачу данных из одного компонента обработки данных в другой. Схема представляет собой дерево с корнем в узле `1. File`, причём данные передаются от корня к листьям. Таким образом, из узлов, которые не имеют дочерних, информация дальше не передаётся. Если у узла несколько дочерних узлов, данные передаются от родителя каждому ребёнку данного узла.

Компонент `2. Attribute Statistics` используется для сбора и показа статистических показателей различных признаков экспериментов, содержащихся в наборе данных – например, таких, как среднее и медианное значение признака. Выводимые им изображения можно найти в Приложении.

Компонент `3. Select Attributes` (рисунок 3.6) применяется с целью установления структуры набора данных — в нём определяется, какие свойства экспериментов являются исходными данными для моделирования (признаками), а какие — выходными данными (предсказанными значениями). В данном случае все свойства экспериментов, кроме свойства `time` (время исполнения программы), являются признаками. Свойство `time` предсказывается моделью.

Компонент `4. Distributions` отображает распределения различных атрибутов набора данных. Эти графики можно найти в Приложении.

Компонент `5. Scatterplot` строит точечные графики зависимости времени исполнения программы от других свойств эксперимента. Пример такого графика можно найти в Приложении. На нём завершается предварительный анализ входных данных.

Компонент `6. Feature Constructor` используется для создания дополнительного свойства эксперимента из уже присутствующих в наборе данных — свойства `size`, определяемого как $size = height \cdot width$ и имеющего смысл агрегированного размера обрабатываемой программой `symm` матрицы.

Компонент `7. Select Data` (рисунок 3.7) может использоваться для фильтрации данных по различным признакам. Например, он может быть использован для удаления из набора данных всех экспериментов, для

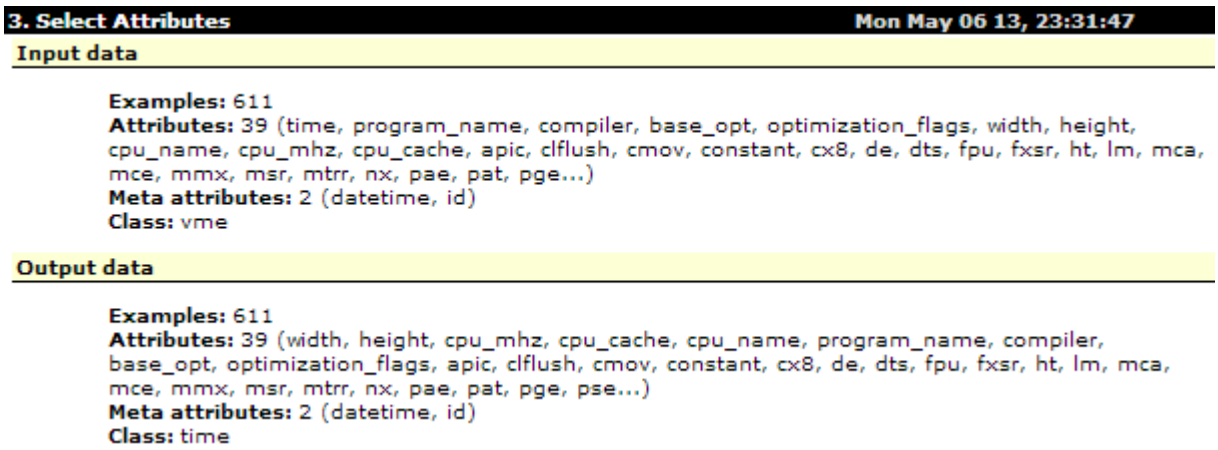


Рисунок 3.6 – Компонент `3. Select Attributes`. Определение структуры входных данных

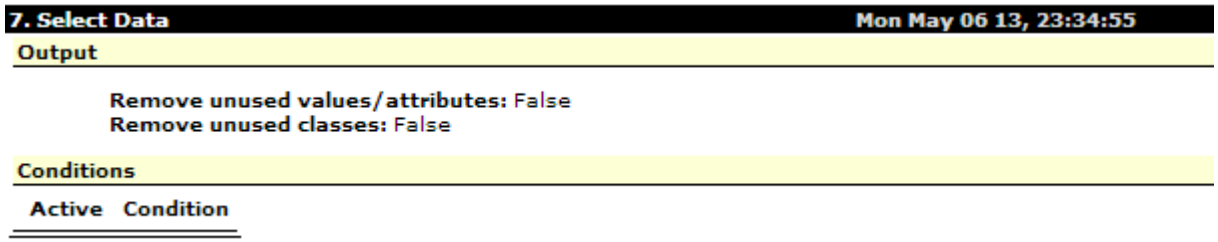


Рисунок 3.7 – Компонент `7. Select Data`. Выбор данных из набора

которых измеренное время исполнения оказалось менее 0,001 с вследствие артефактов измерения. Такая фильтрация может увеличить качество предсказания, поскольку результаты, которые очевидно являются шумом, удаляются из набора данных. Однако, в конечных версиях рассмотренных моделей данный компонент не был задействован, поскольку он требует тонкой настройки и снижает степень автоматизации моделирования.

На этом предварительная обработка данных завершается и начинается собственно построение модели. Компоненты под номерами с 8 по 20 и с 20 по 33 полностью аналогичны. Они представляют собой две ветви конвейера, в одной из которых (с компонентами 8–20) используется простейшая модель на основе единственного признака, а в другой (с компонентами 20–33) — более сложная модель на основе четырёх или пяти признаков.

Опишем обе ветви конвейера, начиная с части, отвечающей за ранжирование признаков и выборку объектов. Эта часть схемы изображена на рисунке 3.8.

Компонент `8. Rank (1)` (рисунок 3.9) и его аналог `21. Rank (2)` выбирает S наиболее значимых признаков набора данных, при этом $S = 1$ для компонента 8 и $S = 4$ или $S = 5$ для компонента 21. Выбор $S = 4$ или $S = 5$ зависит от результата работы алгоритма выделения важных признаков и для первой серии экспериментов $S = 4$, а для второй и третьей —

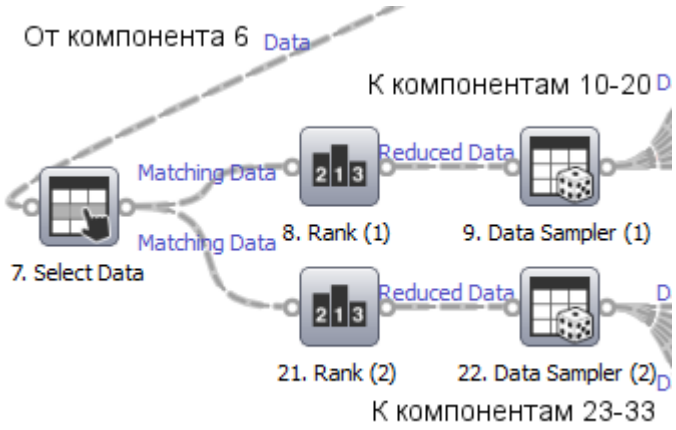


Рисунок 3.8 – Конвейер обработки данных в системе *Orange*. Часть 2 — ранжирование признаков и выборка объектов

$S = 5$. Согласно рассуждениям, приведённым ниже в разделе 3.9, в качестве алгоритма ранжирования признаков во всех случаях используется алгоритм *Earth*.

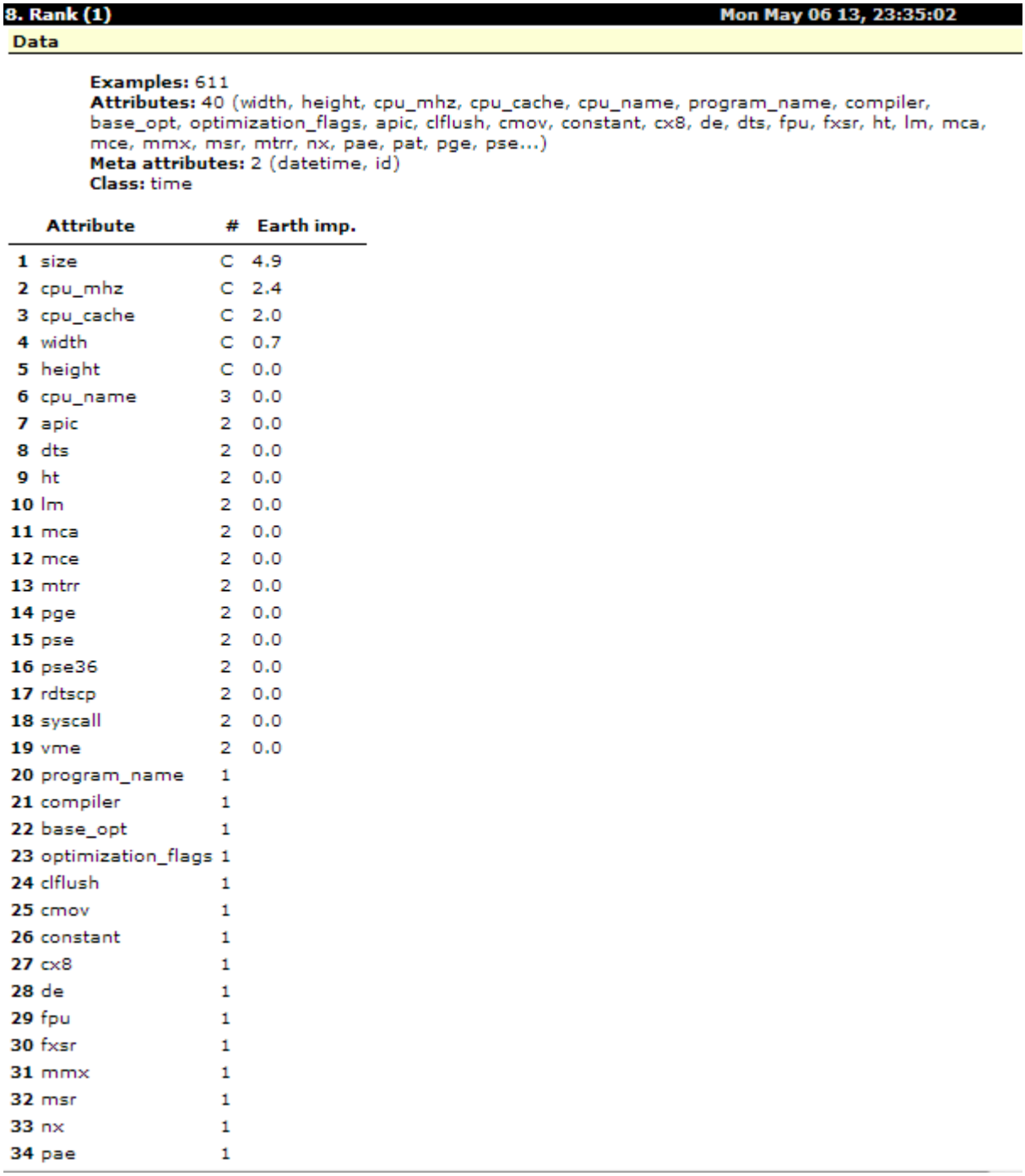
Остановимся подробнее на выборе алгоритма ранжирования признаков.

Алгоритм ранжирования признаков. В системе статистической обработки данных Orange доступны следующие алгоритмы ранжирования признаков модели: *Relief F, MSE, Earth Importance* и *Random Forest Importance*. В ходе изучения третьей серии экспериментов были получены результаты ранжирования признаков, показанные на рисунке 3.10.

Как можно видеть, алгоритм *Relief F* показал результаты, целиком отбрасывающие влияние свойств `cpu_mhz` и `cpu_cache`, а также всех остальных особенностей аппаратной платформы. Это заведомо неверно, поскольку известно, что характеристики аппаратного обеспечения влияют на производительность программы. Этот алгоритм выделил только два самых очевидных свойства, оказывающих влияние на производительность.

Алгоритм *MSE* также присвоил наибольшие веса свойствам `width` и `height`, а на третье место по важности поставил `cpu_cache`, что выглядит разумно. Однако, данный алгоритм присвоил одинаковые веса многим признакам, которые просто коррелируют с изменением свойств `cpu_mhz` и `cpu_cache`, но не обнаружил корреляции – таким образом, свойства `apic, dts` и другие получили вес $0,05159...$. При этом истинно важный признак `cpu_mhz` был оценён важностью, меньшей чем у всех этих коррелирующих признаков, что свидетельствует о невозможности применения этого алгоритма для нашей задачи.

Алгоритм *Earth Importance* выявил все четыре признака, которые являются важными в нашей модели – число строк и столбцов в обрабатываемой матрице, частоту используемого процессора и объём его кэша. При этом «паразитные» коррелирующие признаки не были оценены



8. Rank (1) **Mon May 06 13, 23:35:02**

Data

Examples: 611
Attributes: 40 (width, height, cpu_mhz, cpu_cache, cpu_name, program_name, compiler, base_opt, optimization_flags, apic, clflush, cmov, constant, cx8, de, dts, fpu, fxsr, ht, lm, mca, mce, mmx, msr, mtrr, nx, pae, pat, pge, pse...)
Meta attributes: 2 (datetime, id)
Class: time

	Attribute	#	Earth imp.
1	size	C	4.9
2	cpu_mhz	C	2.4
3	cpu_cache	C	2.0
4	width	C	0.7
5	height	C	0.0
6	cpu_name	3	0.0
7	apic	2	0.0
8	dts	2	0.0
9	ht	2	0.0
10	lm	2	0.0
11	mca	2	0.0
12	mce	2	0.0
13	mtrr	2	0.0
14	pge	2	0.0
15	pse	2	0.0
16	pse36	2	0.0
17	rdtscp	2	0.0
18	syscall	2	0.0
19	vme	2	0.0
20	program_name	1	
21	compiler	1	
22	base_opt	1	
23	optimization_flags	1	
24	clflush	1	
25	cmov	1	
26	constant	1	
27	cx8	1	
28	de	1	
29	fpu	1	
30	fxsr	1	
31	mmx	1	
32	msr	1	
33	nx	1	
34	pae	1	

Рисунок 3.9 – Компонент `8. Rank (1)`. Выбор признаков

	Attribute	#	ReliefF	MSE	Earth imp.
1	width	C	8.4053840637207	0.21186138689518	5.2
2	height	C	15.1900749206543	0.0682036653161049	4.5
3	cpu_mhz	C	0	0.0341536588966846	1.2
4	cpu_cache	C	0	0.0585388652980328	0.2
5	apic	2	0	0.0515984818339348	0.0
6	dts	2	0	0.0515984818339348	0.0
7	ht	2	0	0.0515984818339348	0.0
8	lm	2	0	0.0341536588966846	0.0
9	mca	2	0	0.0515984818339348	0.0
10	mce	2	0	0.0515984818339348	0.0
11	mtrr	2	0	0.0515984818339348	0.0
12	pge	2	0	0.0515984818339348	0.0
13	pse	2	0	0.0515984818339348	0.0
14	pse36	2	0	0.0515984818339348	0.0
15	rdtscp	2	0	0.00259591685608029	0.0
16	syscall	2	0	0.0515984818339348	0.0
17	vme	2	0	0.0515984818339348	0.0
18	clflush	1	0	0	
19	cmov	1	0	0	
20	constant	1	0	0	
21	cx8	1	0	0	
22	de	1	0	0	
23	fpu	1	0	0	
24	fxsr	1	0	0	
25	mmx	1	0	0	
26	msr	1	0	0	
27	nx	1	0	0	
28	pae	1	0	0	
29	pat	1	0	0	
30	sep	1	0	0	
31	ss	1	0	0	
32	sse	1	0	0	
33	sse2	1	0	0	
34	tsc	1	0	0	

Рисунок 3.10 – Сравнение алгоритмов ранжирования признаков модели

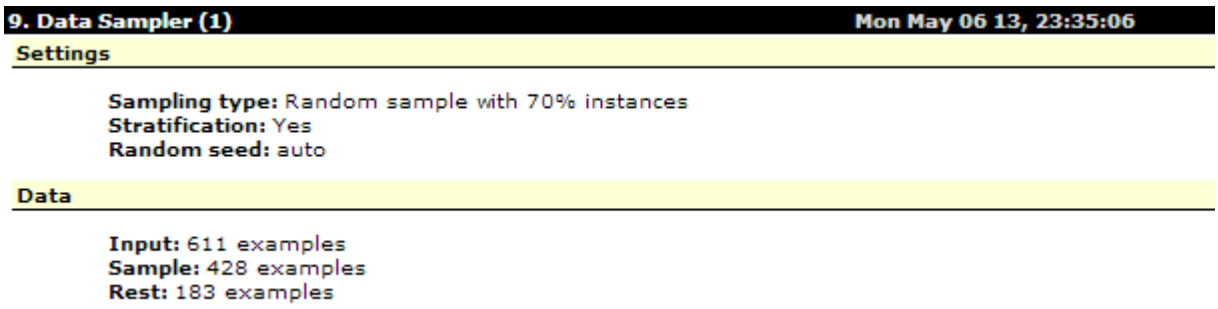


Рисунок 3.11 – Компонент `9. Data Sampler (1)`. Отбор объектов в обучающую и проверочную выборки

алгоритмом как важные — таким образом мы избавляемся от лишних свойств в модели и делаем её минимально необходимой для адекватного описания предметной области.

Алгоритм *Random Forest Importance* отработал на нашем наборе данных некорректно, вызвав программную ошибку в системе Orange. Он был исключён из дальнейшего рассмотрения.

Таким образом, мы выбираем *Earth Importance* в качестве алгоритма ранжирования признаков.

Теперь продолжим рассмотрение конвейера обработки данных в системе *Orange*.

Компоненты `9. Data Sampler (1)` и `22. Data Sampler (2)` производят случайный отбор 70% данных из набора в учебный набор, а остальных 30% — в проверочный набор данных. Это необходимо для проведения перекрёстной проверки, которая в данном случае является одно-проходной ввиду ограниченных ресурсов времени и невысокой степени автоматизации осуществления перекрёстной проверки в системе *Orange*. При этом компонент 9 работает с моделью, построенной на основе единственного признака, который был оценён, как наиболее важный, компонентом 8. Компонент 22 работает с моделью на основе четырёх или пяти признаков, наиболее важных по результатам работы компонента 21. Далее мы не будем останавливаться на различиях моделей в ветвях (8–20) и (20–33).

На этом вторая часть схемы заканчивается и мы переходим собственно к обучению предсказателей для двух моделей — простейшей (с одним признаком) и более сложной (с четырьмя или пятью признаками).

На рисунках 3.12 и 3.13 представлены две последние части конвейера обработки данных в *Orange*. Они полностью аналогичны и отличаются тем, что первая (описывающая компоненты 10–20) получает данные от компонента 9, а вторая (компоненты 23–33) — от компонента 22. Это данные, включающие в себя один признак эксперимента и несколько (четыре или пять) признаков эксперимента, соответственно.

Компоненты `10. Linear Regression (1)` и `23. Linear Regression (2)`

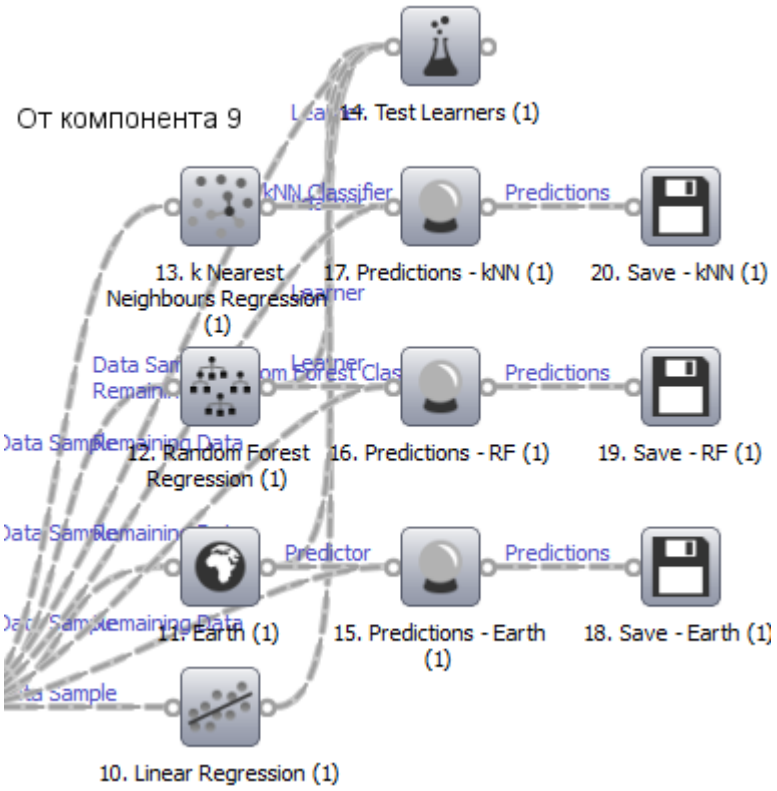


Рисунок 3.12 – Конвейер обработки данных в системе *Orange*. Часть 3 — построение простейшей модели

строят модели линейной регрессии на основе экспериментальных данных. Эти модели являются образцовыми. Это значит, что качество результатов данных моделей является нижней границей для качества результатов любых других моделей. Поскольку линейная регрессия является простейшим предсказателем, рассмотрение моделей, которые дают результаты хуже, чем линейная регрессия, не имеет смысла. Модели из компонентов 10 и 23 передаются в компоненты 14 и 27 соответственно, где те сравниваются с другими предсказателями по нескольким метрикам (см. ниже).

Компоненты `11. Earth (1)`, `12. Random Forest Regression (1)`, `13. k Nearest Neighbours Regression (1)` и `24. Earth (2)`, `25. Random Forest Regression (2)`, `26. k Nearest Neighbours Regression (2)` являются модулями построения предсказателей *Earth*, *Random Forest* и *kNN* для ветвей (8–20) и (20–33) соответственно.

После обучения предсказателей их параметры передаются в компоненты сравнения качества результатов `14. Test Learners (1)` и `27. Test Learners (2)`. Кроме этого, они используются для построения предсказаний в компонентах `15. Predictions - Earth (1)`, `16. Predictions - RF (1)`, `17. Predictions - kNN (1)` и 28. Predictions - Earth (2), 29. Predictions - RF (2), 30. Predictions - kNN (2) соответсвенно.

