МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЁВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

На правах рукописи

Литвинов Владимир Геннадьевич

РАЗРАБОТКА И ПРИМЕНЕНИЕ ТИПОВЫХ РЕШЕНИЙ ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ

05.13.18 — Математическое моделирование, численные методы и комплексы программ

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
Востокин Сергей Владимирович
доктор технических наук, доцент

Самара — 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **5**

1 МОДЕЛИРОВАНИЕ ТИПОВЫХ РЕШЕНИЙ ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ ЧИСЛЕННЫХ АЛГОРИТМОВ **11**

1.1 Метод моделирования параллельных вычислений 11

1.1.1 Обзор моделей параллельных и распределенных вычислений 11

1.1.2 Применение типовых решений для организации параллельных вычислений 15

1.1.3 Автоматизированное распараллеливание 17

1.1.4 Визуализация параллельных вычислений 18

1.2 Метод формального представления типовых решений 20

1.2.1 Описание дискретных систем с использованием темпоральной логики 20

1.2.2 Метод Лампорта для спецификации дискретных систем 24

1.2.3 Построение модели типовых решений на основе метода Лампорта 28

1.2.4 Дискретно-событийное имитационное моделирование типовых решений 30

1.3 Построение визуальных моделей типовых решений 33

1.3.1 Определение графической нотации на примере алгоритма «разветвление-слияние» 33

1.3.2 Графическая модель «метод переменных направлений» 38

1.3.3 Графическая модель «конвейер» 40

1.3.4 Графическая модель «портфель задач» 41

1.4 Выводы 44

2 СИНТЕЗ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ОБУЧЕНИЯ ГИПЕРРАДИАЛЬНОЙ НЕЙРОННОЙ СЕТИ НА ОСНОВЕ ТИПОВОГО РЕШЕНИЯ «ПОРТФЕЛЬ ЗАДАЧ» ДЛЯ РЕШЕНИЯ ЗАДАЧ ПРОГНОЗИРОВАНИЯ ВРЕМЕННЫХ РЯДОВ **45**

2.1 Решение задачи прогнозирования временных рядов нейронной сетью 45

2.1.1 Понятие нейронной сети 45
2.1.2 Постановка задачи прогноза временного ряда нейронной сетью 47
2.1.3 Нейронная сеть на основе радиально-базисных функций 49
2.1.4 Гиперрадиальная нейронная сеть 53
2.2 Разработка алгоритма обучения гиперрадиальной нейронной сети 56
2.2.1 Принцип обучения 56
2.2.2 Обучение с подбором оптимальной архитектуры 57
2.2.3 Синтез параллельного алгоритма обучения на основе типового решения «портфель задач» 58
2.2.4 Методика синтеза параллельных алгоритмов с использованием моделей типовых решений 60
2.3 Экспериментальное исследование алгоритма обучения 63
2.4 Выводы 67

3 ПОСТРОЕНИЕ ПРОГРАММНОГО КОМПЛЕКСА ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ НА ОСНОВЕ ТИПОВЫХ РЕШЕНИЙ И ОПИСАНИЕ ЕГО ПРИМЕНЕНИЙ 69
3.1 Обзор программного обеспечения параллельных вычислений . 69
3.1.1 Прикладные интерфейсы операционных систем 69
3.1.2 Стандартные библиотеки и библиотеки времени исполнения языков программирования 72
3.1.3 Средства управления параллельными вычислениями . . 77
3.2 Архитектура программного комплекса 80
3.2.1 Принципы реализации программного комплекса 80
3.2.2 Описание библиотеки типовых решений 81
3.2.3 Описание реализации программного комплекса 85
3.3 Экспериментальная оценка эффективности распараллеливания задач математического моделирования с использованием типовых решений 88
3.3.1 Оценка сложности реализации численных моделей по объему ручного и автоматического кодирования 88

3.3.2 Результаты применения типового решения «метод переменных направлений» в задаче исследования нелинейных динамических процессов 89
3.3.3 Результаты применения типового решения «конвейер» в задаче о распространении плоской световой волны . . . 91
3.3.4 Результаты применения типового решения «портфель задач» в задаче моделирования фазовых траекторий динамических систем 94
3.4 Выводы 96

ЗАКЛЮЧЕНИЕ **98**

СПИСОК ЛИТЕРАТУРЫ **99**

ПРИЛОЖЕНИЕ А. Акты внедрения **113**

ПРИЛОЖЕНИЕ Б. Свидетельство о регистрации программы **117**

ПРИЛОЖЕНИЕ В. Представление каналов и процессов в виде сети Петри **119**

ВВЕДЕНИЕ

Роль высокопроизводительной вычислительной техники в области математического моделирования чрезвычайно важна. Её применение позволяет сократить время решения задач моделирования, повысить размерность и точность решаемых задач, проводить исследования, которые принципиально невозможны на последовательной *ЭВМ*.

