Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехники и комплексной автоматизации»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к научно-исследовательской работе студента

на тему

«Исследование методов визуализации алгоритмов сложных вычислительных методов»

Студент РК6-72Б (группа) _____ (подпись, дата) Ершов В.А. (ФИО)

Руководитель НИРС _____ (подпись, дата) Соколов А.П. (ФИО)

Консультант _____ (подпись, дата) Першин А.Ю. (ФИО)

Москва, 2021

МГТУ им. Н.Э.Баумана, кафедра «Системы автоматизированного проектирования» (РК-6), шаблон RPT (размещение sa2tml)

РЕФЕРАТ

Аналитический обзор литературы проводится в рамках выполнения работы по разработке web-ориентированного редактора графов. Редактор используется для построения графовых моделей используемых для описания сложных вычислительных методов. Проведен обзор статей и работ, описывающих актуальность визуального программирования и оптимизации процесса разработки вычислительных методов, описаны существующие современные программные разработки, позволяющие использовать визуальное программирование для реализации вычислительных методов.

Тип работы: научно-исследовательская работа студента .

Тема работы: *«Исследование методов визуализации алгоритмов сложных вычислительных методов»*.

Объект исследования: Применение графориентированного подхода для реализации сложных вычислительных методов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При разработке любого программного обеспечения разработчики ставят перед собой несколько очень важных задач: эффективность и гибкость дальнейшей поддержки программного обеспечения. В современной разработке существует множество паттернов проектирования, архитектур и вспомогательных систем, которые позволяют быстрее создавать программное обеспечения и в дальнейшем упрощают его поддержку для новых разработчиков. Таким образом, даже в небольших проектах, над которыми работает небольшая команда разработчиков стараются использовать системы контроля версий, юнит-тестирование. Если приложение представляет из себя API при его разработке используют специальные системы для быстрой документации API. Такие подходы считаются стандартом при разработке программного обеспечения рассчитанное на удовлетворение потребностей конечных пользователей - разработка мобильных приложений, проектирование и реализация API и прочее.

Однако при появлении первых ЭВМ в 1930-х годах [1] программирование использовалось для решения научных задач. Научное программирование сильно отличается от других видов программирования, при разработке научного программного обеспечения очень важно получить корректный и стабильный конечный продукт, а также четко разделить интерфейсную и научную часть. Из этого следует, что разработчик научного программного обеспечения должен быть экспертом в предментной области, а сам разработчик обычно является конечным пользователем [2], в то время как в индустриальном программировании разработчик зачастую не является конечным пользователем и от него не требуется быть экспертом в предметной области разработываемого приложения.

Также научное программирование отличается от индустриального программирования тем, что стандарты проектирования научного программного обеспечения вырабатываются существенно медленне или не вырабатываются вовсе, что приводит к отсутствию каких-либо системных подходов к разработке. Это приводит к сложностям при валидации и дальнейшей поддержке кода. Так, код, написанный эффективно и корректно, может оказаться бесполезным в том если поддержка кода оказывается затруднительной, это приводит к формированию и накоплению “best practices"при программировании численных методов. Вследствие чего стали появляться системы, которые позволяют минимизировать написание кода и снизить трудозатраты на его поддержку.

Обзор источников литературы

В основном существующие платформы используют визуальное программирование [3]. Одними из самых популярных и успешных разработок в этой области являются Simulink и LabView. Simulink позволяет моделировать вычислительные методы с помощью графических блок-диаграмм и может быть интегрирован со средой MATLAB. Также Simunlink позволяет автоматически генерировать код на языке на C для реализации вычислительного метода в

режиме реального времени. LabView используется для аналогичных задах - модерование технических систем и устройств [4]. Среда позволяет создавать виртуальные приборы с помощью графической блок-диаграммы, в которой каждый узел соответствует выполнению какой-либо функции. Представление программного кода в виде такой диаграмы делает его интуитивно понятным инженерам и позволяет осуществлять разработку системы более гибко и быстро. В составе LabView есть множество специализированных библиотек для моделирования систем из конкретных технических областей.