После построения предсказаний, они сохраняются в файлах *CSV*

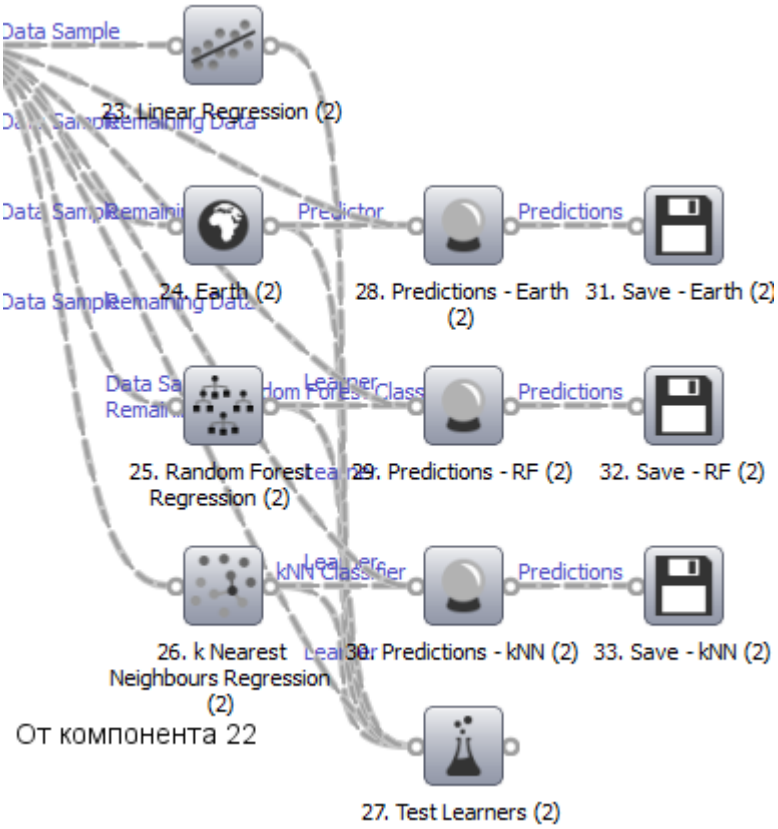


Рисунок 3.13 – Конвейер обработки данных в системе *Orange*. Часть 4 — построение более сложной модели

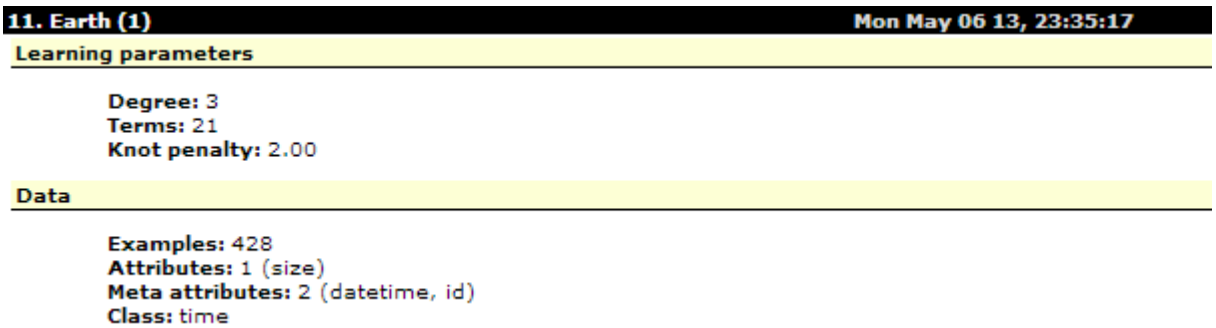


Рисунок 3.14 – Компонент `11. Earth (1)`. Предсказатель *Earth* для модели с одним признаком

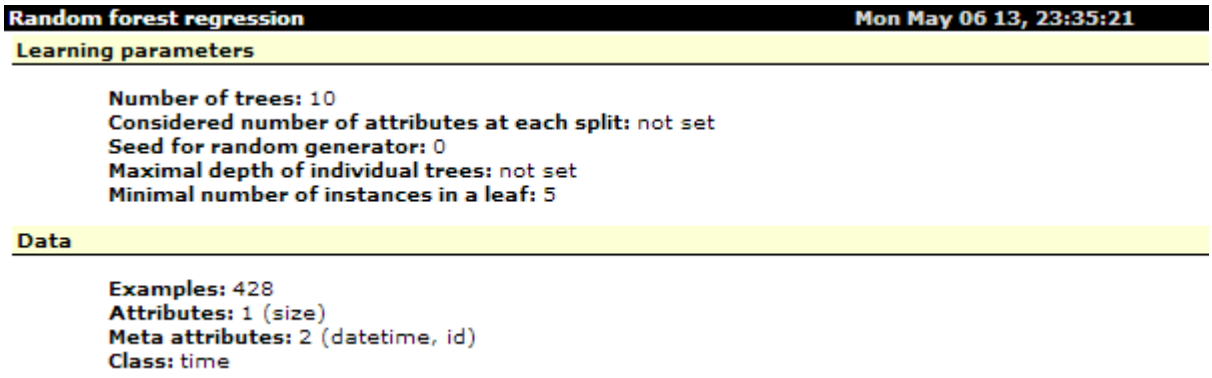


Рисунок 3.15 – Компонент `12. Random Forest (1)`. Предсказатель *Random Forest* для модели с одним признаком

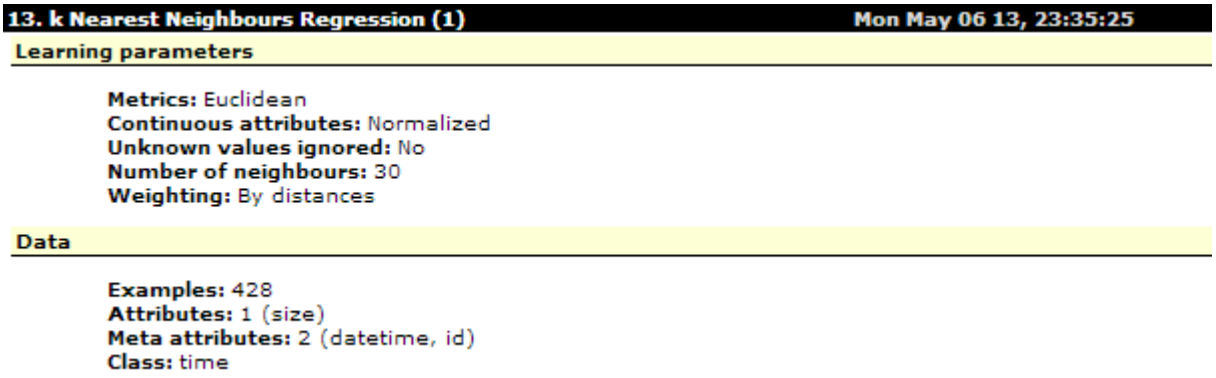


Рисунок 3.16 – Компонент `13. kNN (1)`. Предсказатель *k Nearest Neighbours* для модели с одним признаком

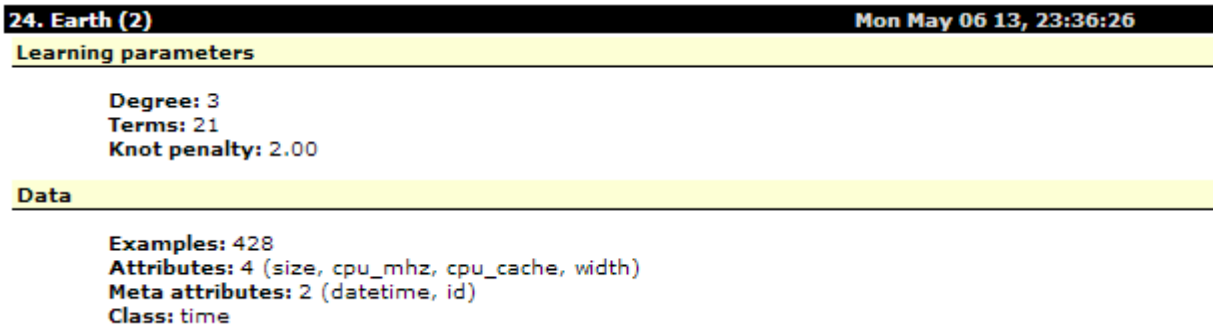


Рисунок 3.17 – Компонент `24. Earth (2)`. Предсказатель *Earth* для модели с четырьмя признаками

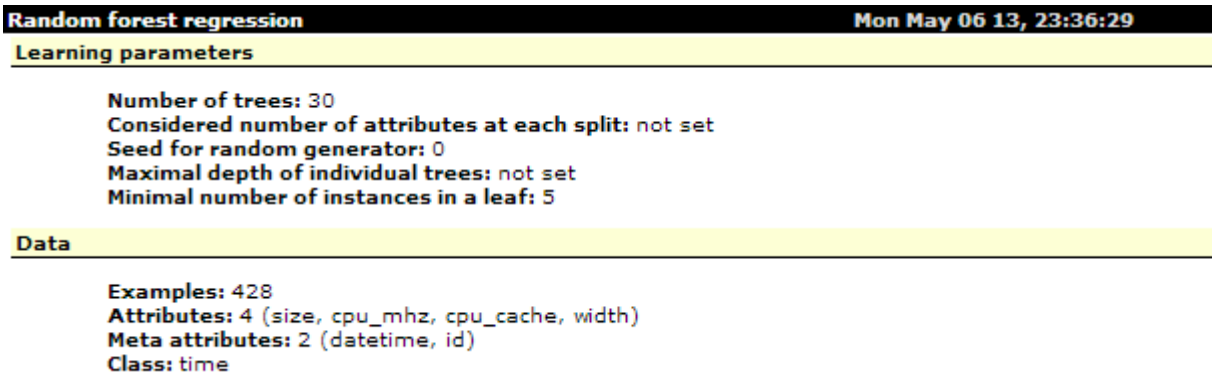


Рисунок 3.18 – Компонент `25. Random Forest (2)`. Предсказатель *Random Forest* для модели с четырьмя признаками

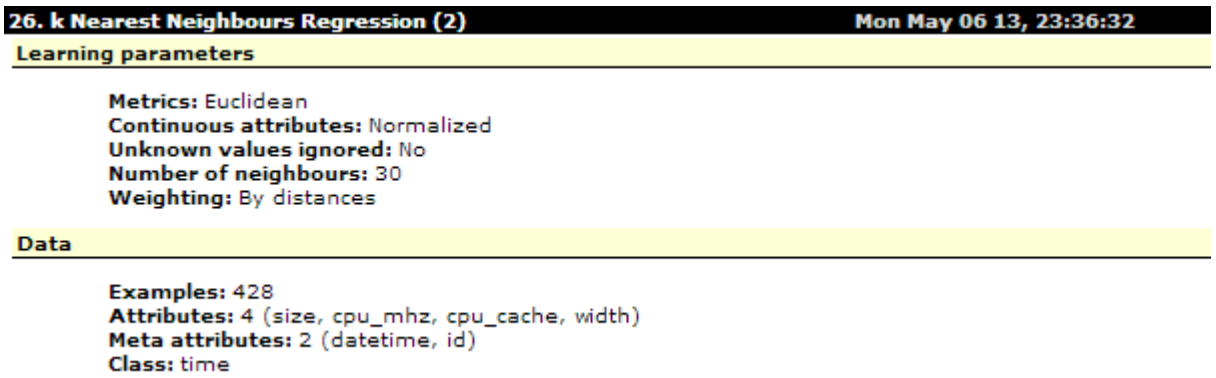


Рисунок 3.19 – Компонент `26. kNN (2)`. Предсказатель *k Nearest Neighbours* для модели с четырьмя признаками

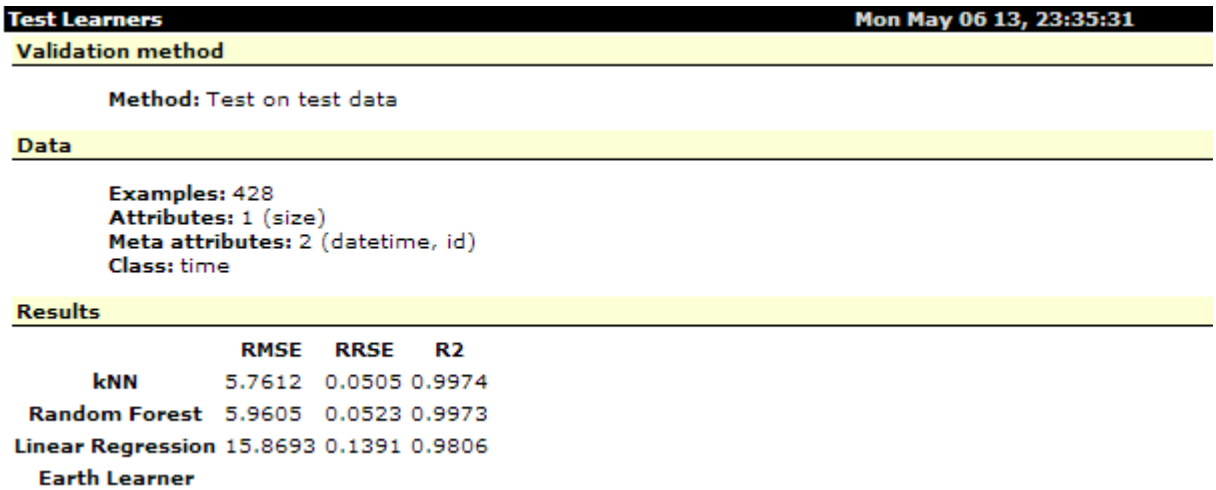
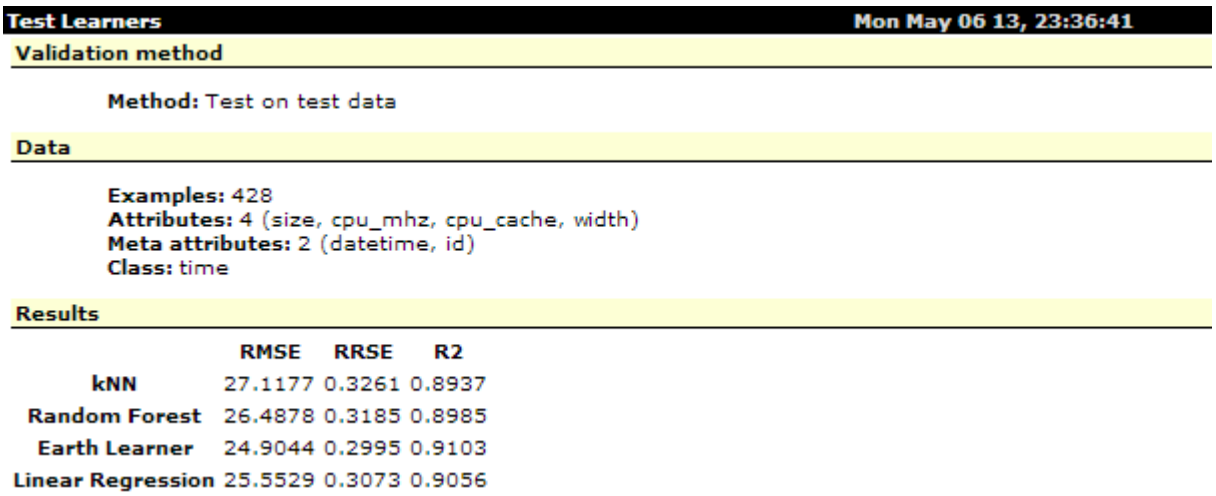


Рисунок 3.20 – Компонент `26. Test Learners (2)`. Сравнение предсказателей для модели с одним признаком



Test Learners — Mon May 06 13, 23:36:41

Validation method

Method: Test on test data

Data

Examples: 428
Attributes: 4 (size, cpu_mhz, cpu_cache, width)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	27.1177	0.3261	0.8937
Random Forest	26.4878	0.3185	0.8985
Earth Learner	24.9044	0.2995	0.9103
Linear Regression	25.5529	0.3073	0.9056

Рисунок 3.21 – Компонент `27. Test Learners (2)`. Сравнение предсказателей для модели с четырьмя признаками

с помощью компонентов `18. Save - Earth (1)`, `19. Save - RF (1)`, `20. Save - kNN (1)` и `31. Save - Earth (2)`, `32. Save - RF (2)`, `33. Save - kNN (2)` в формате, аналогичном формату исходного файла, загруженного компонентом `1. File`.

3.10 Выводы

В рамках конструкторской части дипломного проекта была построена комплексная система, состоящей из набора модулей, разработанных на языке программирования *Python*, и схемы статистической обработки данных в программе *Orange*.

Исследовательская часть

4 Оценка эффективности инструментария *Adaptor*

В рамках оценки эффективности разработанного программного комплекса проведём несколько серий экспериментов с его помощью, а затем осуществим анализ экспериментальных данных.

Далее в этом разделе мы последовательно рассмотрим следующие серии экспериментов и их анализ:

— серия экспериментов по проверке точности измерения времени;

— серия экспериментов по сравнению производительности программ, собранных компиляторами *GCC* и *LLVM* на настройках «по умолчанию»;

— группа серий экспериментов по моделированию и предсказанию производительности программы из набора *Polybench* на различном аппаратном обеспечении:

  — серия экспериментов по моделированию и предсказанию производительности при размере входных данных, который меняется по степенному закону, а обе размерности входных данных одинаковы;

  — серия экспериментов по моделированию и предсказанию производительности при размере входных данных, который меняется случайно при равномерном распределении случайной величины, а обе размерности входных данных одинаковы;

  — серия экспериментов по моделированию и предсказанию производительности при размере входных данных, который меняется случайно при равномерном распределении случайной величины, а размерности входных данных не одинаковы.

4.1 Проверка точности измерения времени

Для проверки точности измерения времени проведём следующий эксперимент.

Сгенерируем семейство программ, которые не делают ничего, кроме вызова функции стандартной библиотеки *C* `usleep`. Аргументами функции будет число 10^n, где n — число от одного до шести включительно. Таким образом, после запуска программа устанавливает таймер на заданное число микросекунд (от 1 мкс до 1000000 мкс = 1 с), останавливается и после его срабатывания завершает работу.

Далее на рис. 4.1 приведён график измеренного времени исполнения семейства таких программ и «реального» времени их выполнения — т.е. времени, указанного в аргументе функции, создающей таймер. График построен с помощью инструментария в автоматическом режиме.

Как мы видим на рис. 4.1, измеренное время асимптотически приближается к значению между 10^{-2} и 10^{-3} — около $0,005$.

Из этого можно сделать вывод, что существуют постоянные расходы на запуск программы из нашей системы и их можно вычесть из измеренного времени для повышения точности измерения. Для этого мы вычисляем время выполнения пустой программы (согласно описанной в подразделе 3.8.2 методике), а затем вычитаем его из каждого измеренного результата выполнения реальных программ.

На рис. 4.2 показан график измеренного времени в данном эксперименте с учётом накладных расходов.

Далее на рис. 4.3 приведён текстовый вывод из инструментария результата описанного эксперимента с учётом накладных расходов.

Таким образом, инструментарий позволяет производить достаточно точное измерение времени (ошибка в пределах 10%) для программ, исполняющихся 10 мс и более.

4.2 Сравнение компиляторов GCC и LLVM на тестовом наборе Polybench

На рисунке 4.4 сравнивается время исполнения программ, собранных компиляторами *GCC* и *LLVM* соответственно на уровне оптимизации `-O2`. Компилятор на этом уровне оптимизации в подавляющем большинстве случаев генерирует наиболее быстрый код (относительно уровней `-O0` и `-O1`).

Как мы видим из графика, на большинстве программ оба компилятора показывают примерно одинаковую производительность. Однако на шести программах компилятор *LLVM* серьёзно превосходит *GCC* – это программы *cholesky, trmm, trisolv, lu, jacobi-1d-imper, fdtd-2d.* Вероятно, *LLVM* использует лучший векторизатор кода, что оказывает большое влияние в

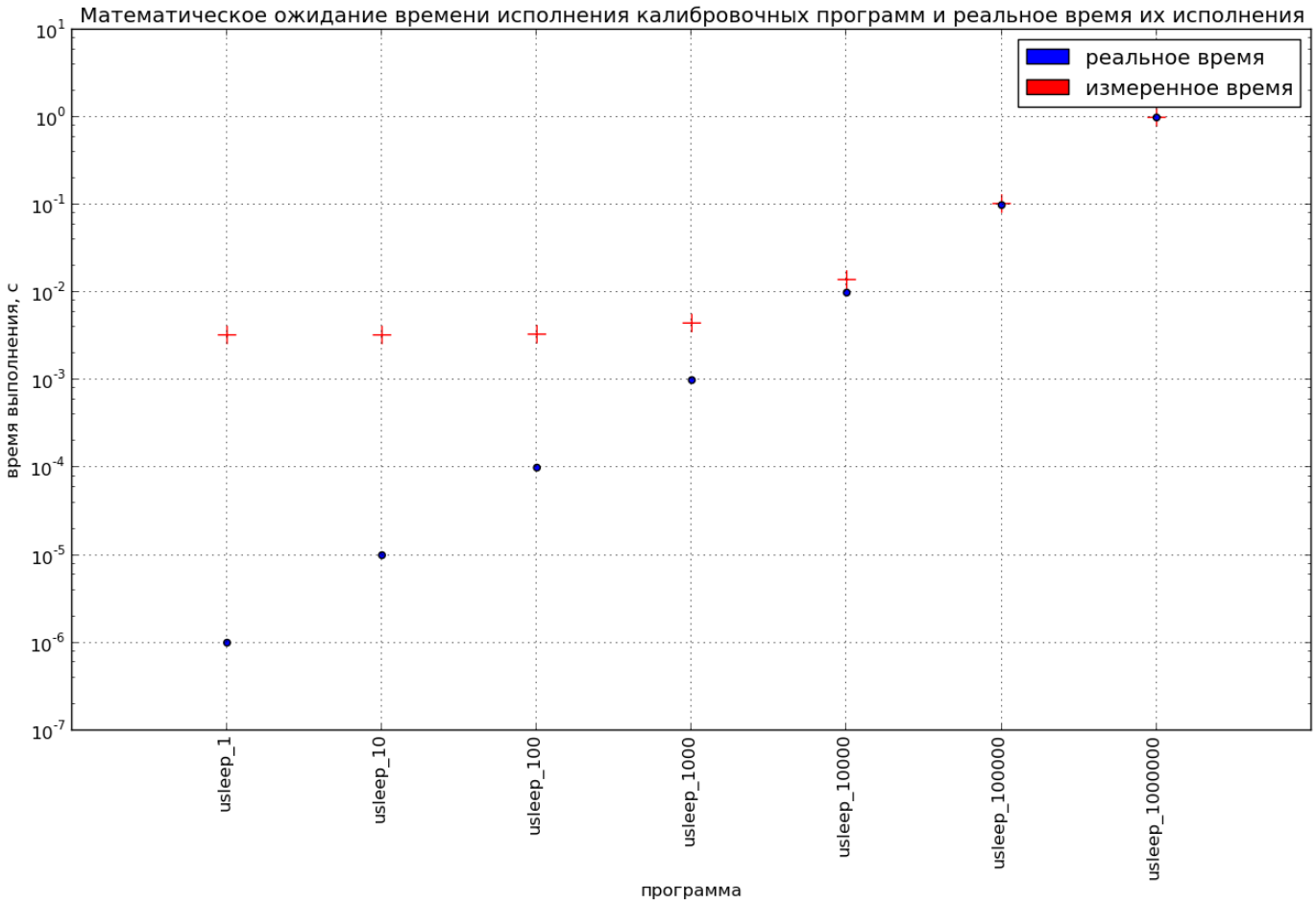


Рисунок 4.1 – График измеренного и реального времени исполнения семейства калибровочных программ