Переход от последовательного численного метода к параллельному и, далее, к вычислительным экспериментам на высокопроизводительных *ЭВМ* является самостоятельной научной и инженерной проблемой. Она обусловлена тем, что к сложности исходного численного метода добавляется сложность организации коммуникации, синхронизации, распределения вычислений по процессорам. Данная проблема выделена академиком Г.И. Марчуком как фундаментальное научное направление, называемое «проблемой отображения» [1].

Одним из подходов к решению «проблемы отображения» является повторное использование имеющихся типовых решений (шаблонов или паттернов) для распараллеливания численных методов разного назначения, который впервые описан в работе Э. Гамма, Р. Хелма, Р. Джонсона и Дж. Влиссидеса [2] для объектно-ориентированного проектирования. В области типовых решений для параллельных научных вычислений известны работы отечественных и зарубежных ученых: М.И. Коула [3], С. Макдональда [4–6], П.К. Берзигиярова [7], В.Э. Малышкина [8], С.В. Востокина [9, 10] и других. В них представлено около двух десятков различных типовых решений в области параллельного программирования.

Однако актуальной задачей исследований остаётся разработка общих подходов для описания разных типовых решений. В диссертации предлагается и исследуется общая модель вычислительного процесса, позволяющая строить специализированные модели типовых решений. Используется графическая форма представления модели типового решения, удобная для конструирования параллельных численных алгоритмов и программ.

В качестве важного приложения методов синтеза параллельных алгоритмов на базе типовых решений в диссертации исследованы методы построения интеллектуальных систем на базе нейронных сетей, предложенных С. Осовским, С. Хайкиным, А.Н. Горбанём, с целью разработки эффективных парал-

лельных программ обучения гиперрадиальных нейронных сетей для решения задач прогнозирования.

Процесс разработки параллельных программ в настоящее время отличается относительно низкой автоматизацией. Поэтому большое практическое значение имеют снижение стоимости и сокращение сроков проведения вычислительных экспериментов на высокопроизводительных *ЭВМ*. В этой связи в диссертации рассматривается проблема построения программного комплекса на основе библиотеки типовых решений и актуальное направление интеграции программного комплекса в инфраструктуру организации научных вычислений на основе *web*-сервиса и в современные процессы разработки.

Цель и задачи исследований. Целью исследований является разработка методики построения эффективных параллельных алгоритмов и комплексов программ на основе математической модели вычислительного процесса; применение данного подхода при постановке численных экспериментов на высокопроизводительных системах в задаче обучения гиперрадиальной нейронной сети.

Задачи исследований, соответствующие поставленной цели диссертационной работы.

а) Анализ актуальных проблем разработки программных комплексов высокопроизводительных научных вычислений.
б) Разработка общей математической модели вычислительного процесса для описания различных типовых решений в области высокопроизводительных научных вычислений.
в) Разработка графической модели алгоритма для типового решения «портфель задач» на основе математической модели вычислительного процесса и графической нотации *Templet*.
г) Построение методики синтеза параллельного алгоритма на примере алгоритма обучения гиперрадиальной нейронной сети с использованием модели типового решения «портфель задач».
д) Разработка программного комплекса численного моделирования на основе типового решения «портфель задач» для использования на высокопроизводительных вычислительных системах.
е) Проведение вычислительных экспериментов для анализа эффективности распараллеливания и сложности разработки с использованием про-

граммного комплекса на основе типового решения «портфель задач».

Научная новизна диссертационной работы. В диссертации получены следующие новые научные результаты.

а) Построена новая математическая модель вычислительного процесса для описания различных типовых решений. В отличие от традиционных моделей типовых решений, она строго формализована с использованием аппарата темпоральной логики Лампорта, является основой для кроссплатформенной эффективной реализации параллельных алгоритмов.

б) Разработана графическая модель алгоритма типового решения «портфель задач». В отличие от известных методов описания принципа организации вычислений в типовом решении, новая графическая модель алгоритма позволяет детально передать логику его функционирования без привязки к языку реализации.

в) Предложена и описана на примере построения параллельного алгоритма обучения нейронной сети гиперрадиального типа новая методика синтеза параллельного алгоритма. Методика основана на нисходящей декомпозиции исходного последовательного алгоритма на основе графической модели параллельного алгоритма. В ней, в отличие от традиционных, не используются примитивы синхронизации и коммуникации языка и среды исполнения.

г) Разработан программный комплекс и приведены результаты его экспериментального исследования на примере решения задачи обучения гиперрадиальной нейронной сети. В отличие о известных программных решений, использование общей математической модели типового решения позволило добиться кроссплатформенности программы с сохранением высокой эффективности её выполнения в разных средах: *Windows*, *POSIX*, *MPI*.

Теоретическая и практическая значимость работы. Теоретическая значимость работы состоит в том, что в ней предложена методика, расширяющая известные методики, основанные на типовых решениях. Новыми элементами методики являются: математическая модель вычислительного процесса и графическое представление параллельных алгоритмов в типовых решениях. Математическая модель является основой кроссплатформенной реализации параллельного алгоритма. Использование графической модели в типовых

решениях обобщает их описание на произвольный алгоритмический язык реализации и программную платформу и задаёт точную декомпозицию параллельного алгоритма на составляющие последовательные процедуры и обрабатываемые ими типы данных.