Существует множество других систем и языков программирования для реализации вычислительных методов, однако, каждый из них является узкоспециализированным и решает определенную задачу. Так, например, система визуального модерования FEniCS [5] используется для решения задач с использованием метода конечных элементов. Система имеет открытый исходный код, а также предоставляет удобный интерфейс для работы с системой на языках Python или C++. FEniCS предоставляет механизмы для работы с конечно-элементными расчетными сетками и функциями решения систем нелинейных уравнений, а также позволяет вводить математические модели в исходной интегрально-дифференциальной форме.

Отдельного упоминания стоит система TensorFlow. TensorFlow представляет из себя библиотеку с открытым исходным кодом, которая используется для машинного обучения. Аналогично LabView и Simulink, TensorFlow позволяет строить программные реализации численных методов. Стоит обратить внимание, что в основе TensorFlow лежит такое понятие как граф потока данных. В самом графе ребра - тензоры, представляют из себя многомерные массивы данных, а узлы - матетиматические операции над ними.

Применение ориентированных графов очень удобно для построения архитектур процессов обработки данных (как в автоматическом, так и в автоматизированном режимах). Вместе с тем многочисленные возникающие в инженерной практике задачи предполагают проведение повторяющихся в цикле операций. Самым очевидным примером является задача автоматизированного проектирования (АП). Эта задача предполагает, как правило, постановку и решение некоторой обратной задачи, которая в свою очередь, часто, решается путём многократного решения прямых задач (простым примером являются задачи минимизации некоторого функционала, которые предполагаю варьирование параметров объекта проектирования с последующим решением прямой задачи и сравнения результата с требуемым согласно заданному критерию оптимизации). Отметим, что прямые задачи (в различных областях) решаются одними методами, тогда как обратные - другими. Эти процессы могут быть очевидным образом отделены друг от друга за счет применения единого уровня абстракции, обеспечивающего определение интерпретируемых архитектур алгоритмом, реализующих методы решения как прямой, так и обратной задач. Очевидным способом реализации такого уровня абстракции стало использование ориентированных графов.

А.П.Соколов и А.Ю.Першин разработали графориентированный программный каркас для реализации сложных вычислительных методов, теоретические основы представлены в ра-

боте [6], а принципы применения графоориентированного подхода зафиксированы в патенте [7]. Заметим, что в отличии от TensorFlow, в представленном графориентированном программном каркасе узлы определяют фиксированные состояния общих данных, а ребра определяют функции преобразования данных. Для описания графовых моделей был разработан формат aDOT, который расширяет формат описания графов DOT [8], входящий в пакет утилит визуализации графов Graphviz. В aDOT были введены дополнительные атрибуты и определения, которые описывают функции-предикаты, функции-обработчики и функции перехода в целом. Подроробное описание формата aDOT приведено в [9].

В описанном в работе методе вводятся такие понятия как функции-обработчики и функции-предикаты, заметим, что функции-предикаты позволяют проверить, что на вход функции-обработчика будут поданные корректные данные. Подобная семантика функций-предикатов предполагает, что функции-предикаты и функции-обработчики должны разрабатываться одновременно в рамках одной функции-перехода. Такой подход существенно ускоряет процесс реализации вычислительного метода - функции-перехода могут разрабатываться параллельно несколькими независимыми разработчиками. Также, возможность параллелизации процесса разработки позволяет организовать модульное тестирование и документирование разрабатаываемого кода.

Список использованных источников

1 Timothy Williamson. History of computers: A brief timeline. 2021.

2 J. E. Hannay C. MacLeod J. S. e. a. How do scientists develop and use scientif-ic software? 2009.

3 Robert van Liere. CSE. A Modular Architecture for Computational Steering. 2015.

4 А.В. Коргин М.В. Емельянов В.А. Ермаков. Применение LabView для решения задач сбора и обработки данных измерений при разработке систем мониторинга несущих конструкций. 2013.

5 Mortensen Mikael Langtangen Hans Petter Wells Garth N. A FEniCS-Based Programming Framework for Modeling Turbulent Flow by the Reynolds-Averaged Navier-Stokes Equations. 2011.

6 Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. 2019.

7 Соколов А.П., Першин А.Ю. Патент на изобретение RU 2681408. Способ и система графо-ориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов. 2019.

8 Stephen C. North Eleftherios Koutsofios. Drawing Graphs With Dot. 1999.

9 Соколов А.П. Описание формата данных aDOT (advanced DOT) [Электронный ресурс]. Облачный сервис SA2 Systems. [Офиц. сайт]. 2020.