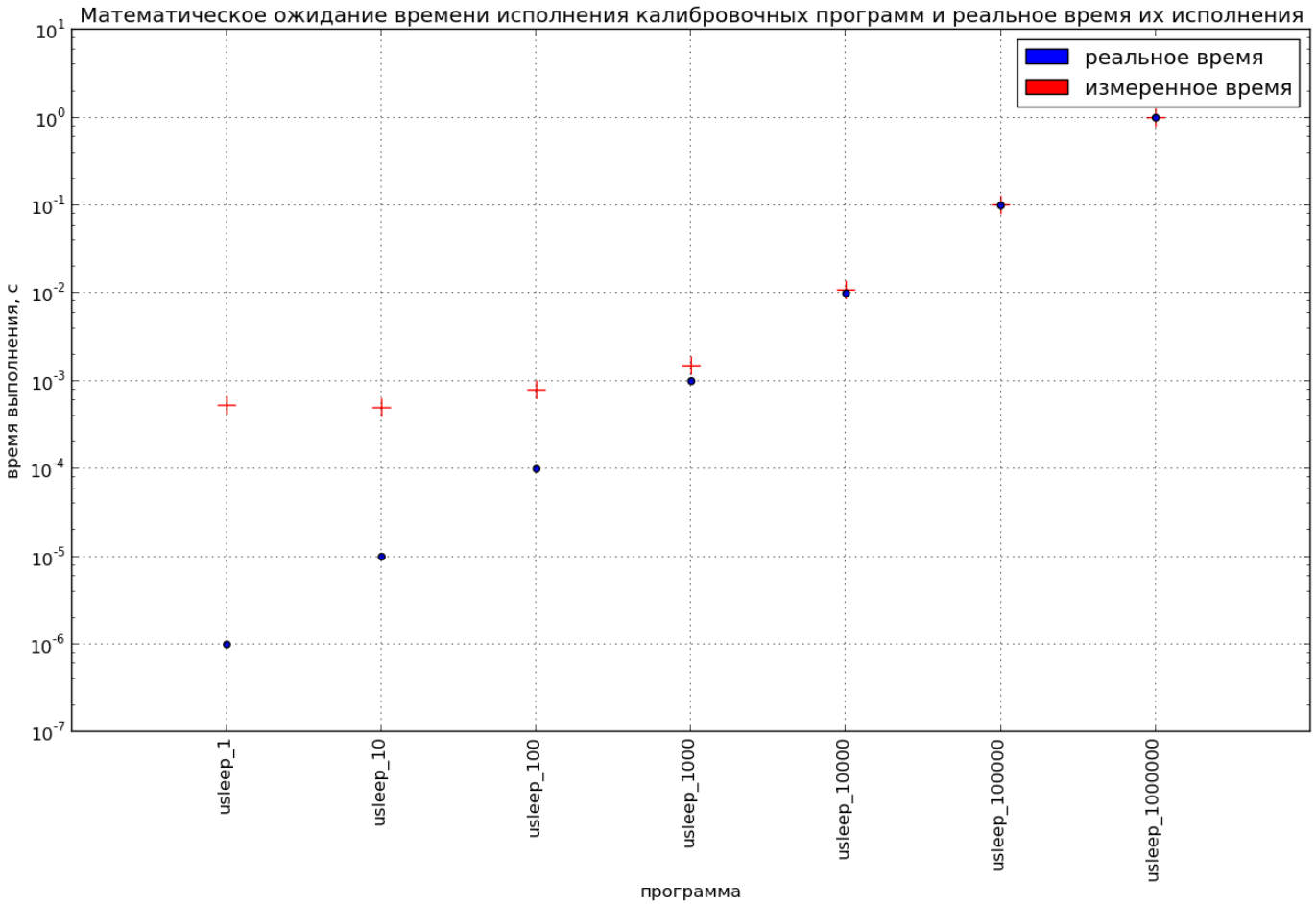


Рисунок 4.2 – График измеренного и реального времени исполнения семейства калибровочных программ с учётом накладных расходов

```
Experiment performed:
    Real time: 0.000001
    Measured time: 0.000531
    Relative error: 530.11

Experiment performed:
    Real time: 0.000010
    Measured time: 0.000498
    Relative error: 48.79

Experiment performed:
    Real time: 0.000100
    Measured time: 0.000795
    Relative error: 6.95

Experiment performed:
    Real time: 0.001000
    Measured time: 0.001499
    Relative error: 0.50

Experiment performed:
    Real time: 0.010000
    Measured time: 0.010893
    Relative error: 0.09

Experiment performed:
    Real time: 0.100000
    Measured time: 0.101603
    Relative error: 0.02

Experiment performed:
    Real time: 1.000000
    Measured time: 1.001015
    Relative error: 0.00
```

Рисунок 4.3 – Результат работы программы для описанного выше эксперимента

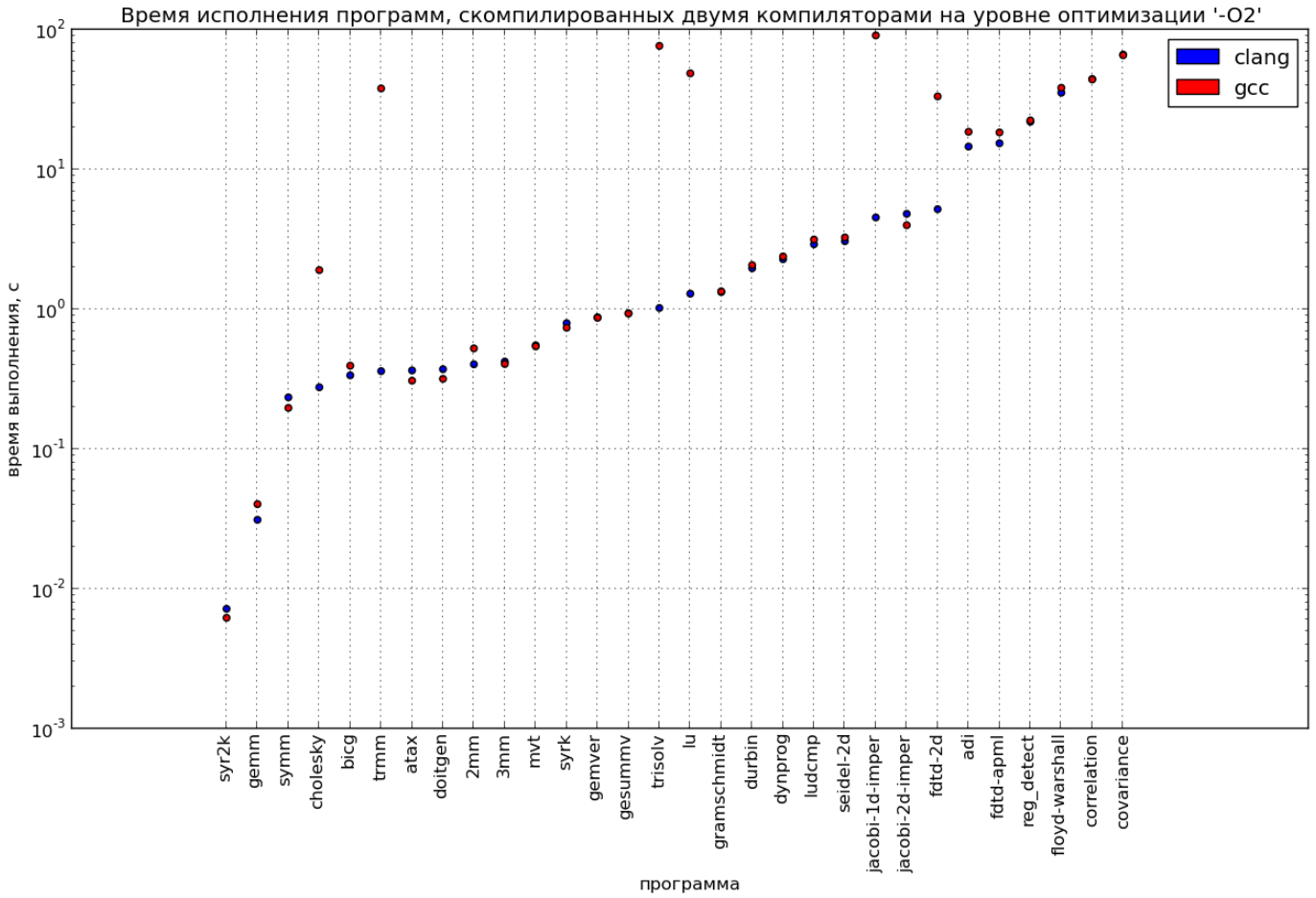


Рисунок 4.4 – График времени исполнения программ из набора *Polybench* для двух компиляторов

этих специфических случаях. В целом, указанные программы относятся к различным классам — это программы решения СЛАУ и СДУ. Выяснение конкретных причин этого превосходства выходит за рамки данной работы.

4.3 Моделирование и предсказание производительности программы из набора Polybench на различном аппаратном обеспечении

Одной из возможностей, предоставляемых инструментарием, является статистический анализ данных серии экспериментов с целью предсказания производительности программ на различном аппаратном обеспечении при различном размере и форме входных данных.

Для этого выберем программу из набора *Polybench*, которую будем анализировать. Критериями выбора являются факторы, приведённые ниже.

— Время исполнения программы при различных размерах и формах входных данных должно быть не слишком велико. Поскольку в рамках серии экспериментов программа должна быть исполнена статистически значимое число раз (по крайней мере 100), мы должны выбрать программу с учётом доступных ресурсов машинного времени и общих ресурсов времени, которое можно потратить на исследование.

— Время исполнения программы при различных размерах и формах входных данных должно быть не слишком мало. При уменьшении времени однократного исполнения программы погрешность измерения реального времени исполнения возрастает – до 50% при времени однократного исполнения 0,001 с (см. раздел 4.1). Поэтому мы должны выбрать программу, которая исполняется секунду или более, для достижения оптимальных результатов. При указанном времени исполнения точность измерения времени достигает как минимум 99% (раздел 4.1).

На основании указанных критериев и данных, полученных в рамках серии экспериментов, описанной в разделе 4.2, выбираем программу `symm` в качестве анализируемой.

Мы осуществим три серии экспериментов по моделированию и предсказанию производительности программы `symm` из набора *Polybench*. Они перечислены ниже.

1) Первая серия экспериментов характеризуется размерами входных данных M (число столбцов обрабатываемой матрицы) и N (число строк обрабатываемой матрицы), задаваемыми по степенному закону $M = N = 2^x, x = [1; 12]$. Таким образом, $M = N \in [2; 4096]$.

2) Вторая серия экспериментов характеризуется размерами входных данных M и N, задаваемыми случайно из диапазона [2;2048]: $M = N = rand([2; 2048])$, где $rand$ – функция случайного выбора целого числа из указанного диапазона по равномерному закону распределения. Таким образом, $M = N \in [2; 2048]$.

3) Третья серия экспериментов характеризуется размерами входных данных M и N, задаваемыми случайно и независимо из диапазона [2;2048]: $M = rand_1([2; 2048])$, $N = rand_2([2; 2048])$, где $rand_i, i \in [1; 2]$ – выборки случайного выбора целого числа из указанного диапазона по равномерному закону распределения (в данном случае выбираются два случайных числа). Таким образом, $M, N \in [2; 2048]$.

В каждой серии экспериментов производится обучение трёх предсказателей – *Earth, Random Forest и k-Nearest Neigbour* – и последующее предсказание производительности программы `symm` на различных аппаратных платформах и при различных размерах входных данных. Обучение производится на двух различных моделях – простейшей (включающей всего один признак из набора экспериментальных данных), и более сложной (включающей в себя до пяти признаков из набора экспериментальных данных). Обучение и предсказание на основе экспериментальных данных производится в системе статистической обработки данных с открытым исходным кодом *Orange* [32].

Хронологически, первой была выполнена серия экспериментов №3 как наиболее сложная, на которой предстояло выбрать надёжный алгоритм выбора признаков, который был бы способен работать и в более простых случаях (в сериях №1 и №2).

Система моделирования и предсказания организована в форме конвейера с ветвлениями, как описано в разделе 3.9.

Рассмотрим результаты выполнения серий экспериментов и моделирования производительности программы `symm` в каждом случае.

4.3.1 Серия экспериментов 1

Результаты сравнения предсказателей показаны на рисунках 4.5 и 4.6.

Как можно видеть на рисунках, в этой серии экспериментов, при использовании модели с одним признаком, все предсказатели показывают примерно одинаково высокие значения метрики $R2 \approx 0,99$, однако *kNN* и *RF* показывают значительно меньшие значения метрики RRSE, что означает более точное предсказание. Это отличный результат.

Из-за внутренней ошибки системы *Orange* проверить предсказатель *Earth* на модели с одним признаком не удалось.

Test Learners **Wed Apr 24 13, 00:25:22**

Validation method

Method: Test on test data

Data

Examples: 428
Attributes: 1 (size)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	5.7612	0.0505	0.9974
Random Forest	5.9605	0.0523	0.9973
Linear Regression	15.8693	0.1391	0.9806
Earth Learner			

Рисунок 4.5 – Сравнение предсказателей для модели с одним признаком. Серия 1

Test Learners **Wed Apr 24 13, 00:25:45**

Validation method

Method: Test on test data

Data

Examples: 428
Attributes: 4 (size, cpu_mhz, width, cpu_cache)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	27.1177	0.3261	0.8937
Random Forest	34.6648	0.4169	0.8262
Earth Learner	24.9044	0.2995	0.9103
Linear Regression	27.1238	0.3262	0.8936

Рисунок 4.6 – Сравнение предсказателей для модели с четырьмя признаками. Серия 1

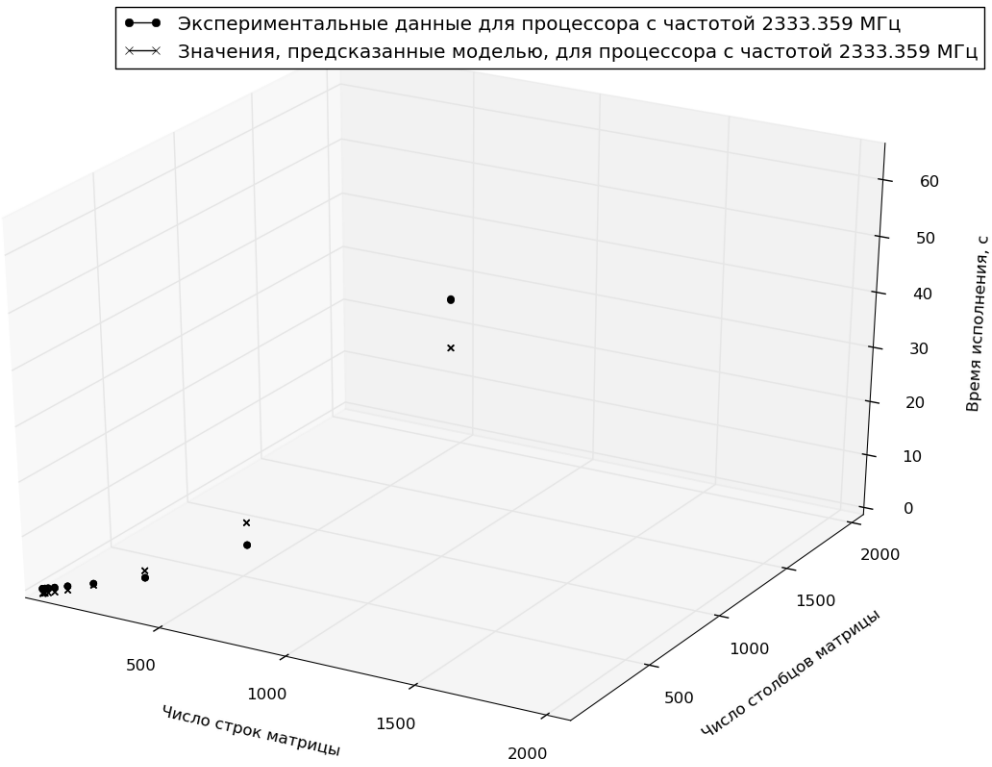


Рисунок 4.7 – Экспериментальные и предсказанные данные для процессора с частотой 2333 МГц. Серия 1. Предсказатель *Earth*

При использовании модели с четырьмя признаками предсказатель *Earth* превосходит все остальные по значениям метрик $RRSE$ и $R2$, и показывает $R2 \approx 0,91$, что следует признать хорошим результатом.

На рисунках 4.7, 4.8, 4.9 приведены графики лучшего предсказателя для модели с четырьмя признаками — *Earth Learner* — по данным для каждого из трёх процессоров.

Как мы видим на графиках, для всех трёх процессоров предсказания находятся достаточно близко к реально полученным данным. Это говорит о хорошей способности модели обобщать, то есть выводить наиболее простой закон получения неизвестного значения времени исполнения по известным значениям всех остальных свойств эксперимента. При этом не наблюдается явления переобучения, которое состоит в том, что модель показывает отличные результаты на учебной выборке и низкие результаты на проверочной.

4.3.2 Серия экспериментов 2

Результаты сравнения предсказателей показаны на рисунках 4.10 и 4.11.

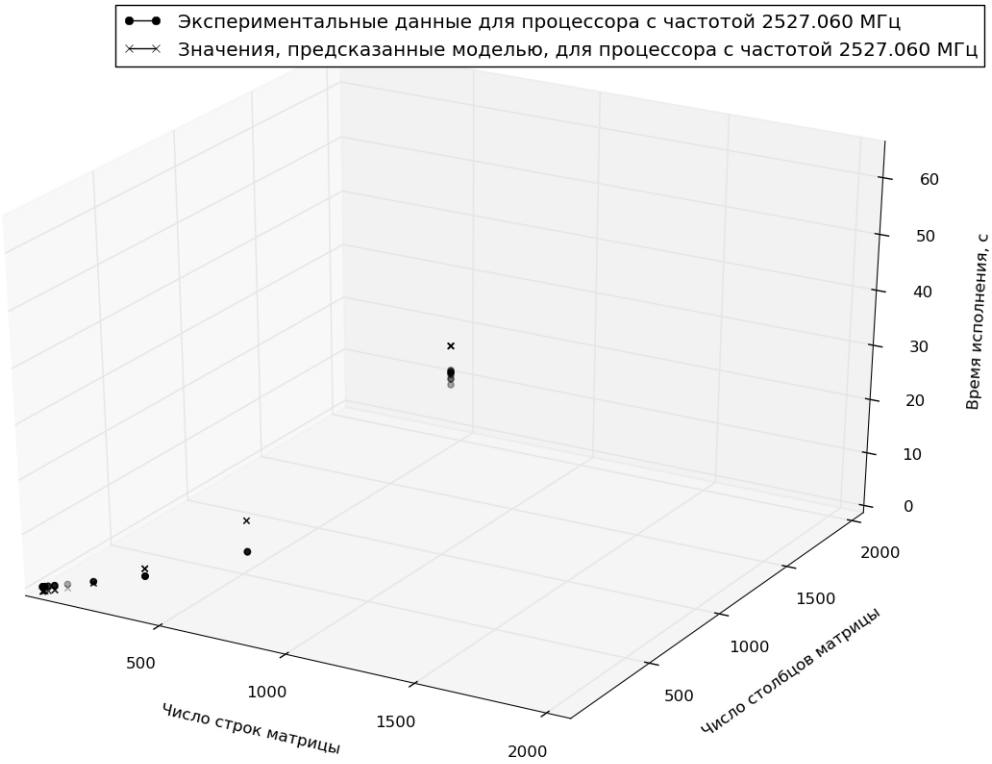


Рисунок 4.8 – Экспериментальные и предсказанные данные для процессора с частотой 2527 МГц. Серия 1. Предсказатель *Earth*

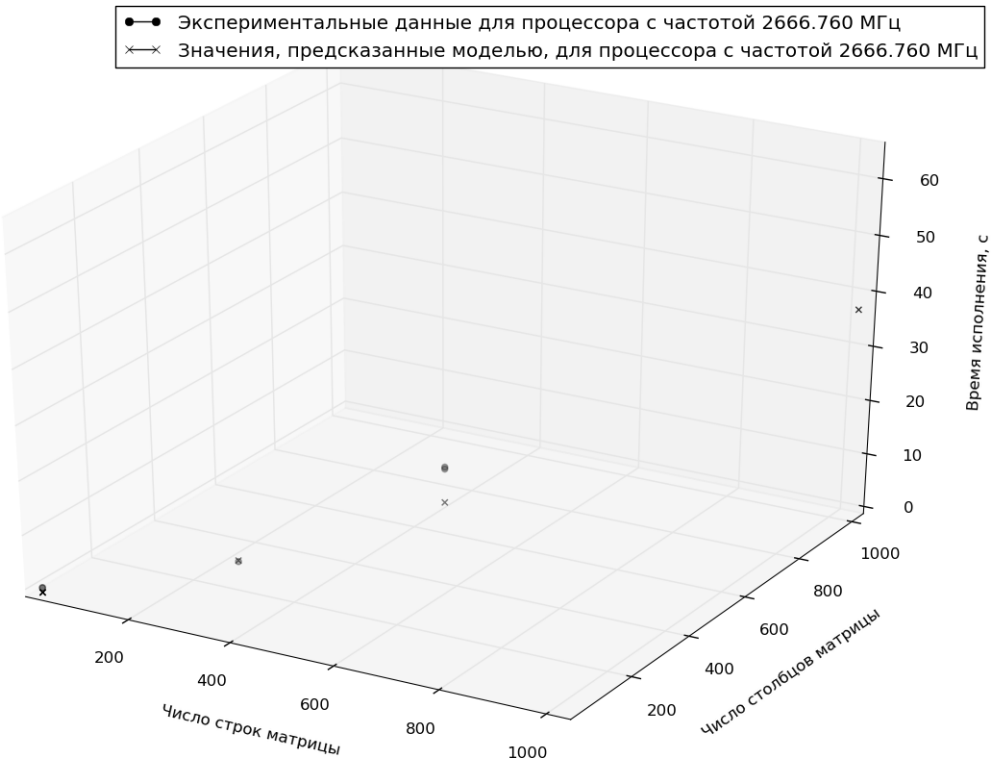
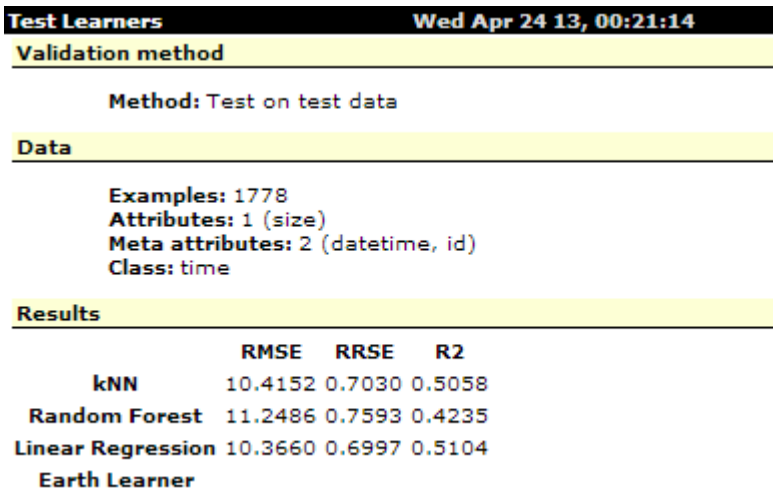