Практическая значимость работы состоит в создании программного комплекса, позволяющего автоматизировать постановку вычислительного эксперимента на высокопроизводительной *ЭВМ*. Его использование сокращает время проведения вычислительного эксперимента, исключает возможные ошибки в параллельной программе, расширяет круг специалистов, применяющих высокопроизводительную вычислительную технику для решения задач математического моделирования. Программный комплекс размещён в сети интернет как компонент *web*-сервиса научных вычислений *Templet Web* (`http://templet.ssau.ru`).

Прикладное значение также имеют реализованные в работе параллельный алгоритм и программа обучения и оптимизации структуры гиперрадиальной нейронной сети. Их применение позволяет снизить вычислительную сложность на стадии функционирования нейронной сети. В модельных экспериментах с нейронной сетью продемонстрирована эффективность предложенной в работе методики разработки параллельных алгоритмов.

Соответствие паспорту специальности. Работа соответствует следующим пунктам паспорта заявленной научной специальности: (п. 1) разработка новых математических методов моделирования объектов и явлений; (п. 3) разработка, обоснование и тестирование эффективных вычислительных методов с применением современных компьютерных технологий; (п. 4) реализация эффективных численных методов и алгоритмов в виде комплексов проблемно-ориентированных программ для проведения вычислительного эксперимента.

Реализация результатов работы. Диссертационная работа выполнена в рамках работ по проекту «Разработка комплекса технологий использования ресурсов суперкомпьютера «Сергей Королёв» в целях развития инновационной и научно-образовательной среды университета» (мероприятия 3.3 Программы развития национального исследовательского университета).

Результаты диссертационной работы внедрены в научной деятельности факультета летательных аппаратов *СГАУ* в 2013 году и применяются при выполнении *НИР* для анализа многомерных динамических систем и процессов с

использованием суперкомпьютера «Сергей Королёв», о чем имеется акт внедрения.

Методы исследований. В работе использованы темпоральная логика Лампорта, графические методы описания алгоритмов, имитационное моделирование, теория искусственных нейронных сетей.

Апробация работы. Основные результаты работы были представлены на следующих всероссийских и международных конференциях: Всероссийской молодежной научной конференции с международным участием «X Королёвские чтения» (Самара, 2009) [11]; Международной конференции с элементами научной школы для молодежи «Перспективные информационные технологии для авиации и космоса (ПИТ-2010)» (Самара, 2010) [12]; Всероссийской молодежной научной конференции с международным участием «XI Королёвские чтения» (Самара, 2011) [13]; Шестой Международной конференции Института проблем управления имени В.А. Трапезникова РАН «Параллельные вычисления и задачи управления» (Москва, 2012) [14]; Научно-технической конференции с международным участием «Перспективные информационные технологии в научных исследованиях, проектировании и обучении (ПИТ-2012)» (Самара, 2012) [15]; Научно-технической конференции с международным участием «Перспективные информационные технологии (ПИТ 2013)» (Самара, 2013) [16].

Авторский вклад. Все результаты, изложенные в диссертации, получены автором лично. В работах, выполненных совместно, автору принадлежат части, относящиеся к типовому решению «портфель задач» и алгоритм обучения нейронной сети. Автор лично осуществлял проведение численных экспериментов, обработку, анализ и интерпретацию полученных результатов.

Достоверность результатов работы. Достоверность полученных в диссертации решений подтверждена корректными математическими выкладками, результатами вычислительных экспериментов на высокопроизводительной технике, а также соответствием результатов математического моделирования реальным процессам.

Основные положения, выносимые на защиту.

а) Математическая модель вычислительного процесса для описания различных типовых решений на основе темпоральной логики Лампорта.

б) Графическая модель алгоритма типового решения «портфель задач».

в) Методика нисходящей декомпозиции последовательного алгоритма на основе графической модели алгоритма типового решения «портфель задач»; параллельный алгоритм обучения нейронной сети гиперрадиального типа, построенный на её основе.

г) Программный комплекс численного моделирования, реализующий типовое решение «портфель задач»; результаты его функционирования на примере задачи обучения гиперрадиальной нейронной сети.

Публикации по теме диссертации. По теме диссертации опубликовано 12 работ, в том числе 3 статьи в журналах, рекомендованных ВАК РФ [17–19], одно свидетельство о регистрации программы для *ЭВМ* [20], 6 работ в материалах и трудах международных и всероссийских конференций [11, 12, 14–16, 21], 2 тезиса доклада [13, 22].

ГЛАВА 1. МОДЕЛИРОВАНИЕ ТИПОВЫХ РЕШЕНИЙ ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ ЧИСЛЕННЫХ АЛГОРИТМОВ

В первой части первой главы приводится обзор известных приёмов моделирования и организации параллельных вычислений [23], на основе которых строится метод моделирования параллельных вычислений [9, 10, 24, 25]. Этими приёмами являются: модель процессов, типовые алгоритмические структуры или паттерны, методы автоматизированного распараллеливания и визуализации графовых моделей параллельных вычислений.

Во второй части с использованием темпоральной логики Лампорта [26, 27] строится модель, на базе которой выполняется точное описание типовых решений. Показано, как на основе предложенной модели построить монитор моделирования, позволяющий воспроизвести исполнение параллельного процесса на последовательном компьютере. Рассмотрен простейший вариант его распараллеливания.

В третьей части главы с использованием графической нотации *Templet* [9, 10, 28, 29] и построенной модели даётся описание часто используемых в практике численного моделирования типовых решений: «метод переменных направлений», «конвейер» и «портфель задач».

Результаты главы опубликованы в работах [14, 16, 19].

1.1 Метод моделирования параллельных вычислений

1.1.1 Обзор моделей параллельных и распределенных вычислений

Формальные методы в области параллельных и распределенных вычислений. Связь между свойствами алгоритма и его реализацией не очевидна, особенно в случае распределенных и параллельных вычислений. Вследствие чего актуально построение формальных методов, цель которых состоит в установлении этой связи. Важной вспомогательной задачей при этом является разработка приемов описания свойств алгоритмов и моделей вычислений. В случае последовательных вычислений свойства алгоритмов — это функции, связывающие корректные входные и выходные данные. Для параллельных вычислений такие функции не являются адекватным представлением,

так как дополнительно требуется описывать взаимодействие между процессом и реализующим его окружением. Актуальность разработки практических (связанных с конкретной программно-аппаратной архитектурой) и теоретических моделей параллелизма вызвана отсутствием единой фундаментальной концепции, аналогичной имеющейся в последовательных вычислениях. Таким образом, в зависимости от подхода к решению задачи о свойствах алгоритмов, а также способов ее описания, можно классифицировать формальные методы в области параллельных и распределенных вычислений.

В зависимости от того, строится алгоритм по спецификации или исследуется готовый алгоритм, выделяют методы синтеза и методы доказательства корректности.

Наиболее распространены следующие методы синтеза. Для получения эквивалентных по результату вычислений параллельных алгоритмов применяется распараллеливание последовательных алгоритмов [30]. Методы распараллеливания разделяют на статические и динамические, для ациклических и циклических участков кода [31] и для отдельных операторов программы [32, 33]. Для синтеза параллельных алгоритмов исследованы методы, использующие непроцедурные спецификации [34]. Разработаны методы, основанные на последовательном построении кода алгоритма совместно с доказательством корректности [35], а также на эквивалентных алгебраических преобразованиях [36]. Формальные методы синтеза параллельных алгоритмов являются, безусловно, привлекательными. Однако они ориентированы на узкий класс алгоритмов и не являются универсальными.

Методы анализа можно разделить на точные и имитационные. Точный метод предполагает доказательство того, что известный алгоритм обладает некоторым свойством. При этом различают методы, использующие верификацию, и методы, использующие спецификацию. Спецификация предполагает наличие высокоуровневой, независящей от реализации нотации, в терминах которой описывают как алгоритм, так и его свойства [27]. Верификация основана на рассуждениях в терминах алгоритма. Распространенным методом верификации является метод утверждений [37]. Методы спецификаций используют разновидности темпоральных логик [38–40]. Для исследования свойств алгоритма или его модели используют имитационное моделирование. Такие методы часто применяются при моделировании распределенных систем. Спе-

циальным приложением является изучение стратегий планирования ресурсов. Наиболее распространены дискретно-событийные модели, управляемые событиями. Также используются методы построения и анализа трасс алгоритма по его спецификации [41].

Все известные формальные методы используют понятие локальности, предполагающее, что действия алгоритмов изменяют лишь часть общего состояния системы. Изменение состояния может описываться как последовательность состояний или последовательность действий, приводящих к смене состояний, называемых событиями. При этом вычислительный процесс представляется в виде множества цепочек из событий (состояний) или деревом событий [36].

Разработано большое число разнообразных методов параллельного программирования, различающихся способом описания действий, изменяющих состояние системы, и их синхронизацией.

Представление алгоритма может использовать или не использовать понятие потока управления.

Если используются потоки управления, то простейшим способом синхронизации является последовательно-параллельный метод [30], в котором отсутствует обмен данными между параллельными потоками, но определяются операции ветвления и объединения ветвей. Более сложные методы предусматривают взаимодействие ветвей, которое может организовываться через общую память (методы, основанные на атомарных операциях, процедурные методы) или через сообщения (синхронные коммуникации и асинхронные коммуникации).

Если отсутствует явное определение потока управления, то такие методы программирования называются асинхронными [42]. По способу управления вычислениями асинхронные методы подразделяются на методы с событийным управлением, где условием активации действия являются специально формируемые события; потоковые, где условие активации — это готовность входных данных функции; а также динамические [43], где условия готовности представляют собой булевы функции от части состояния системы, хранящегося в разделяемых переменных.

Современные практические способы представления параллелизма в алгоритмах и программах используют понятие процесса, поэтому соответству-

ющая группа моделей самая большая.

Классификация типовых моделей процессов. Основными моделями распределенных вычислений являются модели процессов. В таких моделях вычислительная активность представляется как параллельное во времени исполнение последовательных процессов. Другие модели, например, сети Петри, обычно не изучаются как распределенные [44], хотя они могут использоваться для моделирования пространственно распределенных систем [45].