Рисунок 4.9 – Экспериментальные и предсказанные данные для процессора с частотой 2667 МГц. Серия 1. Предсказатель *Earth*



Test Learners — **Wed Apr 24 13, 00:21:14**

Validation method

Method: Test on test data

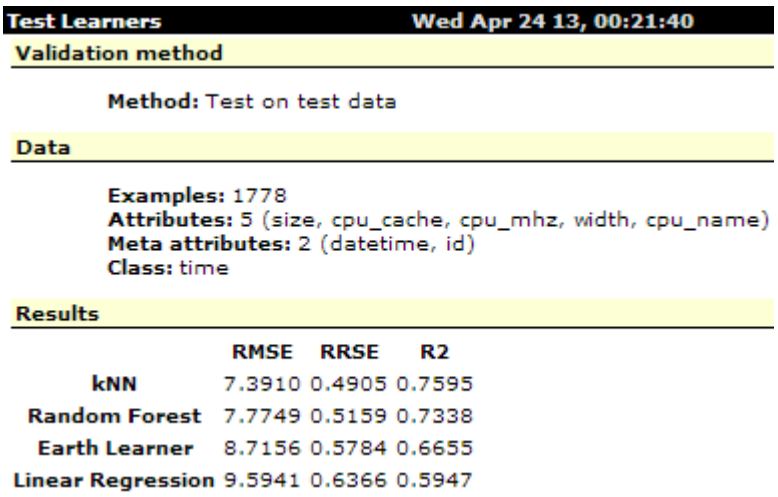
Data

Examples: 1778
Attributes: 1 (size)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	10.4152	0.7030	0.5058
Random Forest	11.2486	0.7593	0.4235
Linear Regression	10.3660	0.6997	0.5104
Earth Learner			

Рисунок 4.10 – Сравнение предсказателей для модели с одним признаком. Серия 2



Test Learners — **Wed Apr 24 13, 00:21:40**

Validation method

Method: Test on test data

Data

Examples: 1778
Attributes: 5 (size, cpu_cache, cpu_mhz, width, cpu_name)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	7.3910	0.4905	0.7595
Random Forest	7.7749	0.5159	0.7338
Earth Learner	8.7156	0.5784	0.6655
Linear Regression	9.5941	0.6366	0.5947

Рисунок 4.11 – Сравнение предсказателей для модели с четырьмя признаками. Серия 2

В этой серии экспериментов результаты предсказания ожидаемо хуже. Значения метрики $R2$ достигают только примерно $0,51$ для модели с одним признаком и примерно $0,76$ для модели с пятью признаками.

Следует отметить, что в случае модели с одним признаком ни один из предсказателей не показал результатов лучше, чем образцовый предсказатель — линейная регрессия. Это говорит о неспособности модели

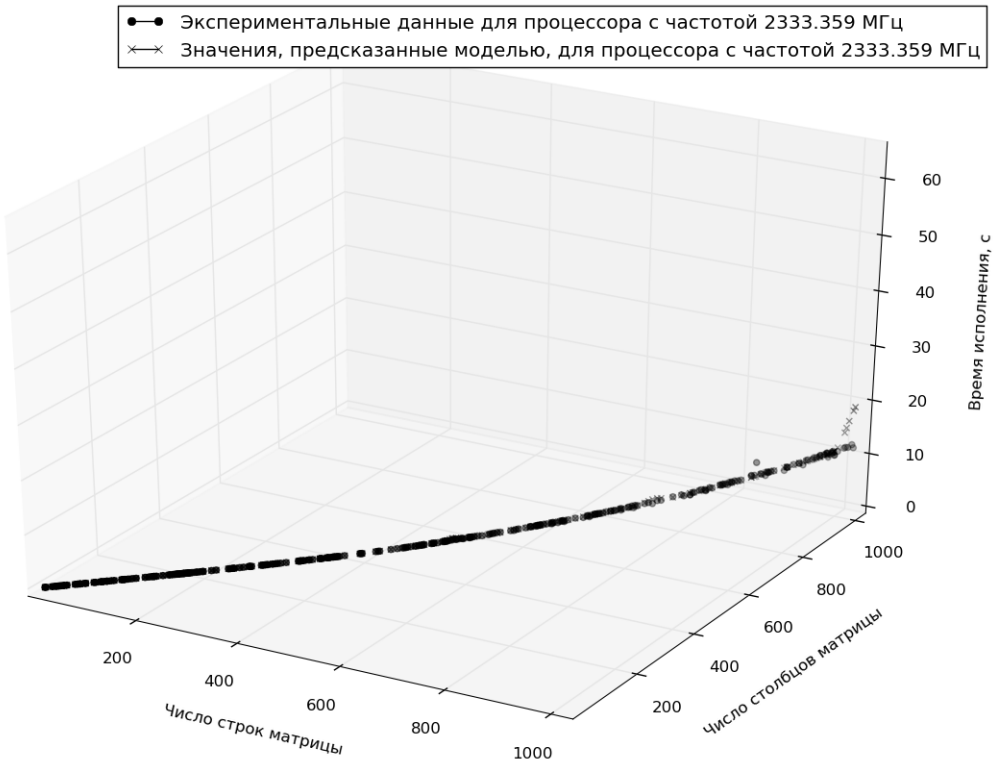


Рисунок 4.12 – Экспериментальные и предсказанные данные для процессора с частотой 2333 МГц. Серия 2. Предсказатель *kNN*

предоставить данные для адекватной работы предсказателей, более сложных, чем линейный. Эта модель не может качественно представлять данные серии экспериментов №2.

При использовании модели с пятью признаками наилучший результат показал предсказатель *kNN*: $RRSE = 0,49; R2 = 0,76$. Однако, этот результат значительно ниже, чем полученный в серии №1. Давайте посмотрим на графики предсказаний (рисунки 4.12, 4.13, 4.14) для определения источника проблем.

Первые два графика показывают довольно адекватные в целом предсказания: только на правом крае графика, при больших значениях размера матрицы, наблюдается небольшое расхождение. Однако для последнего процессора, Intel Xeon 2,66 ГГц, модель начинает предсказывать неадекватные значения времени исполнения уже начиная с размера матрицы 400×400. Можно заметить, что часть экспериментальных данных для матриц больших размеров располагается на показательной кривой, а часть — на практически линейном участке, причём линейная часть находится в диапазоне очень маленьких времён исполнения. Это является артефактом проведения эксперимента на виртуальном сервере, который может не иметь эксклюзивного доступа к аппаратному обеспечению физического сервера, на котором он запущен. Судя по разбросу данных, периодически

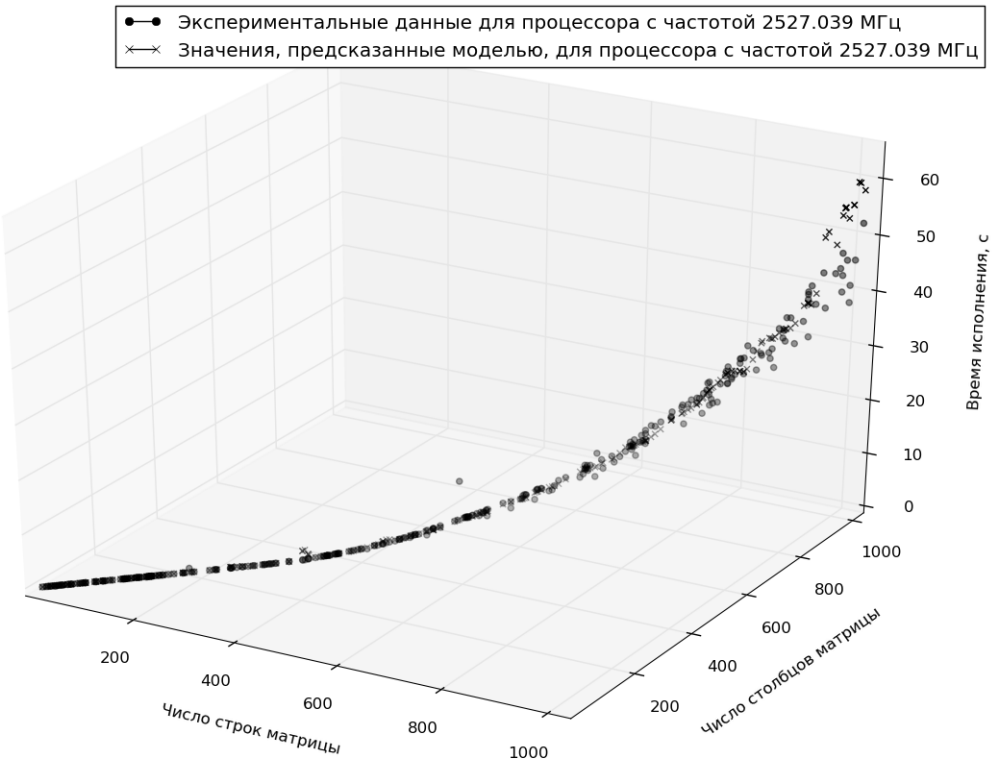


Рисунок 4.13 – Экспериментальные и предсказанные данные для процессора с частотой 2527 МГц. Серия 2. Предсказатель *kNN*

происходило переключение сервера в режим высокой производительности, при котором он получал полный доступ к аппаратным ресурсам, в результате чего исследуемая программа исполнялась за значительно меньшее время. Бóльшую же часть времени сервер мог использовать только часть ресурсов, поэтому время исполнения в основном находится на верхней, показательной, кривой.

Этим объясняется и неадекватность предсказаний для последнего процессора — взглянув на графики, можно увидеть, что предсказатель попытался объяснить и наличие точек на показательной кривой, и на более низком линейном участке, в результате чего кривая предсказаний оказалась посередине и представляет собой сильно зашумленную, неадекватную кривую, состоящую из нескольких линейных участков.

Причиной неадекватности модели в данном случае стало отсутствие в наборе данных каких-то свойств, указывающих на то, что машина является виртуальным сервером и ей могут быть не доступны все ресурсы реального сервера. Проблемой здесь оказалось то, что модулю опроса аппаратного обеспечения сообщаются именно данные о реальном сервере, а не о текущих доступных ресурсах, и нет простого способа распознать использование виртуализации.

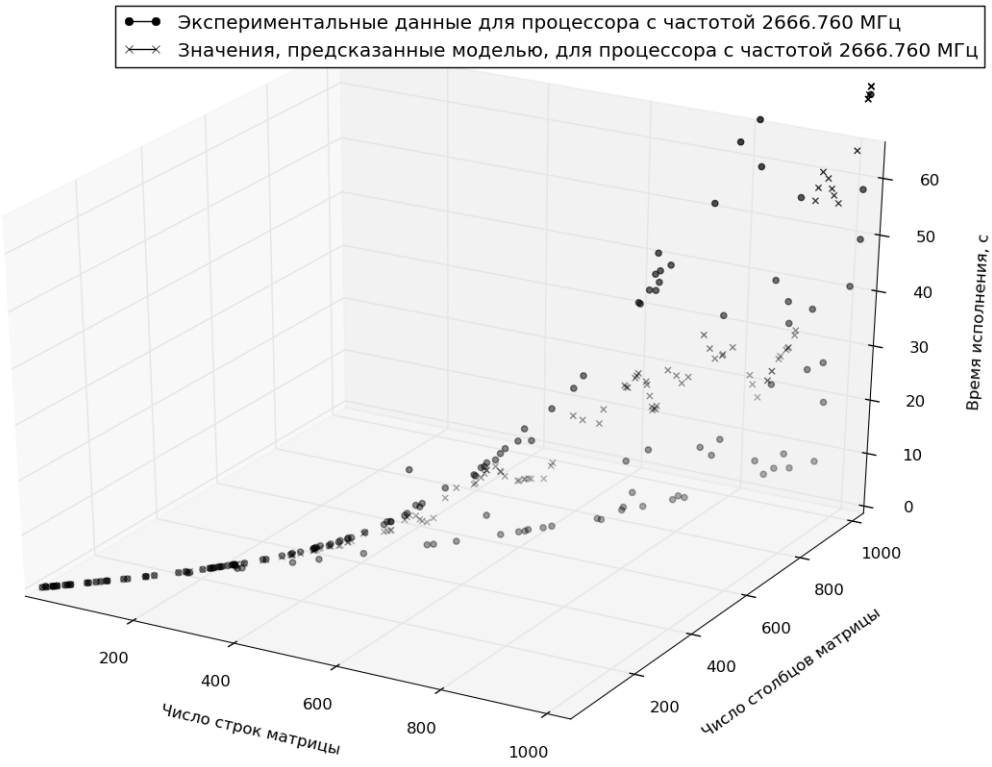


Рисунок 4.14 – Экспериментальные и предсказанные данные для процессора с частотой 2667 МГц. Серия 2. Предсказатель *kNN*

4.3.3 Серия экспериментов 3

Результаты сравнения предсказателей показаны на рисунках 4.15 и 4.16.

В этой серии экспериментов предсказания оказались наименее точными.

Модель с одним признаком оказалась неспособной давать адекватные предсказания, поскольку все предсказатели показали результаты хуже образцового. Линейная регрессия же показала результат с значением метрики $R2 = 0,2336$, что является неудовлетворительным результатом. В реальных условиях модель с одним признаком оказалась непригодной для использования.

Для модели с пятью признаками наилучшие результаты снова показал предсказатель *kNN*, таким образом показав своё превосходство в случаях, когда эксперименты проводятся в более непредсказуемых условиях. Его значения метрик $RRSE = 0,62; R2 = 0,61$. Это удовлетворительный результат, учитывая, что фильтрация данных с целью удаления шума не производилась.

Давайте посмотрим на графики предсказаний (рисунки 4.17, 4.18, 4.19) для определения сложностей, с которыми столкнулся предсказатель.

Поскольку графики являются трёхмерными, выявить вид предсказывающей кривой гораздо сложнее. Можно сказать, что предсказатель страдает от переобучения, поскольку большая часть значений

Test Learners **Wed Apr 24 13, 00:14:23**

Validation method

Method: Test on test data

Data

Examples: 2100
Attributes: 1 (size)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	5.5020	0.9023	0.1858
Random Forest	5.9208	0.9710	0.0571
Linear Regression	5.3382	0.8755	0.2336
Earth Learner			

Рисунок 4.15 – Сравнение предсказателей для модели с одним признаком. Серия 3

Test Learners **Wed Apr 24 13, 00:15:02**

Validation method

Method: Test on test data

Data

Examples: 2100
Attributes: 5 (size, width, height, cpu_mhz, cpu_cache)
Meta attributes: 2 (datetime, id)
Class: time

Results

	RMSE	RRSE	R2
kNN	3.7951	0.6212	0.6141
Random Forest	3.9223	0.6420	0.5878
Earth Learner	4.1847	0.6850	0.5308
Linear Regression	5.0445	0.8257	0.3181

Рисунок 4.16 – Сравнение предсказателей для модели с четырьмя признаками. Серия 3

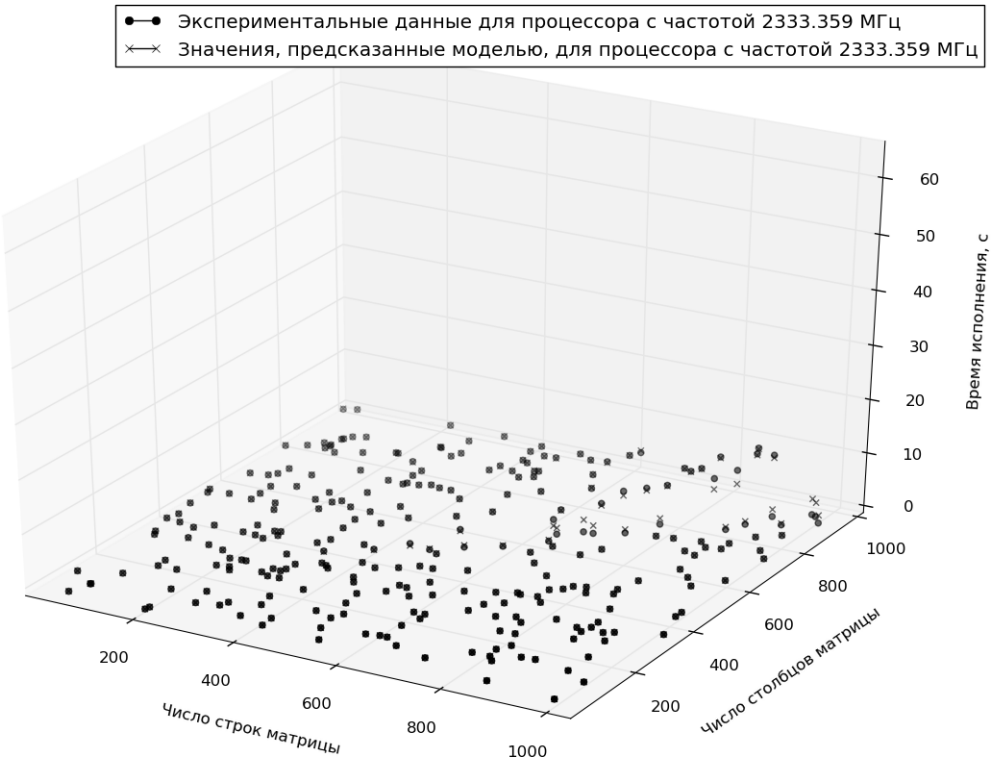


Рисунок 4.17 – Экспериментальные и предсказанные данные для процессора с частотой 2333 МГц. Серия 3. Предсказатель *kNN*

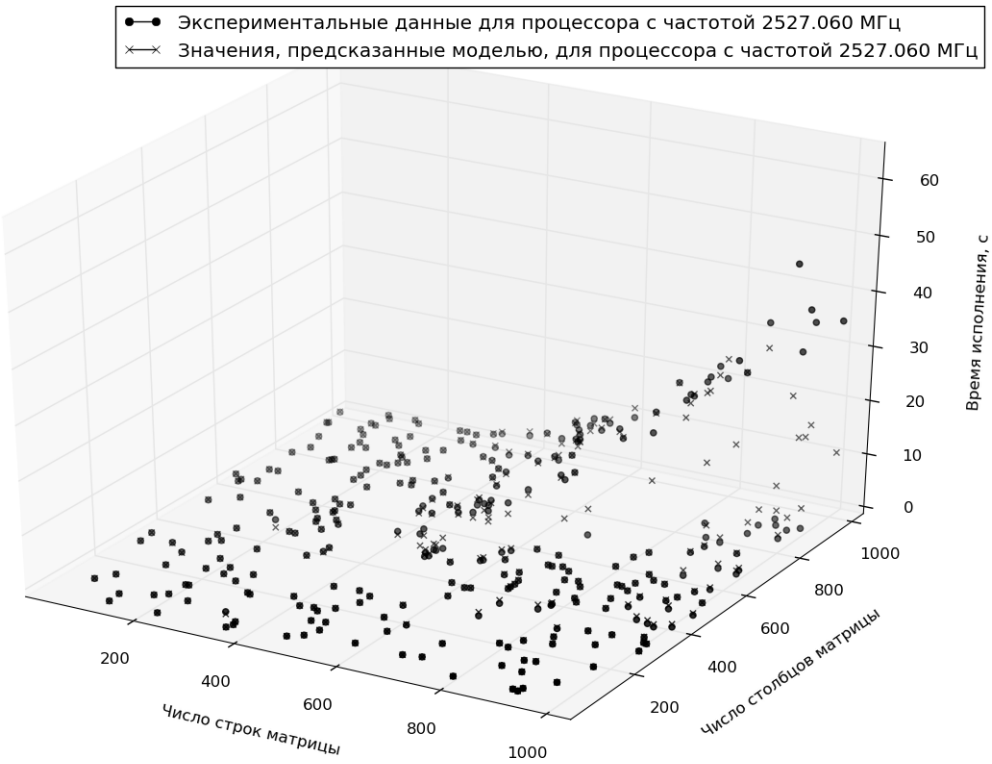


Рисунок 4.18 – Экспериментальные и предсказанные данные для процессора с частотой 2527 МГц. Серия 3. Предсказатель *kNN*

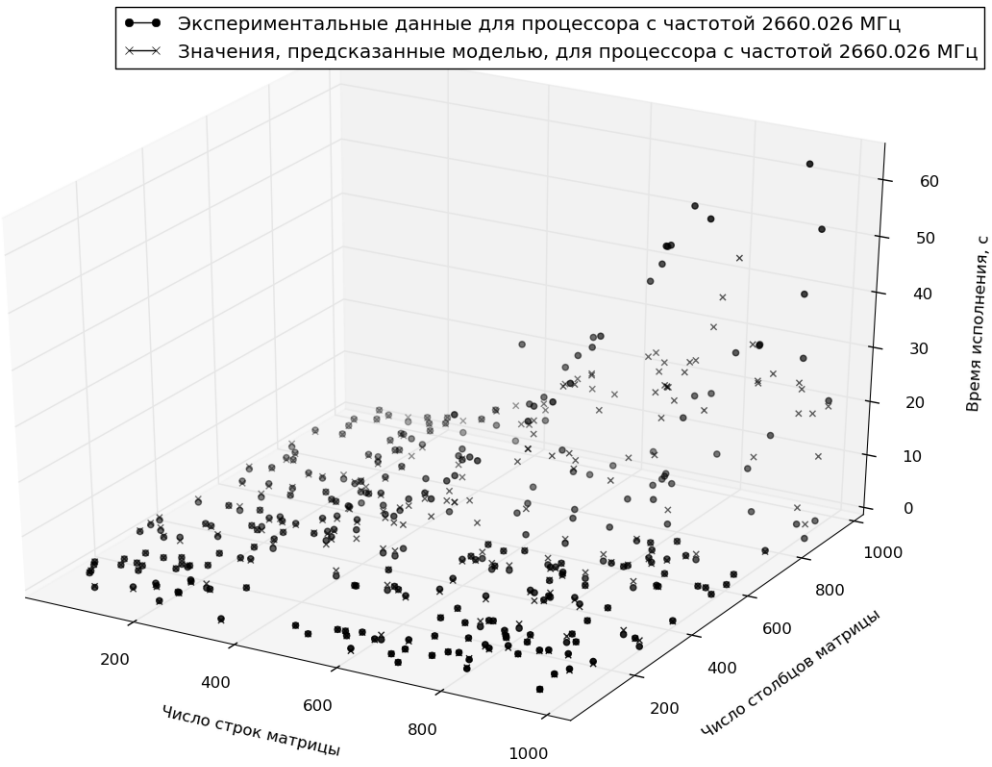


Рисунок 4.19 – Экспериментальные и предсказанные данные для процессора с частотой 2667 МГц. Серия 3. Предсказатель *kNN*

времени оказалась сильно зашумлёнными, минимально возможными к измерению значениями. Из-за этого их колебания очень велики и в ряде случаев из-за артефактов измерения значения времени аномально близки к нулю или даже отрицательны. Это очевидно снижает предсказывающие способности модели, поскольку она производит попытки объяснить наличие зашумленных данных.