Наиболее очевидными моделями распределенных процессов являются модели обмена сообщениями [46]. Процесс посылает сообщение, добавляя его в очередь сообщений. Другой процесс получает сообщение, извлекая его из очереди. Модели могут различаться такими деталями, как длина очереди сообщений, длительность задержки между помещением и извлечением сообщения.

Модели обмена сообщениями, используемые для представления распределенных систем, можно классифицировать по четырем признакам: сетевая топология, синхронность, виды отказов, способ буферизации сообщений.

Другие модели. В ранних моделях параллелизма процессы взаимодействовали через глобальные разделяемые переменные. Это переменные в программе, с которыми любые процессы могут выполнять операции чтения и записи. Изначально к разделяемым переменным разрешался доступ только операциям вычисления выражения и присваивания. Более поздние вариации включали рассмотрение синхронизирующих примитивов, таких как семафоры и мониторы, управляющих доступом к разделяемым переменным [47, 48]. Модели с глобальными разделяемыми переменными являются естественным представлением мультипроцессорной обработки на одном компьютере с одним или несколькими процессорами, присоединенными к центральной разделяемой памяти.

Наиболее естественный и эффективный способ реализации глобальных разделяемых переменных посредством обмена сообщениями заключается в помещении каждой разделяемой переменной в выделенный процесс. Этот процесс получает локальный доступ к переменной, а другому процессу требуется послать два сообщения для чтения или записи этой переменной. Чтение требует запроса и ответа, в котором передается значение. Запись требует посылки нового значения и получения подтверждения, что операция заверши-

лась.

Более узкий класс моделей разрешает межпроцессорное взаимодействие только через локальные разделяемые переменные. Это разделяемые переменные, для которых определено понятие «владение процессом». Локальная разделяемая переменная может читаться несколькими процессами, но в нее может писать только процесс, который ей владеет. Чтение переменной, которой владеет отказавший процесс, предполагает возврат некоторого предопределенного значения.

Еще одной известной моделью вычислений является модель [49], введенная Хоаром в языке *CSP* (взаимодействующие последовательные процессы). В языке *CSP* процесс i посылает значение v процессу j, выполняя команду вывода $j!v$. Процесс j получает это значение и присваивает его переменной x, выполняя команду ввода $i?x$. В отличие от случая обычной посылки сообщения команды ввода и вывода выполняются одновременно. Исполнение $j!v$ операции откладывается до тех пор, пока процесс i не будет готов исполнить $i?x$ операцию и наоборот. То есть, в *CSP* операция взаимодействия ждет, пока соответствующая ей операция не сможет быть выполнена другим процессом.

Похожим на синхронное взаимодействие является удаленный вызов процедуры и рандеву. Удаленный вызов процедуры (*RPC*) происходит почти так же как обычный вызов за исключением того, что процедура исполняется в другом процессе. Он может быть реализован посредством двух сообщений для передачи аргументов вызова и возврата результата. Остановка или потеря значений могут быть обработаны, если процедура возвращает признак ошибки или возбуждает исключение, когда не получено ответа на первое сообщение. Удаленный вызов процедуры в настоящее время является наиболее используемой языковой конструкцией для реализации распределенных систем без явных операций обмена сообщениями.

1.1.2 Применение типовых решений для организации параллельных вычислений

Применение библиотек основано на повторном использовании кода часто встречающихся задач из определенной области, например, линейной ал-

гебры. Другой подход — это обнаружение общих свойств и методов кодирования параллельных алгоритмов из разных областей. Оказывается, что разные алгоритмы включают в себя относительно небольшое число базовых конструкций, повторяющихся от программы к программе. Обобщив данные конструкции можно в дальнейшем использовать их как строительные блоки для автоматизации кодирования новых параллельных алгоритмов.

Выделяют три парадигмы программирования, в которых развивается описанный подход. Впервые исследования по обобщению типовых алгоритмических структур были предприняты в функциональном программировании. С использованием математического аппарата функций высших порядков были формализованы такие известные алгоритмические структуры как конвейер, редукция, «разделяй и властвуй» [3]. Также были исследованы методы иерархической и горизонтальной композиции элементарных структур с целью синтеза сложных программ. Системами программирования, использующими алгоритмические структуры и функциональную парадигму, являются *SPF* [50], *SkelML* [51], *SKIL* [52], *SKIPPER* [53].

В области процедурного программирования данный подход развивается путем определения средств задания алгоритмических шаблонов. Примером являются системы *P3L* [54] и *SkIE* [55]. Интерес представляют исследования в области эквивалентных преобразований для языков схем. При этом, как показано в работе [7], удается оптимизировать порождаемый код, сократив накладные расходы на обмен данными.

В области объектно-ориентированного направления данный подход носит название паттернов проектирования [2]. Параллельные объектно-ориентированные паттерны реализуются в специальных системах программирования, например, *FrameWorks* [56], *DPnDP* [57], *Enterprise* [4], CO_2P_3S [6]. Объектно-ориентированные языки программирования, в частности *C++*, имеют средства описания параметризованных классов. С использованием этих средств можно строить специальные библиотеки, называемые каркасами приложений (*framework*), реализующие отдельные паттерны или языки паттернов для построения параллельных и распределенных приложений.

1.1.3 Автоматизированное распараллеливание

Автоматизированное распараллеливание заключается в выявлении неявного параллелизма в алгоритме, записанном некоторым способом. Обычно в системах распараллеливания используют три способа записи алгоритма: функциональный, процедурный и процедурный с распараллеливающими директивами.

К функциональным языкам, предназначенным для автоматического распараллеливания, относится язык *Sisal* [58]. Компилятор языка определяет все зависимости, распределяет работу по процессорам, вставляет необходимые пересылки и синхронизации. Наиболее проработанным российским проектом данного типа является проект *Т-система* [59]. В рамках проекта предложены реализации средств динамического распараллеливания для разных аппаратных платформ, в том числе для кластерных и грид-систем. Разработан самостоятельный язык и библиотека шаблонных классов на языке *C++*, реализующая вычислительную модель *Т-системы*.

Для языков *Fortran* и *C* разработано большое число распараллеливающих компиляторов и анализаторов кода, работающих в автоматическом или полуавтоматическом режиме. Система *BERT 77* [60] выполняет распараллеливание и генерирует код с использованием модели передачи сообщений *MPI* или *PVM*. Примером среды для анализа программ на языке *Fortran* с возможностью генерации исходных текстов с директивами *OpenMP*, а также *MPI* и *PVM*, является система *PIPS Workbench*. Среди отечественных разработок данного класса можно отметить систему *V-Ray* [61].

Другим распространенным подходом к распараллеливанию, имеющим стандартные реализации, например, рассмотренный выше стандарт *OpenMP*, является использование распараллеливающих директив, встроенных в текст последовательной программы. Примером служит расширение языка *Fortran 90* — язык *HPF* (*High Performance Fortran*) [62]. Он включает средства для распределения данных по процессорам. При этом необходимые коммуникации и синхронизации реализуются компилятором автоматически. Часть расширений выполнена в виде функций и операторов языка, а часть как директивы компилятору в форме комментариев. К российским разработкам аналогичного

назначения для языков *Fortran* и *C* относится система *DVM* [63]. В нее, кроме препроцессора, также входит библиотека поддержки времени исполнения, отладчик, предсказатель выполнения, анализатор производительности.

Можно отметить, что в некотором смысле распараллеливание аналогично автоматическому выявлению типовых параллельных структур в тексте последовательного алгоритма. Поэтому для эффективного исполнения параллельной программы исходный последовательный алгоритм нужно писать не произвольным образом, а так, чтобы компилятору удалось распознать конструкции с неявным параллелизмом. Несмотря на этот недостаток, метод существенно снижает сложность программирования и повышает скорость разработки, так как кодирование и отладка выполняются для последовательной программы.

1.1.4 Визуализация параллельных вычислений

Ввиду того, что код на текстовом языке представляет собой последовательность из синтаксических элементов, программа является адекватным представлением абстракции алгоритма, также определяемого как последовательность действий. При этом, в случае параллельного программирования нельзя однозначно упорядочить все действия. Поэтому текстовое представление в виде последовательности синтаксических элементов скрывает от разработчика существенные детали поведения параллельной программы, влияющие на корректность и эффективность ее исполнения. Подход, используемый в визуальных средствах проектирования параллельных программ, основан на графических способах преставления как структуры программы, так и ее поведения, позволяющих описывать не только бинарные, но и произвольные отношения между элементами.

Наиболее популярными способами визуализации параллелизма являются: диаграммы потоков управления (*ДПУ*), диаграммы потоков данных (*ДПД*), системы перерисовки графов, методы описания взаимодействия объектов.

В языках использующих, *ДПУ* диаграммы, программы представляются ориентированными графами, в которых дуги обозначают передачу управления, а вершины обозначают либо последовательные вычисления, либо управ-

ляющие конструкции. Примером применения данного способа служат: язык *HeNCE* [64], язык *Phred* [65], *D2R* [66], технология графо-символического программирования — *ГСП* [67], технология *ГРАФКОНТ* [68].

В языках, основанных на *ДПД* диаграммах, программы имеют вид ориентированных графов, в которых дуги обозначают связь по данным. Описание вершины обычно состоит из определения соответствия формальных параметров процедуры фактическим параметрам; условий активации последовательной процедуры вершины; самой последовательной процедуры; описания того, куда передавать полученные значения; описания локальных данных вершины. Такие языки позволяют специфицировать интерфейсы и организовывать, тем самым, повторное использование компонентов. Типичными языками, использующими *ДПД* графы, являются *CODE* [69], *Paralex* [70], *Neptune* [71].