Вторым фактором является само по себе преобладание предельно коротких запусков программы. Это ведёт к тому, что вероятность возникновения маленьких времён исполнения в результате предсказания возрастает, поскольку в учебном наборе преобладают короткие запуски.

Обе эти проблемы частично решаются фильтрацией шумных результатов, однако определение критериев, по которым измеренное значение можно считать шумом, является не тривиальной задачей и выходит за рамки этой работы.

4.4 Выводы

В рамках оценки эффективности разработанного инструментария было произведено несколько исследований, в ходе которых была доказана возможность эффективно решать задачу статистического анализа производительности программ на различном аппаратном обеспечении.

Организационно-экономическая часть

5 Введение

Экономическая часть посвящена разработке комплекса мероприятий организационно–экономического и финансового планов, которые необходимо выполнить для создания программного продукта, позволяющего проводить анализ и моделирование производительности программ. Расчёты производятся согласно методическим указаниям [46].

Система разработана на языке программирования Python, и включает в себя группу проектов анализа данных в системе проведения статистических исследований Orange. Программный продукт не может быть интегрирован в другие программные комплексы и предназначен для внутреннего использования. Заказчиком является структурное подразделение МГТУ им. Н. Э. Баумана. Данный проект не преследует цель получения прибыли.

6 Основные этапы проекта разработки нового изделия

Основные этапы проекта разработки нового изделия представлены в таблице 6.1.

Таблица 6.1 – Основные этапы разработки проекта

№	Условное обозначение	Описание
1	Техническое задание (ТЗ)	Предпроектное обследование. Постановка задачи по созданию программного продукта
2	Эскизный проект (ЭП)	Проработка использования основных технологий и инструментов, необходимых для выполнения программного продукта
3	Технорабочий проект (ТП)	Разработка модели анализа: структуры данных и реализации алгоритмов анализа. Разработка программного решения
4	Документация и внедрение (В)	Подготовка и передача программного продукта и программной документации

7 Расчёт трудоёмкости разработки программного продукта

Трудоёмкость разработки программного продукта рассчитывается по формуле

$$\tau_{\text{пп}} = \tau_{\text{тз}} + \tau_{\text{эп}} + \tau_{\text{тп}} + \tau_{\text{рп}} + \tau_{\text{в}},$$

где $\tau_{\text{тз}}$ – трудоёмкость разработки технического задания на создание ПП; $\tau_{\text{эп}}$ – трудоёмкость разработки эскизного проекта ПП; $\tau_{\text{рп}}$ – трудоёмкость разработки рабочего проекта ПП; $\tau_{\text{в}}$ – трудоёмкость внедрения разработанного ПП.

Трудоёмкость разработки технического задания можно рассчитать по формуле

$$\tau_{\text{тз}} = T_{\text{зрз}} + T_{\text{зрп}}$$

где $T_{\text{зрз}}$ – затраты времени разработчика постановки задачи на разработку ТЗ, человеко-дни, $T_{\text{зрп}}$ – затраты времени разработчика программного обеспечения на разработку ТЗ, человеко-дни.

Значения затрат времени $T_{\text{зрз}}$ и $T_{\text{зрп}}$ определяют по формулам:

$$T_{\text{зрз}} = t_{\text{з}} K_{\text{зрз}}$$
$$T_{\text{зрп}} = t_{\text{з}} K_{\text{зрп}},$$

где $t_з$ – норма времени на разработку ТЗ на программный продукт в зависимости от функционального назначения и степени новизны разрабатываемого ПП, человеко-дни;

$K_{зрз}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задач на стадии ТЗ (в случае совместной с разработчиком ПО разработки $K_{зрз} = 0,65$, в случае самостоятельной разработки ТЗ $K_{зрз} = 1$); $K_{зрп}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком ПП на стадии ТЗ (в случае совместной с постановки задач разработки $K_{зрп} = 0,35$, в случае самостоятельной разработки ТЗ $K_{зрп} = 1$).

В данном случае следует принять следующие значения коэффициентов

$$t_з = 3\text{чел.} - \text{дней}. \tag{4.1}$$

Тогда трудоёмкость разработки технического задания

$$\tau_{тз} = T_{зрз} + T_{трп} = 3 \cdot 1 = 3\text{чел.} - \text{дней}. \tag{4.2}$$

Трудоёмкость разработки эскизного проекта ПП рассчитывают по формуле

$$\tau_{эп} = T_{эрз} + T_{эрп}, \tag{4.3}$$

где $T_{эрз}$ – затраты времени разработчика постановки задачи на разработку эскизного проекта, человеко-дни, $T_{эрп}$ – затраты времени разработчика ПП на разработку эскизного проекта, человеко-дни.

Значения величин $T_{эрз}$ и $T_{эрп}$ рассчтываются по формулам:

$$T_{эрп} = t_э K_{эрп} T_{эрз} = t_э K_{эрз},$$

где t_{erev} – норма времени на разработку эскизного проекта на ПП в зависимости от функционального назначения и степени новизны разрабатываемого ПП, человеко-дни;

$K_{эрп}$ – коэффициент, учитывающий удельный вес трудоёмкости работ, выполняемых разработчиком постановки задач на стадии эскизного проекта;

$K_{эрз}$ – коэффициент, учитывающий удельный вес трудоёмкости работ, выполняемых разработчиком ПП на стадии эскизного проекта.

В данном случае следует принять следующие значения коэффициентов

$$t_э = 18\text{чел.} - \text{дней}, K_{эрп} = 0, K_{эрз} = 1.$$

Тогда трудоемкость разработки эскизного проекта ПП равна

$$\tau_{эп} = T_{эрз} + T_{эрп} = 18\text{чел.} - \text{дней}.$$

Трудоемкость разработки технического проекта зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации и определяется как сумма времени, затраченного разработчиком постановки задач и разработчиком программного обеспечения:

$$\tau_{\text{тп}} = (t_{\text{трз}} + t_{\text{трп}}) \cdot K_{\text{в}} K_{\text{р}},$$

где $t_{\text{трз}}$ и $t_{\text{трп}}$ – норма времени, затрачиваемого на разработку ТП разработчиком постановки задач и разработчиком ПП соответственно, человеко-дни ($t_{\text{трп}} = 20$чел. $-$ дней);

$K_{\text{в}}$ – коэффициент учёта вида используемой информации;

$K_{\text{р}}$ – коэффициент учёта режима обработки информации ($K_{\text{р}} = 1,26$).

Значение коэффициента $K_{\text{в}}$ определяется по формуле

$$K_{\text{в}} = \frac{K_{\text{п}} n_{\text{п}} + K_{\text{нс}} n_{\text{нс}} + K_{\text{б}} n_{\text{б}}}{n_{\text{п}} + n_{\text{нс}} + n_{\text{б}}}$$

где $K_{\text{п}}$, $K_{\text{нс}}$, $K_{\text{б}}$, – значения коэффициентов учета вида используемой информации для переменной, нормативно–справочной информации и баз данных соответственно; $n_{\text{п}}$, $n_{\text{нс}}$, $n_{\text{б}}$ – количество наборов данных переменной, нормативно–справочной информации и баз данных соответственно.

$$K_{\text{в}} = (1,0 \cdot 4 + 0,72 \cdot 2 + 2,08 \cdot 4)/10 = 1,376$$

Тогда трудоемкость разработки технического проекта

$$\tau_{\text{тп}} = (t_{\text{трз}} + t_{\text{трп}}) \cdot K_{\text{в}} K_{\text{р}} = 20 \cdot 1,26 \cdot 1,376 = 35\text{чел.} - \text{дней},$$

Трудоемкость разработки рабочего проекта зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации, сложности алгоритма, сложности контроля информации, степени использования готовых программных модулей, уровня алгоритмического языка программирования и определяется по формуле:

$$\tau_{\text{рп}} = (t_{\text{ррз}} + t_{\text{ррп}}) \cdot K_{\text{к}} K_{\text{р}} K_{\text{я}} K_{\text{г}} K_{\text{иа}},$$

где $K_{\text{к}}$ – коэффициент учета сложности контроля информации ($K_{\text{к}} = 1,00$); $K_{\text{р}}$ – коэффициент учета режима обработки информации ($K_{\text{р}} = 1,32$); $K_{\text{я}}$ – коэффициент учета уровня используемого алгоритмического языка программирования ($K_{\text{я}} = 1,00$); $K_{\text{г}}$ – коэффициент учета степени использования готовых программных модулей – 25% – 40% ($K_{\text{г}} = 0,70$); $K_{\text{иа}}$ – коэффициент учета вида используемой информации и сложности алгоритма ПП. Значение коэффициента $K_{\text{иа}}$ определяют по формуле

$$K_{\text{иа}} = \frac{K'_{\text{п}} n_{\text{п}} + K'_{\text{нс}} n_{\text{нс}} + K'_{\text{б}} n_{\text{б}}}{n_{\text{п}} + n_{\text{нс}} + n_{\text{б}}}$$

где $K'_{\text{п}}$, $K'_{\text{нс}}$, $K'_{\text{б}}$ – значения коэффициентов учета сложности алгоритма ПП и вида используемой информации для переменной, нормативно–справочной информации и баз данных соответственно, $K_{\text{иа}} = 0,98$; $t_{\text{ррз}}$, $t_{\text{ррп}}$ – норма времени, затраченного на разработку РП на алгоритмическом языке высокого уровня разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел.–дни. ($t_{\text{ррп}} = 51$ чел.-дней), так как используется 2 вида входной и 2 вида выходной информации.

Соответственно трудоемкость технорабочего проекта можно рассчитать следующим образом:

$$\tau_{\text{трп}} = 0,85 \cdot \tau_{\text{тп}} + \tau_{\text{рп}} = 0,85 \cdot 35 + 50 = 80\text{чел.} - \text{дней.}$$

Трудоемкость выполнения стадии внедрения рассчитывается по формуле:

$$t_{\text{в}} = (t_{\text{врз}} + t_{\text{врп}}) \cdot K_{\text{к}} K_{\text{р}} K_{\text{з}},$$

где $t_{\text{врз}}$, $t_{\text{врп}}$ – норма времени, затрачиваемого разработчиком постановки задач и разработчиком программного обеспечения соответственно на выполнение процедур внедрения ПП, человеко–дни.

$$t_{\text{врп}} = 10\text{чел.} - \text{дней},$$
$$K_{\text{к}} = 1,08, K_{\text{р}} = 1,32, K_{\text{з}} = 0,7.$$

Подставив значения коэффициентов, получим

$$t_{\text{в}} = (t_{\text{врз}} + t_{\text{врп}}) \cdot K_{\text{к}} K_{\text{р}} K_{\text{з}} = 10 \cdot 1,08 \cdot 1,32 \cdot 0,7 \approx 10\text{чел.} - \text{дней.}$$

Таким образом, трудоемкость разработки программной продукции составляет

$$\tau_{\text{пп}} = 3 + 18 + 80 + 10 = 111\text{чел.} - \text{дней.}$$

Планирование и контроль хода выполнения разработки проводят по календарному графику выполнения работ. Можно использовать ленточный график (график Гантта), который представляет собой графическое отображение выполненной работы и времени, затраченного на эту работу, т. е. продолжительность выполнения данной работы. Продолжительность выполнения всех работ по этапам разработки ПП определяют из выражения

$$T_i = \frac{(t_i + Q)}{n_i \cdot f},$$

где t_i – трудоемкость i-й работы, чел.–дни; Q – трудоемкость дополнительных работ, выполняемых исполнителем, чел-дни; n_i – количество исполнителей, выполняющих i-ю работу, чел; f – коэффициент пересчета из рб.–дн. в кл.–дн., равный, для 2013 года, 248/365=0,679.

Таблица 7.1 – Расчёт основных этапов разработки проекта

этап	t_i, чел.-дн.	кол-во исполнителей	рб.-дн.	кл-дн.
ТЗ	3	1	3	4
ЭП	18	1	18	26
ТРП	80	1	80	112
В	10	1	10	15
Всего			111	157

8 Календарный план-график проекта

Для иллюстрации последовательности проводимых работ проекта применяют ленточный график, календарный план–график, диаграмму Гантта. На диаграмме Гантта на оси X показывают календарные дни (по рабочим неделям) от начала проекта до его завершения. По оси Y – выполняемые этапы работ.

Для наглядности приведём график Гантта отражающий взаимодействие шагов проекта во времени. График показывает последовательность действий в нужном порядке и те из них, которые могут выполняться одновременно.

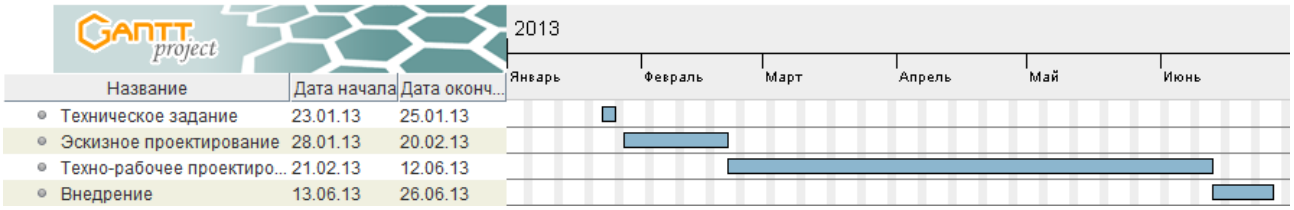


Рисунок 8.1 – Диаграмма Гантта.

9 Затраты на разработку программного продукта

Затраты на разработку программной продукции могут быть представлены в виде сметы затрат, включающей в себя следующие статьи:

— материальные затраты;

— амортизационные отчисления;

— расчёт заработной платы;

— отчисления в фонды;

— прочие затраты.

9.1 Расчёт материальных затрат

Суммарные затраты на материалы состоят из стоимости материалов и транспортно-заготовительных расходов, т.е.

$$C_{\text{м}} = K_{\text{тп}} \sum_i \text{Ц}_i V_i,$$

где $K_{\text{тп}}$ – коэффициент транспортно-заготовительных расходов ($K_{\text{тп}} = 0,19$);
Ц_i – цена единицы i-го материала, руб.;
V_i – приобретённое количество i-го материала, шт.

Таблица 9.1 – Расходы на материалы

Наименование материала	Единица измерения	Количество	Цена единицы, руб.	Сумма, руб
Диск DVD+R	шт.	2	30	60
Устройство флэш-памяти	шт.	1	800	800
Итого				860

9.1.1 Расчёт затрат на оборудование

Расчёт ведётся по той же формуле, что и для материальных затрат.

Таблица 9.2 – Расходы на оборудование

Наименование оборудования	Единица измерения	Кол-во	Цена ед., руб.	Сумма, руб.
Настольный компьютер-сервер	шт.	1	41000	41000
Ноутбук	шт.	1	35340	35340
Итого				76340

9.2 Расчёт амортизационных отчислений

Амортизационные отчисления производятся предприятиями ежемесячно исходя из установленных норм амортизации и балансовой (первоначальной или восстановительной) стоимости основных фондов по отдельным группам или инвентарным объектам, состоящим на балансе предприятия. Нормы амортизации устанавливаются государством и они едины для всех предприятий и организаций.

В элементе «Амортизация основных фондов» отражается сумма амортизационных отчислений на полное восстановление основных производственных фондов, исчисленная из балансовой стоимости и утвержденных в установленном порядке норм, включая и ускоренную амортизацию их активной части, производимую в соответствии с законодательством.

$$A_{\text{ниокр}} = \frac{K_{\text{пс}} H_a t_{\text{об}}}{\Phi_{\text{д}}},$$

где $K_{\text{пс}}$ – остаточная стоимость основных фондов на начало соответствующего года, руб.;

H_a – норма годовых амортизационных отчислений, % (электронные цифровые машины общего назначения, специализированные и управляющие – 12,5%);

$t_{\text{об}}$ – время работы оборудования, $t_{\text{об}} = 8\text{ч/день} \cdot 111\text{дней} = 808\text{ч}$;

$\Phi_{\text{д}}$ – действительный годовый фонд рабочего времени, час/год, $\Phi_{\text{д}} = 8\text{ч/день} \cdot 5\text{дней} \cdot 52\text{недели} = 2080\text{ч/год}$.

$$A_{\text{ниокр}} = \frac{76340 \cdot 0,125 \cdot 808}{2080} = 3707\text{руб.}$$

9.3 Расчёт заработной платы

В статью «Основная заработная плата» включается основная заработная плата всех исполнителей, непосредственно занятых разработкой данной программной продукции, с учётом их должностного оклада и времени участия в разработке. Расчёт ведётся по формуле

$$C_{\text{зо}} = \sum_{i=1}^{n} З_i \cdot d$$

где $З_i$ – однодневный размер оплаты труда i-го исполнителя, руб./дн.; d – количество рабочих дней, отработанных исполнителем. В статье «Дополнительная заработная плата» учитываются все выплаты непосредственным исполнителям за время, не проработанное на

производстве, и определяются по формуле

$$C_{\text{зо}} = C_{\text{зо}} \cdot \alpha_{\text{д}}$$

где $\alpha_{\text{д}}$ – коэффициент отчислений на дополнительную зарплату, примем 0,2.

На начало 2013 года МРОТ составляет 4 330 рублей. Количество рабочих дней $d = 111$дней. Рассчитанные значения заработной платы приведены в таблице 9.3.

Таблица 9.3 – Расчёт заработной платы

Исполнитель	Месячный оклад, руб.	Дневная заработная плата, руб.	Продолжительность работы, дней	Основная заработная плата, руб.	Дополн. заработная плата, руб.
Инженер-программист	45000	2142	111	216342	43268

Получаем расходы на заработную плату

$$C_{\text{зп}} = C_{\text{зо}} + C_{\text{зд}} = 216342 + 43268 = 259610\text{руб.}$$

9.4 Расчёт отчислений в социальные фонды

Отчисления на социальные нужды определяются по формуле

$$C_{cc} = \alpha_{cc} \cdot (C_{\text{зо}} + C_{\text{зд}})$$

где α_{cc} – процентная ставка взносов. Ставка взносов при общей системе налогообложения разбивается по фондам как показано в таблице 9.4.

Таблица 9.4 – Расчёт отчислений в социальные фонды.

Наименование фонда		Ставка взносов, %
Пенсионный фонд	Страховая часть Накопительная часть	16 6
Фонд медицинского страхования	ФФОМС ТФОМС	2,1 3
Фонд социального страхования		2,9
Всего		30

Таким образом, получим

$$C_{\text{цф}} = \alpha_{cc} \cdot (C_{\text{зо}} + C_{\text{зд}}) = 0,3 * 259610\text{руб.} = 77883\text{руб.}$$

9.5 Прочие затраты

Расходы на интернет

$$C_{\text{пр1}} = 6 \cdot 800\text{руб.} = 4800\text{руб.}$$

Расходы на услуги аренды виртуального сервера

$$C_{\text{пр2}} = 200\text{руб.}$$

Расходы на печать документов

$$C_{\text{пр3}} = 4 \cdot 100 \cdot 4 = 1600\text{руб.}$$

Итого

$$C_{\text{пр}} = C_{\text{пр1}} + C_{\text{пр2}} + C_{\text{пр3}} = 4800 + 200 + 1600\text{руб.} = 5600\text{руб.}$$

10 Определение цены программного продукта

Результаты расчётов затрат на разработку программного продукта приведены в таблице 10.1.

Таблица 10.1 – Результаты расчёта затрат.

Статья затрат	Сумма, руб.
Материальные затраты	860
Амортизационные отчисления	3707
Заработная плата	259610
Отчисления в фонды	77883
Прочие затраты	5600

Таким образом, затраты на разработку программной продукции (сметная стоимость) составили С = 347660 руб.

Структура затрат приведена на рисунке 10.1.

Рисунок 10.1 – Структура затрат.

11 Выводы по организационно–экономической части дипломного проекта

Данная работа требует на свое выполнение 111 чел.-дней в лице одного программиста с заработной платой в 45000 руб./мес. Себестоимость проекта составляют 347660 тыс. руб. Цена является договорной и составляет 467660 руб. Коммерческая реализация ПП отсутствует, так как данный проект разрабатывался для структурного подразделения МГТУ им. Н. Э. Баумана. Применение данного ПП в иных организациях, или же возможность интегрировать его в другие программные комплексы отсутствует.

Охрана труда и экология

12 Проектирование рабочего места оператора ПЭВМ

Требования к компьютерной технике и к условиям работы с ней в Российской Федерации регламентируются санитарными нормами и правилами СанПиН 2.2.2/2.4.1340-03 и СанПиН 2.2.2.000-02. Рассмотрим основные нормы, необходимые для проектирования рабочего места оператора ПЭВМ.

12.1 Требования к рабочим помещениям

Согласно СанПиН 2.2.2/2.4.1340-03, помещения для работы с компьютерами должны оборудоваться системами отопления, кондиционирования воздуха или эффективной приточно-вытяжной вентиляцией. Звукоизоляция помещений и звукопоглощение ограждающих конструкций помещения должны отвечать гигиеническим требованиям и обеспечивать нормируемые параметры шума на рабочих местах. Помещения должны иметь естественное и искусственное освещение.

Поверхность пола в помещениях должна быть ровной, без выбоин, нескользкой, удобной для очистки и влажной уборки, обладать антистатическими свойствами. При строительстве новых и реконструкции действующих средних, средних специальных и высших учебных заведений помещения для работы с компьютером следует проектировать высотой (от пола до потолка) не менее 4,0 м.

Расположение рабочих мест для взрослых пользователей в подвальных помещениях не допускается.

12.2 Требования к освещению

Естественное освещение должно осуществляться через светопроемы, ориентированные преимущественно на север и северо-восток. Оконные проемы в помещениях использования компьютеров должны быть оборудованы регулируемыми устройствами типа жалюзи, занавесей, внешних

козырьков и др. Система естественного освещения должна обеспечивать коэффициент естественной освещенности (КЕО) не ниже 1,5.

Искусственное освещение в помещениях должно осуществляться системой общего равномерного освещения. В производственных и административно - общественных помещениях, в случаях преимущественной работы с документами, допускается применение системы комбинированного освещения (к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов).

Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 лк. Допускается установка светильников местного освещения для подсветки документов. Местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

Следует ограничивать прямую блескость от источников освещения, при этом яркость светящихся поверхностей (окна, светильники и др.), находящихся в поле зрения, должна быть не более 200 кд/кв. м. В качестве источников света при искусственном освещении должны применяться преимущественно люминесцентные лампы типа ЛБ. При устройстве отраженного освещения в производственных и административно-общественных помещениях допускается применение металлогалогенных ламп мощностью до 250 Вт. Допускается применение ламп накаливания в светильниках местного освещения.

Для освещения помещений следует применять светильники серии ЛПО36 с зеркализованными решетками, укомплектованные высокочастотными пускорегулирующими аппаратами (ВЧ ПРА). Коэффициент пульсации не должен превышать 5%, что должно обеспечиваться применением газоразрядных ламп в светильниках общего и местного освещения с ВЧ ПРА.

Допускается применять светильники серии ЛПО36 без ВЧ ПРА только в модификации «Кососвет», а также светильники прямого света – П, преимущественно прямого света – Н, преимущественно отраженного света – В. При этом лампы многоламповых светильников или рядом расположенные светильники общего освещения следует включать на разные фазы трехфазной сети.

Применение светильников без рассеивателей и экранирующих решеток не допускается. Яркость светильников общего освещения в зоне углов излучения от 50°до 90°с вертикалью в продольной и поперечной плоскостях должна составлять не более 200 кд/кв. м, защитный угол светильников должен быть не менее 40°. Светильники местного освещения должны иметь непросвечивающий отражатель с защитным углом не менее 40°.

Для обеспечения нормируемых значений освещенности в помещениях

использования ВДТ и ПЭВМ следует проводить чистку стекол оконных рам и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

Коэффициент запаса (Кз) для осветительных установок общего освещения должен приниматься равным 1,4.

Для внутренней отделки интерьера помещений должны использоваться диффузно-отражающие материалы с коэффициентом отражения

— для потолка $\rho_{\text{пот}} = 0,7 - 0,8$;

— для стен $\rho_{\text{ст}} = 0,5 - 0,6$;

— для пола $\rho_{\text{пол}} = 0,3 - 0,5$;

12.3 Расчёт системы освещения в помещении

В помещении, где находится рабочее место оператора, используется смешанное освещение, т.е. сочетание естественного и искусственного освещения. В качестве естественного – боковое освещение через окно. Искусственное освещение используется при недостаточном естественном освещении. В данном помещении используется общее искусственное освещение.

Расчет его осуществляется по методу светового потока с учетом потока, отраженного от стен и потолка.

Нормами для данных работ установлена необходимая освещенность рабочего места $E_{\text{н}} = 300$лк (средняя точность работы по различению деталей размером от 1 до 10 мм).

Площадь помещения:

$$S = A \cdot B = 4 \cdot 6 = 24\text{м}.$$

Общий световой поток определяется по формуле:

$$F = \frac{E \cdot K \cdot S \cdot Z}{n},$$

где E – нормированная минимальная освещённость, лк. Работа программиста относится к разряду точных работ, следовательно, минимально освещённость будет $E = 300$лк при газоразрядных лампах;

Z – отношение средней освещённости к минимальной (обычно принимается равным $1,1 - 1,2$, положим $Z = 1,1$);

K – коэффициент запаса, учитывающий уменьшение светового потока лампы в результате загрязнения светильников в процессе эксплуатации (его значение определяется по таблице коэффициентов запаса для различных помещений и в нашем случае $= 1,5$);

n – коэффициент использования (выражается отношением светового потока, падающего на расчетную поверхность, к суммарному потоку всех ламп и исчисляется в долях единицы; зависит от характеристик светильника, размеров помещения, окраски стен и потолка, характеризуемых коэффициентами отражения от стен P_c и потолка $P_{\text{п}}$). Значение коэффициентов P_c и $P_{\text{п}}$ определим по таблице зависимостей коэффициентов отражения от характера поверхности: $P_c = 30\%$, $P_{\text{п}} = 50\%$. Значение n определим по таблице коэффициентов использования различных светильников. Для этого вычислим индекс помещения по формуле:

$$I = \frac{S}{h \cdot (A + B)},$$

где S – площадь помещения, $S = 24\text{м}^2$;

h – расчётная высота подвеса, $h = 3\text{м}$;

A – длина помещения, $A = 6\text{м}$;

B – ширина помещения, $B = 4\text{м}$.

Подставив значения, получим:

$$I = \frac{24}{3 \cdot (6 + 4)} = 0,8$$

Зная индекс i, P_c, $P_{\text{п}}$, по таблице находим, что значения $n = 0,36$.

Подставим все значения в формулу для определения светового потока F:

$$F = \frac{300 \cdot 1,5 \cdot 24 \cdot 1,1}{0,36} = 33000\text{лм}$$

Люминесцентные лампы имеют ряд преимуществ перед лампами накаливания: их спектр ближе к естественному, обладают более высоким КПД (в 1,5-2 раза выше, чем КПД ламп накаливания), имеют большую экономичность (больше светоотдача) и срок службы (в 10-12 раз). Наряду с этим имеются и недостатки: их работа сопровождается иногда шумом; хуже работают при низких температурах; их нельзя применять во взрывоопасных помещениях; имеют малую инерционность. Для нашего помещения люминесцентные лампы подходят.

Для освещения выбираем люминесцентные лампы типа ЛБ40-1, световой поток которых F = 4320 лм.

Рассчитаем необходимое количество ламп по формуле:

$$N = \frac{F}{F_{\text{Л}}},$$

где N – определяемое число ламп;

F – световой поток, $F = 33000\text{лм}$;

$F_л$ – световой поток лампы, $F_л = 4320$лм.

$$N = \frac{33000}{4320} = 8\text{шт.}$$

При выборе осветительных приборов используем светильники типа ОД. Каждый светильник комплектуется двумя лампами ЛБ40. Размещаются светильники двумя рядами, по два в каждом ряду.

Рассчитаем суммарную мощность осветительной установки общего назначения:

$$P\sum = 40 \cdot 16 = 640\text{Вт.}$$

12.4 Требования к микроклимату

Оптимальным температурным режимом работы принимается температура окружающей среды от 19°до 21°и соответствующие им значения относительной влажности от 62% до 55%. Абсолютная влажность воздуха должна быть около 10г/м^3. Скорость движения воздуха требуется ограничить значением менее 0,1 м/с . Для наиболее производительной и успешной работы инженера-программиста (пользователя ПЭВМ), требуется обеспечить характеристики микроклимата по значениям, не уступающим вышеизложенным.

Вышеуказанные требования СанПиН были соблюдены на рабочем месте, где проходило выполнение данного дипломного проекта. Система отопления включала основную и вспомогательную системы. Основная система – система приточной вентиляции с подогревом воздуха, также выполняющая роль системы вентиляции помещения. Вспомогательная – статический обогрев помещения за счет батарей центрального отопления. Контроль за соблюдением техники безопасности и удовлетворением норм микроклимата регулярно проводится инженером по технике безопасности. Рабочее место аттестовано сертифицированными службами и допущено к эксплуатации.

13 Расчёт системы вентиляции

Помещения, в которых производится эксплуатации ПЭВМ, должны быть оборудованы системами приточно-вытяжной вентиляции, при работе которой будут обеспечены допустимые параметры микроклимата в помещении. Расчет производят только для вытяжной ветви приточно-вытяжной системы вентиляции.

$V_{вент}$ – объем воздуха, необходимый для обмена, м^3/ч; $V_{пом}$ – объем рабочего помещения, м^3. Для расчета примем следующие размеры рабочего помещения:

— длина $= 6$м;

— ширина $= 4$м;

— высота $= 3$м.

Соответственно, объем помещения равен:

$$V_{\text{пом}} = A \cdot B \cdot H = 72\text{м}^3.$$

Необходимый для обмена объём воздуха $V_{\text{вент}}$ определим исходя из уравнения теплового баланса:

$$V_{\text{вент}} \cdot (t_{\text{уход}} - t_{\text{приход}}) \cdot Y = 3600 \cdot Q_{\text{изб}},$$

где $Q_{\text{изб}}$ – избыточная теплота (Вт);
$C = 1005\text{Дж}/(\text{кг} \cdot \text{К})$ – удельная теплопроводная воздуха;
$Y = 1,2\text{мг}/\text{см}^3$ – плотность воздуха.
Температура уходящего воздуха определяется по формуле:

$$t_{\text{уход}} = t_{\text{рм}} + (H - 2) \cdot t,$$

где $t = 1 - 5^\circ$ – превышение t на 1м высоты помещения;
$t_{\text{рм}} = 21^\circ$ – температура на рабочем месте;
Н = 3м – высота помещения;

$$t_{\text{уход}} = 21 + (3 - 2) \cdot 4,7 = 25,47^\circ$$
$$Q_{\text{изб}} = Q_{\text{изб1}} + Q_{\text{изб2}} + Q_{\text{изб3}},$$

где $Q_{\text{изб}}$ – избыток тепла от электрооборудования и освещения.

$$Q_{\text{изб1}} = E \cdot p$$

где E – коэффициент потерь электроэнергии на теплоотвод ($E = 0,55$ для освещения);
p – мощность, $p = 40\text{Вт} \cdot 8 + 500\text{Вт} = 820\text{Вт}$.

$$Q_{\text{изб1}} = 0,55 \cdot 820 = 461\text{Вт},$$

$Q_{\text{изб2}}$ – теплопоступление от солнечной радиации,

$$Q_{\text{изб2}} = m \cdot S \cdot k \cdot Q_c,$$

где m – число окон, примем $m = 2$;
S – площадь окна, $S = 2,2 \cdot 1,5 = 3,3\text{м}^2$;

k – коэффициент, учитывающий остекление (для двойного остекления $k = 0,6$);

$Q_c = 127$Вт/м – теплопоступление от окон.

$$Q_{\text{изб2}} = 3,3 \cdot 2 \cdot 0,6 \cdot 127 = 505,92\text{Вт}.$$

$Q_{\text{изб3}}$ – тепловыделение людей.

$$Q_{\text{изб3}} = n \cdot q,$$

где $q = 80$Вт/чел.,

n – число людей ($n = 1$).

Тогда

$$Q_{\text{изб3}} = 1 \cdot 80 = 80\text{Вт}$$
$$Q_{\text{изб}} = 461 + 502 + 80 = 1143\text{Вт}$$

Из уравнения теплового баланса следует:

$$V_{\text{вент}} = \frac{3600 \cdot 1143}{1005 \cdot 1,2 \cdot (25,47 - 19)} = 527,35\text{м}^3/\text{час}$$

13.1 Выбор вентилятора

В нашем случае будет использоваться приточно-вытяжная вентиляция.

Вентиляционная система состоит из следующих элементов (рисунок 13.1):

1) забор воздуха;

2) система кондиционирования;

3) вентилятор;

4) решетка;

5) стальной воздуховод с круглым сечением;

6) стальной воздуховод с круглым сечением;

7) выброс воздуха.

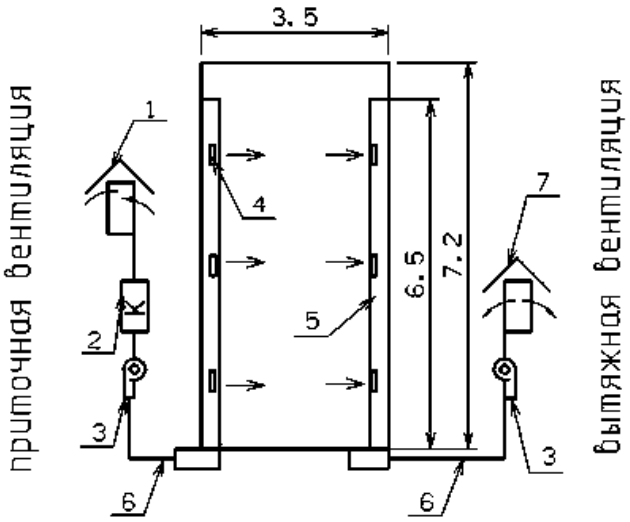


Рисунок 13.1 – Вентиляционная система.

Исходными данными для выбора вентилятора является расчётная производительность вентилятора:

$$V_{\text{расч}} = 1,1 \cdot V_{\text{вент}} = 1,1 \cdot 527 = 579,7\text{м}^3/\text{час},$$

где $1,1$ – коэффициент, учитывающий утечки и подсосы воздуха.

Потери давления в вентиляционной системе определяются по формуле:

$$H = R \cdot l + \xi \cdot \frac{V^2 \cdot p}{2},$$

где H – потери давления, Па;

R – увдельные потери давления на трение в воздуховоде, Па/м;

$I = 4,5\text{м}$ – длина воздуховода;

$V = 7\text{м/с}$ – скорость воздуха;

$p = 1,2\text{кг/м}$ – плотность воздуха.

Необходимый диаметр воздуховода для данной вентиляционной системы:

$$L = \frac{d^2 \cdot \pi}{4} \cdot W,$$

где $L = V_{\text{расч}} = 579,7\text{м}^3/\text{час}$;

$W = 7\text{м/с}$.

$$d = \sqrt{\frac{4 \cdot L}{\pi \cdot W}} = 0,17\text{м}.$$

Принимаем в качестве диаметра ближайшую большую стандартную величину – 0,2 м, при которой удельные потери давления на трение в воздуховоде – $R = 3,58\text{Па/м}$.

После перехода на стандартный диаметр производим перерасчёт скорости:

$$W_{\text{нор}} = \frac{4}{d^2 \cdot \pi} \cdot L = 5,12\text{м/с}.$$

Местные потери возникают в железной решётке ($\xi = 1,2$), в изгибе трубопровода ($\xi = 1,7$). Отсюда суммарный коэффициент местных потерь в системе:

$$\xi = (1,2 \cdot 3) + 1,7 = 5,3.$$

Тогда

$$H = 3,58 \cdot 4,5 + 5,3 \cdot \frac{26,2 \cdot 1,2}{2} = 99,4\text{Па}.$$

С учётом 10% запаса:

$$H = 1,1 \cdot 99,4 = 109,3\text{Па}.$$

Далее в проводимом расчёте по вычисленному напору определяется марка вентилятора из соответствующих таблице. Рассмотрим, например, соответствующую таблицу для вентиляторов радиальных (центробежных) серии ВЦ-4-70 (таблица 13.1).

Таблица 13.1 – Вентиляторы радиальные (центробежные) серии ВЦ-4-70.

Марка вент.	Электродвигатель		Характеристики вентилятора		Исполнение			
	Мощность, кВт	Частота вращения, об./мин.	Расход воздуха, $м^3 \cdot 10^3$/час	Напор, Па	Обычное	Коррозионноестойкое	Искрозащищённое	Взрывозащищённое
ЦВ-4-70-2,5	0,18	1500	0,37-0,92	140-190	+	+	+	+
	0,75	3000	0,75-1,92	500-760	+	+	+	+
ЦВ-4-70-3,15	0,27	1500	0,78-2,0	190-310	+	+	+	+
	1,5	3000	1,65-4,10	800-1300	+	+	+	+
ЦВ-4-70-4	0,18	950	1,15-2,6	130-220	+	+	+	+
	1,1	1500	1,85-4,0	300-490	+	+	+	+
	7,5	3000	3,6-7,2	1200-2250	+	+	+	+
ЦВ-4-70-5	0,55	950	2,3-5,1	160-360	+	+	+	+
	2,2	1500	3,5-8,0	380-840	+	+	+	+
ЦВ-4-70-6,3	1,5	950	4,2-10,2	310-500	+	+	+	+
	5,5	1500	7,0-16,0	580-1280	+	+	+	+

По этим данным выбираем вентилятор ВЦ-4-70-2,5:

— расход воздуха: $370 - 920 м^3$/час;

— давление: 140-190 Па;

— скорость вращения: 1500 об./мин.;

— мощность электродвигателя: 0,18 кВт.

13.2 Требования к размещению оборудования

Площадь на одно рабочее место с компьютером для взрослых пользователей должна составлять не менее $6,0 м^2$, а объем – не менее $20,0 м^2$. Площадь на одно рабочее место с компьютером во всех учебных и

дошкольных учреждениях должна быть не менее $6,0\text{м}^2$, а объем – не менее $24,0\text{м}^3$.

Рабочие места по отношению к световым проемам должны располагаться так, чтобы естественный свет падал сбоку, преимущественно слева. Схемы размещения рабочих мест с ВДТ и ПЭВМ должны учитывать расстояния между рабочими столами с видеомониторами (в направлении тыла поверхности одного видеомонитора и экрана другого видеомонитора), которое должно быть не менее $2,0\text{м}$, а расстояние между боковыми поверхностями видеомониторов – не менее $1,2\text{м}$.

Рабочие места в помещениях с источниками вредных производственных факторов должны размещаться в изолированных кабинах с организованным воздухообменом. Рабочие места с ВДТ и ПЭВМ при выполнении творческой работы, требующей значительного умственного напряжения или высокой концентрации внимания, следует изолировать друг от друга перегородками высотой $1,5-2,0\text{м}$.

Размещение оборудования на рабочих столах должно отвечать следующим требованиям.

При работе с монитором в глазу происходят процессы аккомодации и конвергенции. Аккомодация – изменение геометрии хрусталика, при котором изображение фокусируется на сетчатке. Раньше считалось, что расслабление мышц глаза и хрусталика происходит, когда глаз смотрит в бесконечность. Более свежие данные говорят о том, что расслабление соответствует фокусировке на «мнимый» объект на расстоянии примерно 800 мм. Мышцы, поворачивающие глазные яблоки в процессе конвергенции тоже находятся в постоянном напряжении. Новые данные говорят о том, что расслабление глазодвигательных мышц происходит при сведении глазных яблок на «мнимый» объект, удаление которого зависит от угла зрения. Пристальный взгляд под углом 30°расслабляет глазодвигательные мышцы при удалении предмета на 530 мм. Поэтому экран видеомонитора должен находиться на расстоянии 600-700 мм от глаз пользователя на высоте его головы.

Зона досягаемости составляет 350-400 мм. Ближней зоне соответствует область, охватываемая рукой при прижатом к туловищу локте, дальней зоне – область вытянутой руки. Поэтому клавиатуру следует располагать на поверхности стола на расстоянии 100-300 мм от края, обращенного к пользователю или на специальной выдвижной панели стола.

На основе данных о физиологии человека были разработаны оптимальные параметры элементов рабочего места, которые в настоящее время стандартизированы и включены в санитарно-гигиенические и эргономические нормативные документы. Для того чтобы обеспечивать свободную и удобную рабочую позу (оптимальные условия труда) элементы рабочего места должны удовлетворять требованиям СанПиН 2.2.2/2.4.1340-

03. В табл. 13.2 приведены оптимальные размеры основных элементов рабочего места (рабочий стол и стул).

Таблица 13.2 – Параметры оптимального рабочего места пользователя ПК.

Элемент рабочего места	Параметры	Величина (мм)	Диапазон рег-я (мм)
Рабочий стол	Высота рабочей поверхности	725	680-800
	Ширина	800, 1000, 1200, 1400	нет
	Пространство для ног		
	высота	600	нет
	глубина на уровне колен	450	нет
	глубина на уровне вытянутых ног	650	нет
Рабочий стул (подъёмно-поворотный)	Ширина сиденья	400	нет
	Глубина сиденья	400	нет
	Высота поверхности сиденья	475	400-550
	Угол наклона сиденья		
	вперёд	0°	0°-15°
	назад	0°	0°-15°
	Высота опорной пов-ти спинки	300	280-320
	Ширина спинки	380	нет
	Радиус кривизны спинки в гор. плоскости	400	нет
	Угол наклона спинки в вертикальной плоскости	0°	от -30° до +30°
	Расстояние от переднего края сиденья до спинки	330	260-400
Подлокотники (съёмные или стационарные)	Длина	250	нет
	Ширина	50-70	нет
	Высота над сиденьем	230	200-260
	Расстояние между подлокотниками	425	350-500
Подставка для ног	Ширина	300	нет
	Глубина	400	нет
	Высота	150	нет

Элемент рабочего места	Параметры	Величина (мм)	Диапазон рег-я (мм)
	Наклон опорной поверхности	0°	0°-20°

Исходя из данных таблицы, приведём параметры стола программиста:

— высота стола – 725 мм;

— длина стола – 1200 мм;

— ширина стола – 800 мм;

— глубина стола – 400 мм;

— поверхность для письма:

  — в глубину – 40 мм;

  — в ширину – 600 мм;

13.3 Требования к мониторам

Изображение на экране монитора может быть охарактеризовано широким набором параметров. С точки зрения безопасности, первостепенную роль имеют факторы, определяющие характер зрительной работы и вызывающие усталость и нагрузку на зрительный аппарат пользователя компьютера. Эти факторы называются визуальными эргономическими параметрами. К ним относится яркость изображения, четкость изображения, размер знаков, отклонение положения знаков, искажения изображения, мерцание изображения. К визуальным параметрам также относится отражательная способность экрана, обрамления экрана и корпуса монитора. В таблице 13.3 приведены описания нормируемых параметров и их оптимальные значения.

Таблица 13.3 – Визуальные эргономические параметры по СанПиН 2.2.2/2.4.1340-03.