Графы с динамической структурой задаются при помощи правил репликации вершин. Системы перерисовки графов (граф-грамматики) отражают произвольные динамические изменения в структуре программы, не описываемые простыми правилами репликации. В языках этого типа вершины могут порождаться и удаляться, а связи между вершинами устанавливаться и разрушаться во время исполнения программы. Эти изменения задаются в форме правил перерисовки, которые состоят из предикатов, определенных над некоторым набором дуг и вершин; операций перерисовки, выполняющихся при срабатывании условий. Примером использования граф-грамматики в языках параллельного программирования являются системы *Delta* [72] и *ParaGraph* [73].

Большую группу методов визуального представления параллелизма составляют методы визуализации объектных моделей. Диаграммы взаимодействия объектов иллюстрируют взаимосвязи параллельно протекающих процессов во времени. Описано несколько диаграмм такого вида, например, диаграммы трассировки событий Румбаха [74] или диаграммы взаимодействий Джекобсона [75]. Часто используются диаграммы, имеющие вид графов, вершины которых обозначают активные объекты, а дуги — отношения клиент-сервер между объектами. Эти диаграммы применяются в комбинации с диаграммами состояний и переходов Херлии [76], описывающих внутреннее поведение объектов.

Графические методы оказываются удобными на всех стадиях разработки

программы: проектировании; имитационном моделировании вычислительного процесса; кодировании и генерации кода; отладки логики и отладки производительности; отображения программы на процессоры. Графические средства автоматизации обычно реализуются как надстройки над традиционными текстовыми средствами программирования. Многие методы стандартизированы в нотациях *UML* и *IDEF3* и широко применяются в промышленном программировании.

1.2 Метод формального представления типовых решений

1.2.1 Описание дискретных систем с использованием темпоральной логики

Для формализации представленного описания вычислительной модели воспользуемся специальной модальной логикой — темпоральной логикой *TLA* (*temporal logic of actions*), предложенной Лампортом [26, 27]. Удобство *TLA* при формализации модели заключается в использовании переменных для представления состояния вычислений (как в традиционных языках программирования) и в возможности однозначного сопоставления модели в терминах *TLA* со структурным представлением в виде графа.

В качестве наблюдаемого поведения дискретной системы будем рассматривать множество ее историй [9, 10]. Пусть вычислительный процесс представлен множеством историй σ: бесконечных последовательностей дискретных состояний s, причиной изменения которых во времени являются действия A

$$s_0 \xrightarrow{A_1} s_1 \xrightarrow{A_2} s_2 \xrightarrow{A_3} s_3 \xrightarrow{A_4} \ldots . \tag{1.1}$$

Моменты наблюдения состояний связаны с отсчетами дискретного времени. Возникает проблема: следует ли рассматривать конечные или бесконечные цепочки событий в каждой отдельной истории. Возможно, требуется включать в описание как конечные, так и бесконечные цепочки. Разные методы спецификации основаны на разных допущениях. В силу того, что нашей целью является математическое описание, мы ограничиваемся рассмотрением

только бесконечных историй по следующим соображениям. Смысл математической нотации подразумевает описание состояния универсума вообще, а не отдельно рассматриваемой системы. Например, некоторое уравнение — это утверждение о значениях всех переменных, а не только тех, которые входят в него в явном виде. Подразумевается, что остальные переменные могут иметь произвольные значения. Таким образом, описание системы в математическом смысле — это описание всего универсума, включающего систему. Остановка вычислений в рассматриваемой системе не влечет остановку глобального дискретного времени. Соответственно, если описываемая система состоит из нескольких подсистем, то описание их совокупности требует, чтобы ход времени в модели не зависел от возможности продолжить вычисления в каждой из подсистем. Кроме того, остановка вычислений очевидным образом представима как отсутствие изменений состояния, начиная с некоторого отсчета дискретного времени. Таким образом, непрерывность времени требуется для корректного представления композиции подсистем. Поэтому, не уменьшая выразительной силы описания важных свойств реальных дискретных систем, можно ограничиться рассмотрением только бесконечных историй, порожденных последовательностью неделимых (атомарных) действий из некоторого конечного множества [26].

С учетом вышесказанного, модель системы в темпоральной логике — это формула, смысл которой определен в терминах историй бесконечной длины (1.1). Для описания системы нам требуется построить формулу, которая бы однозначно определяла заданное множество историй. Это требует определения интерпретации формулы в терминах историй σ.

Обозначим интерпретацию формулы знаком $[[\cdot]]$. Интерпретация формулы F — это функция, которая ставит в соответствие формуле F истинное или ложное высказывание об историях вычислительного процесса (символ $\triangleq$ означает равен по определению):

$$\langle s_0, s_1, s_2, \ldots \rangle \, [[F]] \triangleq \sigma \, [[F]] : \mathbf{St}^\infty \to \{\text{И}, \text{Л}\} \, . \tag{1.2}$$

Множество историй, для которых интерпретация формулы (1.2) истинна, однозначно определяет дискретную систему в темпоральной логике. Поэтому формула F является математическим описанием дискретной системы, эквива-

лентным множеству историй (1.1).