Параметр	Описание и определение	Допустимое значение
Яркость изображения	Яркость знака или фона, измерененная в темноте	от 35 до 120 кд/м^2

Параметр	Описание и определение	Допустимое значение
Чёткость изображения	Размер минимального элемента (зерна) изображения	не более 0,3 мм
Размер знаков	Отношение ширины знака к его высоте для прописных букв	от 0,7 до 0,9 оптимально от 0,5 до 1,0 допустимо
Отклонение положения знаков	Допустимое горизонтальное смещение однотипных знаков в % от ширины знака Допустимое вертикальное смещение однотипных знаков в % от высоты знака	не более 5 не более 5
Отклонение формы рабочего поля экрана	По горизонтали: $\Delta B = \frac{2(B_1-B_2)}{(B_1+B_2)}$ По вертикали: $\Delta H = \frac{2(H_1-H_2)}{(H_1+H_2)}$ По диагонали: $\Delta D = \frac{2(D_1-D_2)}{(D_1+D_2)}$	$\Delta B \leq 0,02$ $\Delta H \leq 0,02$ $\Delta D \leq 0,04 \cdot (H_1 - H_2)$
Мерцание изображения	Допустимая временная нестабильность изображения	не должна быть зафиксирована 90% наблюдателей
Отражательная способность	Зеркальное и смешанное отражение (блики) экрана и обрамления экрана	не более 1%

B_1 и B_2 – длины верхней и нижней строк текста на рабочем поле экрана, мм; H_1 и H_2 – длины крайних столбцов на рабочем поле экрана, мм; D_1 и D_2 – длины диагоналей рабочего поля экрана, мм.

Дизайн монитора и системного блока должен предусматривать окраску корпуса в спокойные мягкие тона с диффузным (рассеянным) отражением света. Все поверхности должны быть одного цвета с коэффициентом отражения 0,4-0,6 и не иметь блестящих деталей, способных создавать блики.

Важнейшим визуальным эргономическим параметром, доступным для настройки и регулировки пользователем и определяющим утомляемость зрительного аппарата, является мерцание изображения. Субъективное

восприятие мерцания напрямую связано с частотой обновления изображения (refresh mode), но зависит также от других факторов (направления взора, контраста, яркости и цветовой гаммы изображения, динамичности изображения). К тому же разные люди имеют неодинаковую субъективную граничную частоту обновления при прочих равных условиях. Поэтому СанПиН не регламентируют оптимальное значение частоты обновления, которое будет различаться в зависимости от изображения на экране монитора, расположения пользователя, световой среды в рабочем помещении, СанПиН требует отсутствия субъективного восприятия мерцания изображения не менее чем у 90% пользователей.

Таким образом, для определения оптимальное частоты обновления изображения в каждом конкретном случае визуальной работы необходимо проводить экспериментальное исследование. Указанный способ достаточно трудоемкий, поэтому представляется целесообразным установить минимальную физиологическую граничную частоту обновления для самого худшего случая зрительной работы. За нее можно принять величину 78-80 Гц.

Из СанПиН 2.2.2/2.4.1340-03 следует, что использование мониторов и программного обеспечения, поддерживающих частоту обновления ниже 75 Гц, недопустимо. Эти требования учитываются производителями мониторов. В настоящее время подавляющее большинство ЭЛТ-мониторов поддерживает частоту обновления 85 Гц при разрешении 1280 x 800, а наибольшее распространение получили ЖК-мониторы, в которых эффект мерцания отсутствует.

13.4 Требования по электробезопасности

Защита от поражения электрическим током обеспечивается различными способами, в том числе:

— размещением разъемов электропитания на тыльной стороне истемного блока и монитора;

— применением надежных изоляционных материалов;

— использованием кабелей электропитания с заземляющими роводниками;

— использованием для электропитания устройств управления низковольтных напряжений (не более 12В).

Системный блок и монитор подключены к трехфазной четырёхпроводной сети переменного тока с заземлённой нейтралью,

напряжением 220 В и частотой 50 Гц, нетоковедущие корпуса монитора и системного блока занулены.

Зануление электроустановки – преднамеренное электрическое присоединение её частей, нормально не находящихся под напряжением, к нейтрали трансформатора через нулевой провод сети.

При повреждении изоляции зануленного электрооборудования цепь аварийного тока замыкания имеет малое сопротивление, равное сумме сопротивлений фазного и нулевого проводов сети. Ток в этом случае значительно больше, чем при использовании только заземления, и защитная аппаратура сработает эффективнее. Быстрое и полное отключение поврежденного оборудования – основное назначение зануления.

При обрыве нулевого провода на всех зануленных посредством этого провода корпусах появится напряжение, так как через остаток нулевого провода и хотя бы один включенный потребитель оказываются подключенными к фазному проводу. Поэтому недопустима установка в нулевой провод внутренней сети помещения предохранителей и автоматических выключателей, которые разрывали бы его при срабатывании.

В сети 380/220 В недопустимо применять только заземление одних аппаратов и зануление других, так как в случае повреждения изоляции заземленного аппарата на нулевом проводе и зачтенном оборудовании может появиться напряжение. Заземленный корпус аппарата должен иметь металлическое соединение с нулевым проводом сети.

Дополнительными мерами при проектировании рабочего места пользователя являются применение правил электробезопасности при эксплуатации электрических приборов.

14 Выводы

В экологической части дипломного проекта было выполнено проектирование рабочего места оператора ЭВМ. Созданные условия должны обеспечивать комфортную работу. На основании принятых требований по технике безопасности и санитарных норм были указаны размеры рабочего стола и кресла, рабочей поверхности, проведен выбор системы и расчет оптимального освещения помещения, а также расчёт вентиляции помещения.

В результате расчетов количество люминесцентных ламп типа ЛБ40-1 должно равняться 16шт. Для вытяжной ветви приточно-вытяжной системы вентиляции должен использоваться вентилятор ВЦ-4-70-2,5.

Также были изложены стандарты безопасности на мониторы, на основании которых был выбран монитор для разработки дипломного проекта. Соблюдение условий, определяющих оптимальную

организацию рабочего места программиста, позволит сохранить хорошую работоспособность в течение всего рабочего дня, повысит производительность труда программиста как в количественном, так и в качественном отношении.

Список использованных источников

1. Predictive Runtime Code Scheduling for Heterogeneous Architectures / Víctor J. Jiménez, Lluís Vilanova, Isaac Gelado et al. // Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers. — HiPEAC '09. — Berlin, Heidelberg : Springer-Verlag, 2009. — P. 19–33. — URL: http://dx.doi.org/10.1007/978-3-540-92990-1_4.

2. Finding representative sets of optimizations for adaptive multiversioning applications / Lianjie Luo, Yang Chen, Chengyong Wu et al. // International Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion. — Paphos, Chypre, 2009. — Январь. — URL: http://hal.inria.fr/inria-00436034.

3. Compilers: Principles, Techniques, and Tools (2nd Edition) / Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2006. — ISBN: 0321486811.

4. Muchnick Steven S., Gibbons Phillip B. Efficient instruction scheduling for a pipelined architecture // SIGPLAN Not. — 2004. — Апрель. — Vol. 39, no. 4. — P. 167–174. — URL: http://doi.acm.org/10.1145/989393.989413.

5. The Importance of Prepass Code Scheduling for Superscalar and Superpipelined Processors / Daniel M. Lavery, Pohua P. Chang, Scott A. Mahlke et al. // IEEE Trans. Comput. — 1995. — Март. — Vol. 44, no. 3. — P. 353–370. — URL: http://dx.doi.org/10.1109/12.372029.

6. Abraham Santosh G., Kathail Vinod, Deitrich Brian L. Meld scheduling: relaxing scheduling constraints across region boundaries // Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture. — MICRO 29. — Washington, DC, USA : IEEE Computer Society, 1996. — P. 308–321. — URL: http://dl.acm.org/citation.cfm?id=243846.243903.

7. Stephenson Mark W. Automating the construction of compiler heuristics using machine learning : Ph. D. thesis / Mark W. Stephenson. — Cambridge, MA, USA : Massachusetts Institute of Technology, 2006. — AAI0810106.

8. Meta optimization: improving compiler heuristics with machine learning / Mark Stephenson, Saman Amarasinghe, Martin Martin, Una-May O'Reilly // SIGPLAN Not. — 2003. — Май. — Vol. 38, no. 5. — P. 77–90. — URL: http://doi.acm.org/10.1145/780822.781141.

9. Using Machine Learning to Focus Iterative Optimization / F. Agakov, E. Bonilla, J. Cavazos et al. // Proceedings of the International Symposium on Code Generation and Optimization. — CGO '06. — Washington, DC, USA : IEEE Computer Society, 2006. — P. 295–305. — URL: http://dx.doi.org/10.1109/CGO.2006.37.

10. Bodin F., Kisuki T., Knijnenburg P. M. W. et al. Iterative Compilation in a Non-Linear Optimisation Space. — 1998.

11. Rapidly Selecting Good Compiler Optimizations using Performance Counters / J. Cavazos, G. Fursin, F. Agakov et al. // ACM International Conference on Code Generation and Optimization (CGO'07). — San Jose, California, 2007. — Март. — P. 185–197.

12. ACME: adaptive compilation made efficient / Keith D. Cooper, Alexander Grosul, Timothy J. Harvey et al. // Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems. — LCTES '05. — New York, NY, USA : ACM, 2005. — P. 69–77. — URL: http://doi.acm.org/10.1145/1065910.1065921.

13. A Feasibility Study in Iterative Compilation / Toru Kisuki, Peter M. W. Knijnenburg, Michael F. P. O'Boyle et al. // Proceedings of the Second International Symposium on High Performance Computing. — ISHPC '99. — London, UK, UK : Springer-Verlag, 1999. — P. 121–132. — URL: http://dl.acm.org/citation.cfm?id=646347.690219.

14. Portable compiler optimisation across embedded programs and microarchitectures using machine learning / Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla et al. // Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. — MICRO 42. — New York, NY, USA : ACM, 2009. — P. 78–88. — URL: http://doi.acm.org/10.1145/1669112.1669124.

15. Dubach Christophe, Jones Timothy M., O'Boyle Michael F.P. Exploring and predicting the architecture/optimising compiler co-design space // Proceedings of the 2008 international conference on Compilers, architectures and synthesis for

embedded systems. — CASES '08. — New York, NY, USA : ACM, 2008. — P. 31–40. — URL: http://doi.acm.org/10.1145/1450095.1450103.

16. Fursin Grigori. Collective Tuning Initiative: automating and accelerating development and optimization of computing systems // Proceedings of the GCC Developers' Summit. — MICRO 42. — New York, NY, USA : ACM, 2009. — P. 78–88. — URL: http://doi.acm.org/10.1145/1669112.1669124.

17. The LLVM Compiler Infrastructure // http://llvm.org/.

18. Fursin G., O'Boyle M., Knijnenburg P. Evaluating Iterative Compilation // Languages and Compilers for Parallel Computing / Ed. by Bill Pugh, Chau-Wen Tseng. — Springer Berlin / Heidelberg, 2005. — Vol. 2481 of Lecture Notes in Computer Science. — P. 362–376. — 10.1007/11596110:24. URL: http://dx.doi.org/10.1007/11596110:24.

19. М.Б.Лагутин. Наглядная математическая статистика : учебное пособие — 2-е изд., испр. — М.: БИНОМ. Лаборатория знаний, 2012. — 472 с. : ил.

20. Норман Дрейпер Гарри Смит. Прикладной регрессионный анализ — 3-е изд. — М.: «Диалектика», 2007. — 912 с.

21. Станислав Радченко. Методология регрессионного анализа: Монография. — К.: «Корнийчук», 2011. — 376 с.

22. Станислав Радченко. Устойчивые методы оценивания статистических моделей: Монография. — К.: ПП «Санспарель», 2005. — 504 с.

23. Перекрестная проверка. Статья в словаре BaseGroup Labs (электронный ресурс). — (Дата обращения: 17 марта 2013). URL: http://www.basegroup.ru/glossary/definitions/cross_validation/.

24. Variable importance (электронный ресурс). — (Дата обращения: 03 июня 2013). URL: http://caret.r-forge.r-project.org/varimp.html.

25. Grid Search: setting estimator parameters (электронный ресурс). — (Дата обращения: 19 марта 2013). URL: http://scikit-learn.org/0.13/modules/grid_search.html.

26. Test-driven development (электронный ресурс). — Материал из Википедии – Свободной Энциклопедии. (Дата обращения: 03 января 2013). URL: http://en.wikipedia.org/wiki/Test-driven_development.

27. Python Programming Language (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 03 января 2013). URL: http://www.python.org/.

28. TIOBE Programming Community Index for December 2012 (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 04 января 2013). URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

29. PyPy (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 03 января 2013). URL: http://pypy.org/.

30. Scientific Tools for Python (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 04 января 2013). URL: http://www.scipy.org/.

31. Matplotlib (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 03 января 2013). URL: http://matplotlib.org/.

32. Orange. Open source data visualization and analysis for novice and experts (электронный ресурс). — (Дата обращения: 19 марта 2013). URL: http://orange.biolab.si/.

33. scikit-learn: machine learning in Python (электронный ресурс). — (Дата обращения: 19 марта 2013). URL: http://scikit-learn.org/stable/.

34. Distributed revision control (электронный ресурс). — Материал из Википедии – Свободной Энциклопедии. (Дата обращения: 03 января 2013). URL: http://en.wikipedia.org/wiki/Distributed_revision_control.

35. Github — Powerful collaboration, review, and code management for open source and private development projects (электронный ресурс). — Веб-сайт. (Дата обращения: 04 января 2013). URL: https://github.com/.

36. Bitbucket — Unlimited private repositories to collaborate on your code – Git or Mercurial (электронный ресурс). — Веб-сайт. (Дата обращения: 04 января 2013). URL: https://bitbucket.org/.

37. Analysis of Git and Mercurial (электронный ресурс). — Веб-сайт. (Дата обращения: 04 января 2013). URL: http://code.google.com/p/support/wiki/DVCSAnalysis.

38. JSON (электронный ресурс). — Официальный веб-сайт. (Дата обращения: 04 января 2013). URL: http://www.json.org/.

39. Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris comparison (электронный ресурс). — Статья. (Дата обращения: 04 января 2013). URL: http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis.

40. MapReduce (электронный ресурс). — Материал из Википедии — свободной энциклопедии. (Дата обращения: 04 января 2013). URL: http://ru.wikipedia.org/wiki/MapReduce.

41. CouchApp.org: Simple JavaScript Applications with CouchDB (электронный ресурс). — Официальный сайт. (Дата обращения: 04 января 2013). URL: http://couchapp.org/page/index.

42. GCC — Option Summary (электронный ресурс). — Официальный вебсайт. (Дата обращения: 04 января 2013). URL: http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html.

43. Breecher Jerry. Operating Systems Scheduling (электронный ресурс). — Лекция №5 из курса Operating Systems. (Дата обращения: 04 января 2013). URL: http://web.cs.wpi.edu/~cs3013/c07/lectures/Section05-Scheduling.pdf.

44. PolyBench/C the Polyhedral Benchmark suite (электронный ресурс). — Официальный сайт. (Дата обращения: 04 января 2013). URL: http://www.cse.ohio-state.edu/~pouchet/software/polybench/.

45. IPython (электронный ресурс). — Официальный сайт. (Дата обращения: 04 января 2013). URL: http://ipython.org/.

46. Ю.Б.Сажин С.В.Самохин. Методическое пособие по выполнению организационно-экономической части дипломных проектов по разработке и использованию программных продуктов. — Москва : Издательство МГТУ, 2004.

Приложение

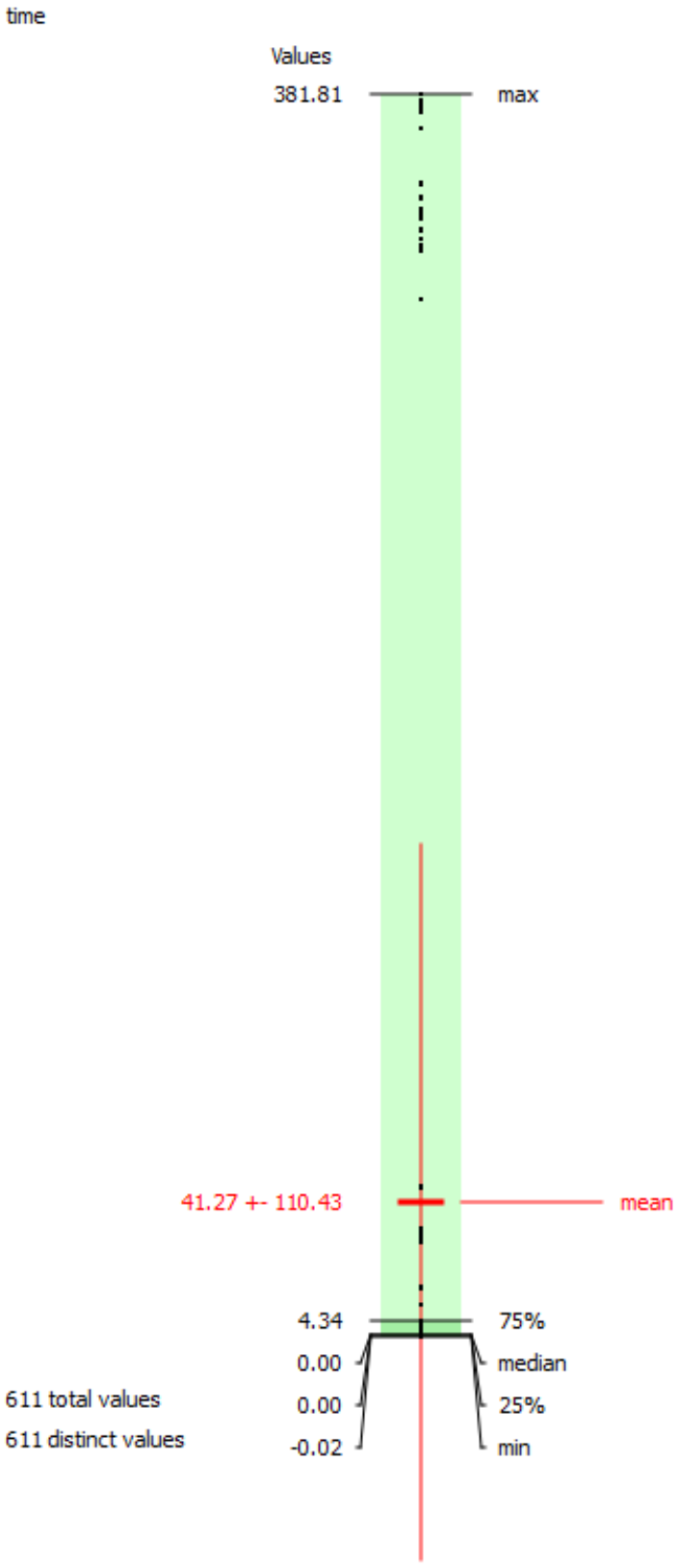


Рисунок 14.1 – Компонент `2. Attribute Statistics`. Статистические показатели свойства `time`.

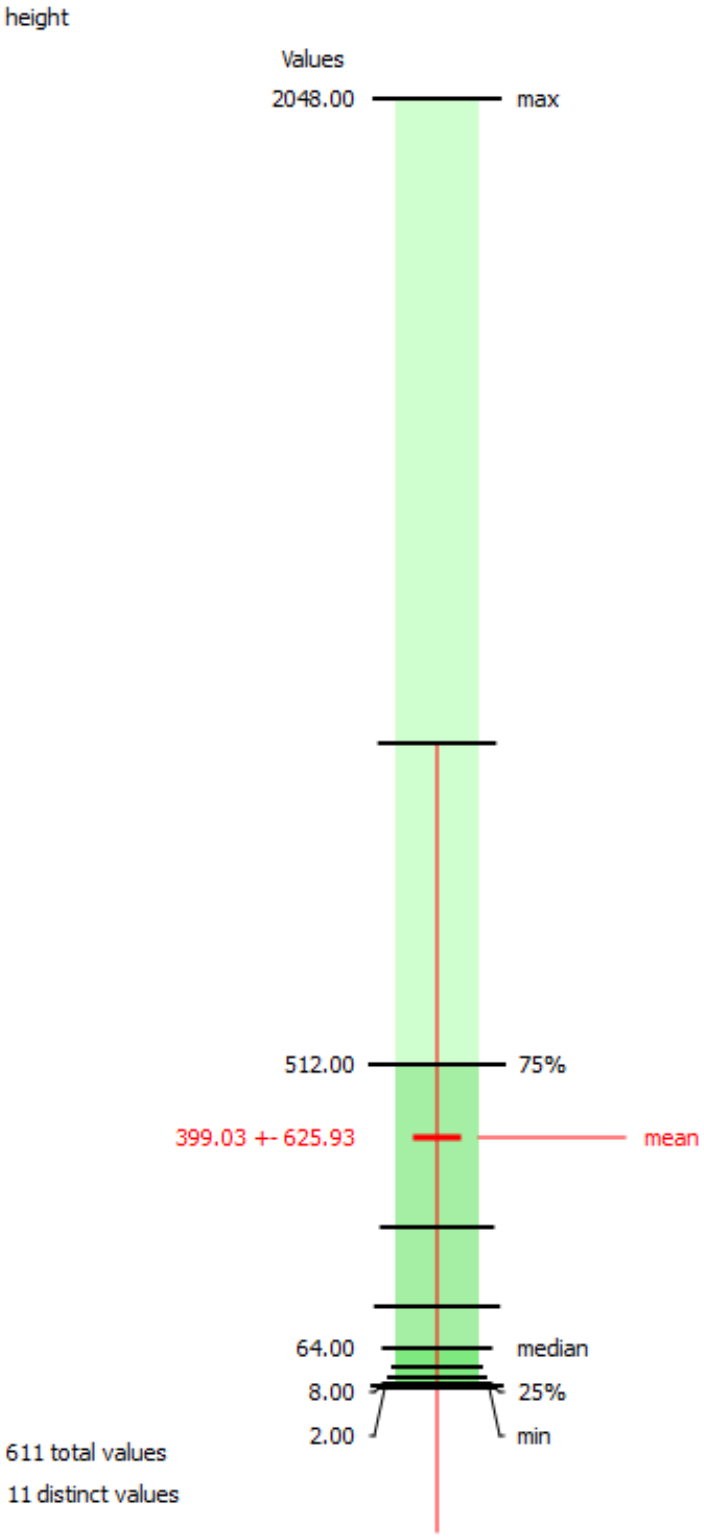


Рисунок 14.2 – Компонент `2. Attribute Statistics`. Статистические показатели свойства `height`.

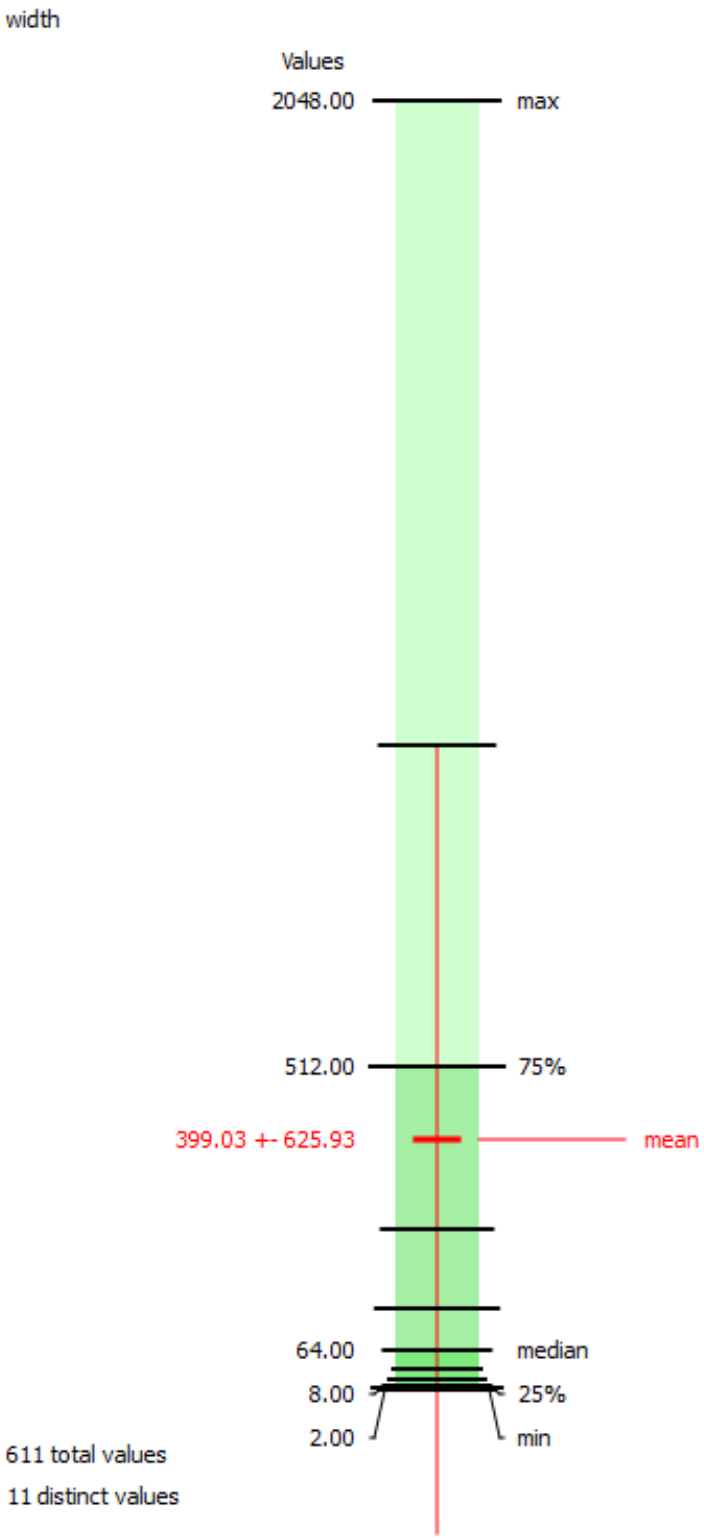


Рисунок 14.3 – Компонент `2. Attribute Statistics`. Статистические показатели свойства `width`.

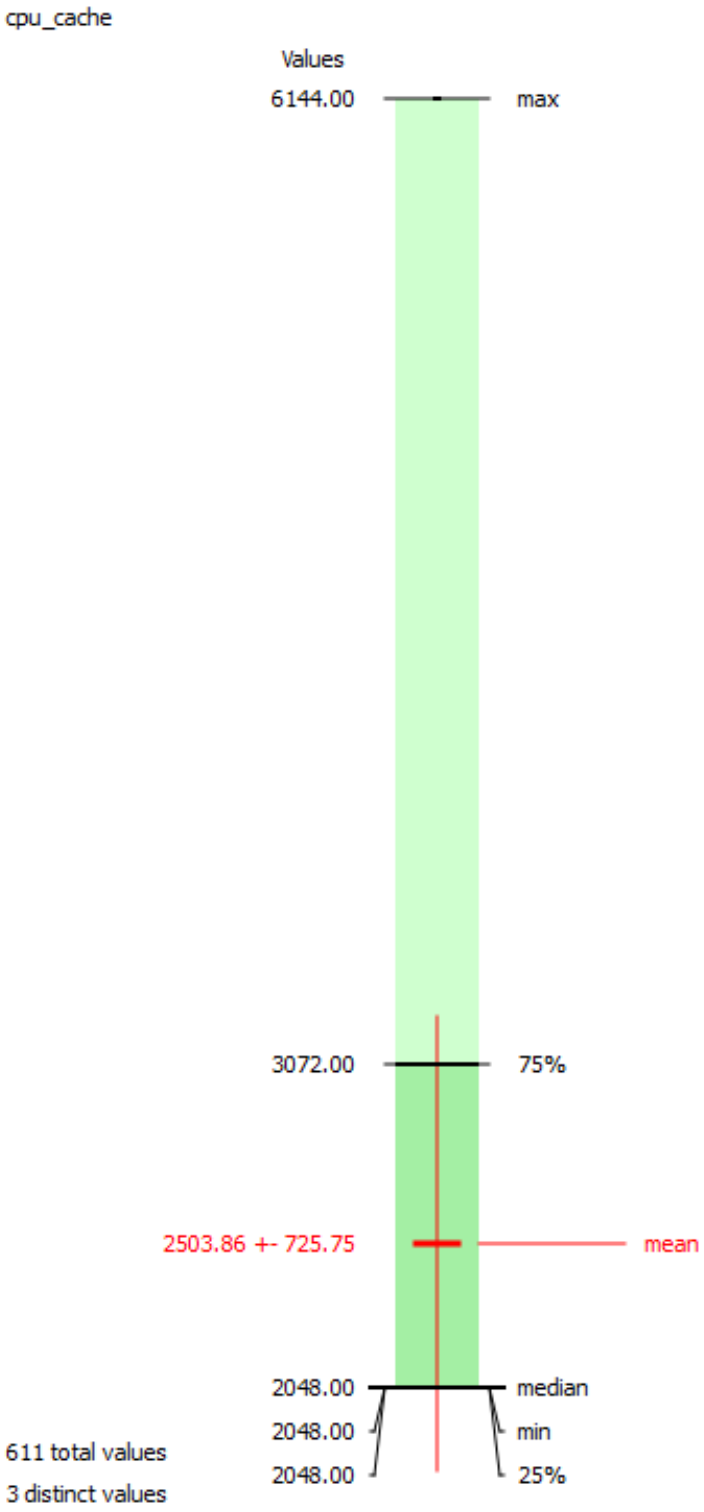


Рисунок 14.4 – Компонент `2. Attribute Statistics`. Статистические показатели свойства `cpu_cache`.

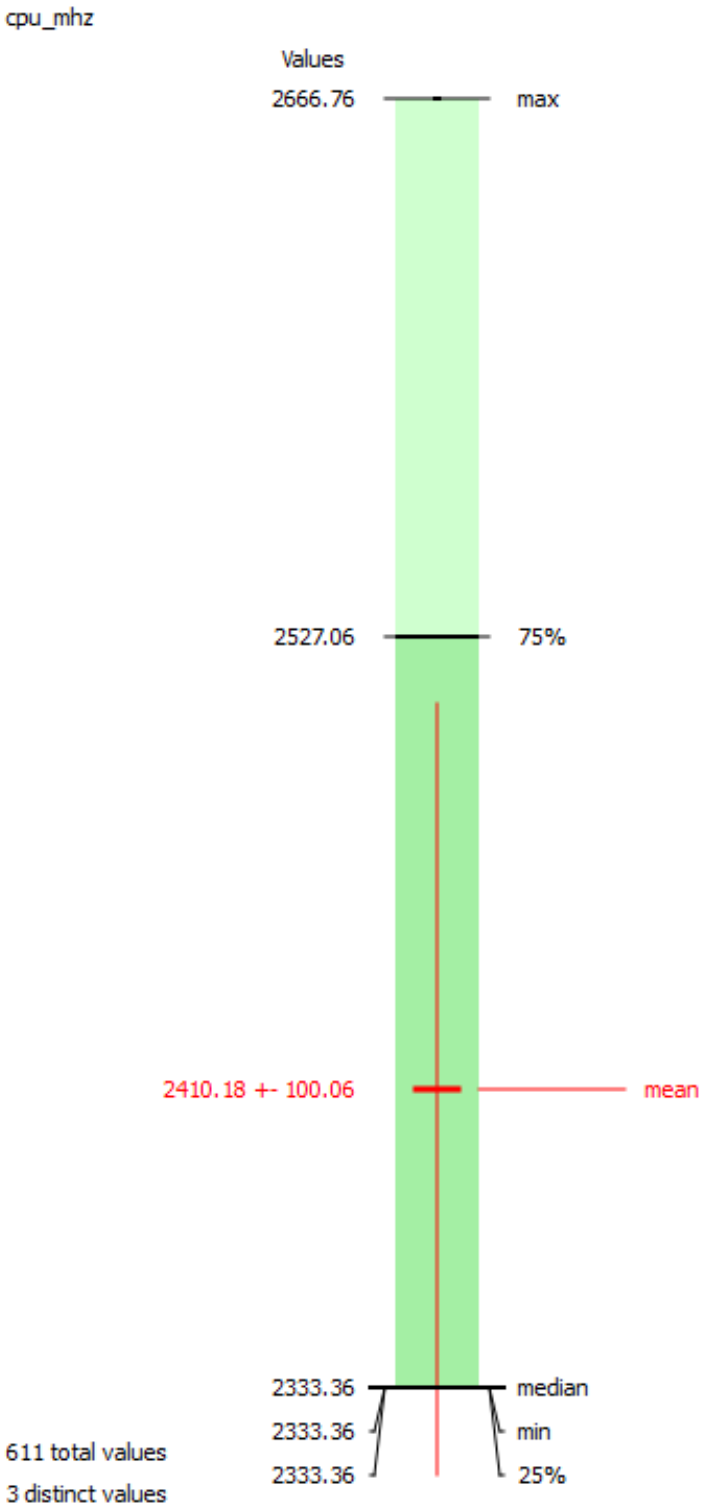


Рисунок 14.5 – Компонент `2. Attribute Statistics`. Статистические показатели свойства `cpu_mhz`.

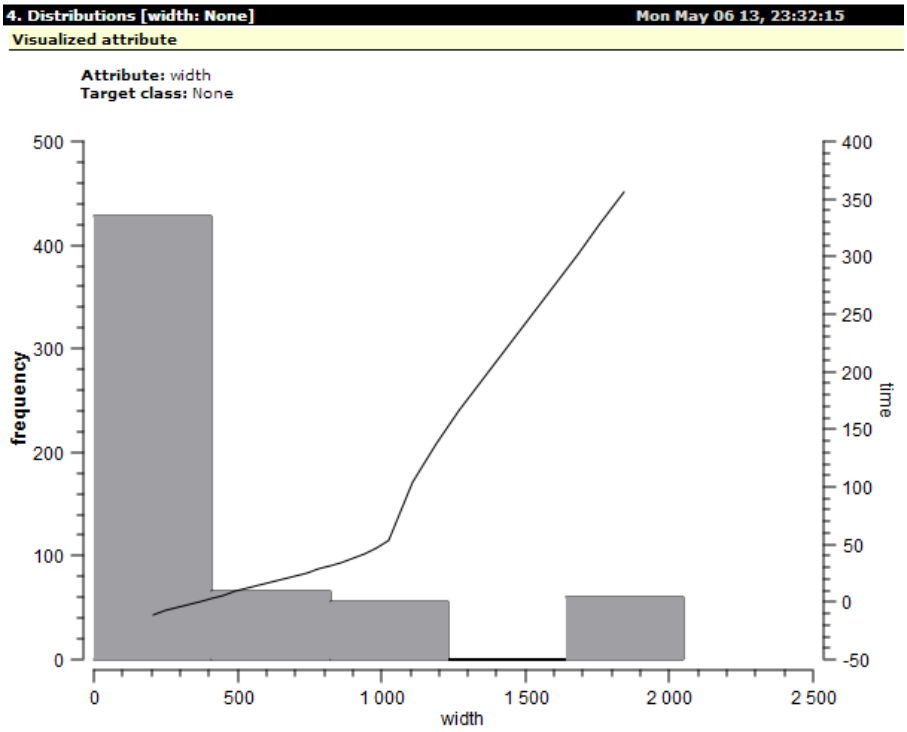


Рисунок 14.6 – Компонент `4. Distributions`. Распределение свойства `width` и его влияние на свойство `time`.

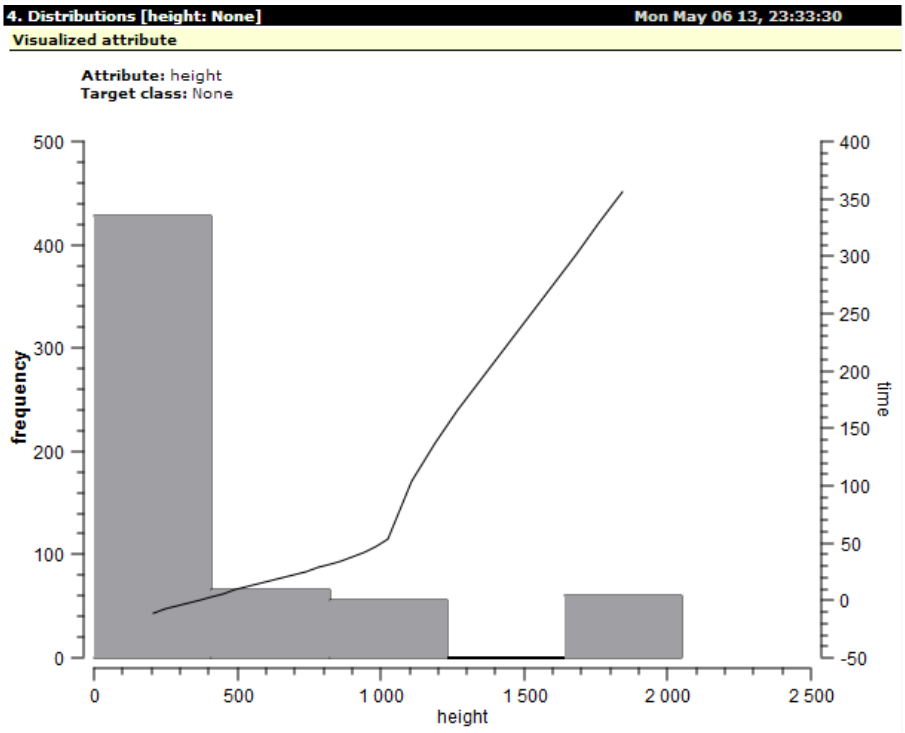


Рисунок 14.7 – Компонент `4. Distributions`. Распределение свойства `height` и его влияние на свойство `time`.

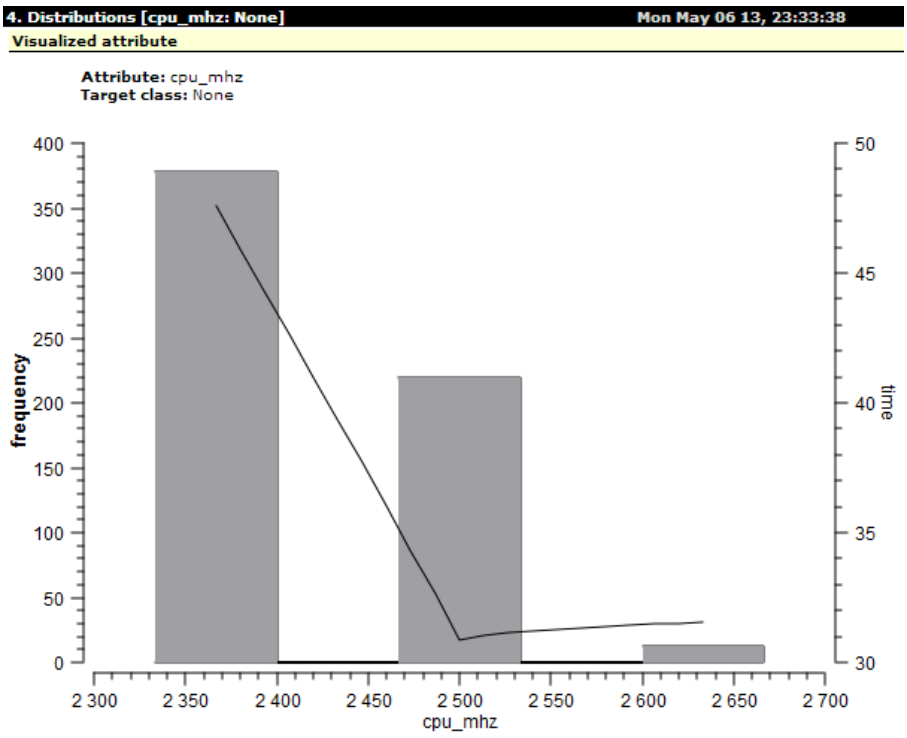


Рисунок 14.8 – Компонент `4. Distributions`. Распределение свойства `cpu_mhz` и его влияние на свойство `time`.

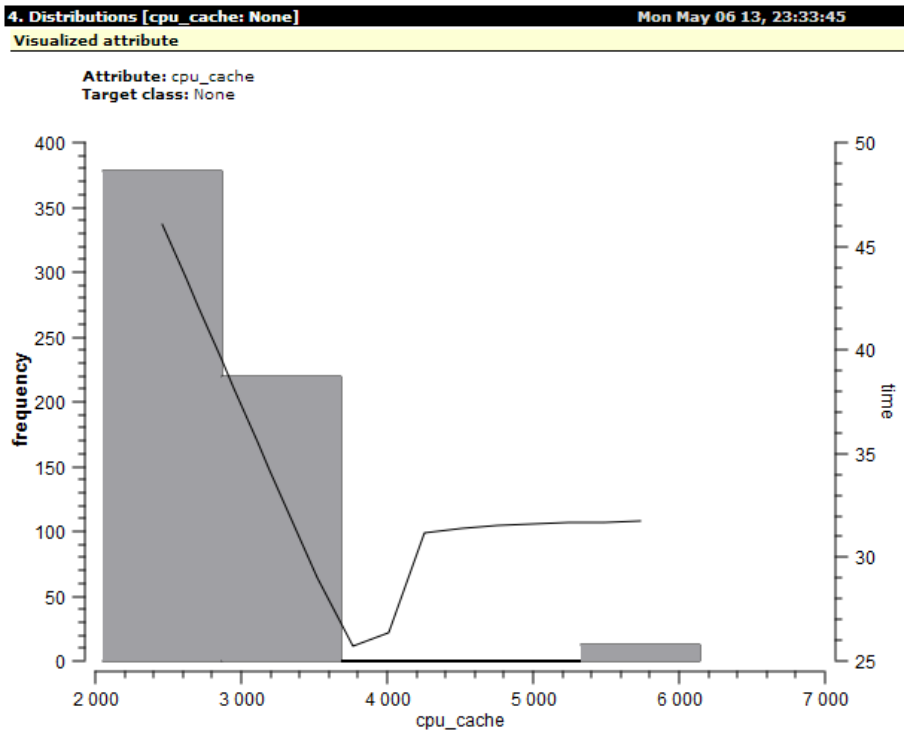


Рисунок 14.9 – Компонент `4. Distributions`. Распределение свойства `cpu_cache` и его влияние на свойство `time`.

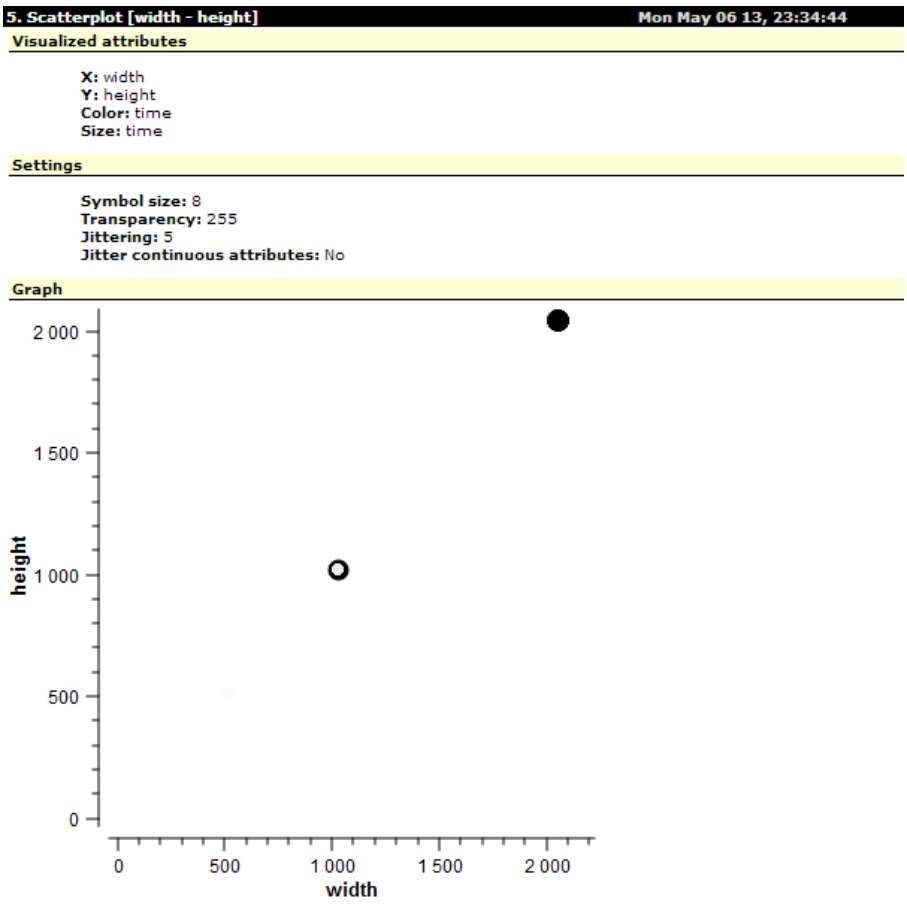


Рисунок 14.10 – Компонент `5. Scatterplot`. Зависимость свойства `time` от свойств `width` и `height`.