Определим функцию интерпретации для нашей темпоральной логики. Семантическим понятием, на котором основана функция интерпретации является состояние. В математическом смысле состояние тоже представляет собой функцию, отображающую множество переменных $\mathbf{Var}$ на множество значений переменных $\mathbf{Val}$. То есть, связь между состоянием, множеством переменных и множеством значений переменных модели передается в виде

$$\mathbf{St}\colon \mathbf{Var} \to \mathbf{Val}. \tag{1.3}$$

Таким образом, смысл переменных в используемой темпоральной логике состоит в представлении значений в некотором состоянии вычислений (1.3). Исходя из этого, можно определить смысл произвольных выражений. Математические выражения, построенные из переменных, знаков математических операций, знаков отношений, логических связок, кванторов и т.п. называются функциями состояния и интерпретируются согласно выражению

$$s\left[\left[f\right]\right] \triangleq f\left(\forall \text{`}v\text{'} : s\left[\left[v\right]\right]/v\right). \tag{1.4}$$

Выражение (1.4) означает, что, если вместо каждого вхождения переменной в функцию состояния подставить ее значение, то получится интерпретация формулы для состояния s. Специальным случаем функции состояния является предикат состояния. Предикат состояния — функция состояния, принимающая значение истина или ложь.

Для удобства описания систем следует различать два типа переменных. Переменные, о которых говорилось выше, являются темпоральными переменными. Их значения могут различаться для различных состояний, то есть меняться с течением времени. Имеется специальные переменные, значения которых предполагаются неизвестными, но одинаковыми для любого состояния. Такие переменные называются темпоральными константами.

Действие — это выражение, аналогичное функции состояния по своей структуре, которое дополнительно включает переменные с символом апострофа (y' — обозначает новое значение переменной y), принимает значения истина или ложь и интерпретируется в виде

$$s\,[[A]]\,t \triangleq A\,(\forall\text{`}v\text{'} : s\,[[v]]\,/v, t\,[[v]]\,/v')\,. \tag{1.5}$$

Действие по своему смыслу является причиной (событием), приводящей к смене состояния. Согласно (1.5) формула для действия дает ответ на вопрос: могут ли два состояния s и t быть соседними в некоторой истории (1.1). При этом предикаты состояния, включающие все переменные без апострофов, можно рассматривать как особую разновидность действий. Действиями также являются булевы выражения, включающие только темпоральные переменные с апострофами. Данный способ математического представления действий (событий) в дискретных системах был предложен Пнуели [38].

Введенная интерпретация формул (1.5) позволяет описывать отдельные состояния в истории, а также пары смежных состояний. На основе (1.5) можно расширить определение формул для историй. Темпоральные формулы согласно выражению (1.2) интерпретируются для историй бесконечной длины. Для описания историй бесконечной длины с использованием действий вводится специальный модальный оператор необходимости ($\Box$), который читается «всегда». Смысл введенного оператора определяется через интерпретацию следующим образом:

$$\sigma\,[[\Box F]] \triangleq \forall n \in \mathbb{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots\rangle\,[[F]]\,. \tag{1.6}$$

Под формулой F в выражении (1.6) понимаются произвольные предикаты состояния и действия. Согласно ранее введенным определениям, сами действия и предикаты состояния определены для пары соседних состояний в истории. Чтобы не допускать синтаксических ограничений на структуру темпоральной формулы (обязательность модельного оператора $\Box$ в произвольной формуле), описывающей множество бесконечных историй, смысл функций состояния и действий расширяют на бесконечные истории следующим образом:

$$\langle s_0, s_1, s_2, \ldots\rangle\,[[A]] \triangleq s_0\,[[A]]\,s_1,$$

$$\langle s_0, s_1, s_2, \ldots\rangle\,[[P]] \equiv s_0\,[[P]]\,.$$

С учетом введенных определений становится возможным описание множества историй с использованием темпоральных формул. Описывать беско-

нечные истории можно на основе следующих интерпретаций формул, построенных из действий и предикатов состояний с использованием модального оператора $\Box$ «всегда»:

$$\langle s_0, s_1, s_2, \ldots\rangle[[\Box A]] \equiv \forall n \in \mathbb{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots\rangle[[A]] \equiv \forall n \in \mathbb{N} : s_n\,[[A]]\,s_{n+1},$$

$$\langle s_0, s_1, s_2, \ldots\rangle\,[[\Box P]] \equiv \forall n \in \mathbb{N} : s_n\,[[P]]\,.$$

Сложные темпоральные формулы строятся путем комбинирования предикатов состояний и действий с использованием логических связок и кванторов. Для передачи смысла таких формул достаточно определить семантику логической основы формулы для произвольной системы базисных функций. Например, семантику логической основы составной формулы F можно передать выражениями

$$\sigma\,[[\neg F]] \triangleq \neg\sigma\,[[F]]\,,$$

$$\sigma\,[[F \wedge G]] \triangleq \sigma\,[[F]] \wedge \sigma\,[[G]]\,,$$

и распространить на другие логические связки и кванторы очевидным образом по их определению из булевой алгебры.

Таким образом, темпоральная логика позволяет на базе традиционной математической нотации, расширенной знаками ($'$) и ($\Box$), описать множество историй, тем самым точно определить поведение требуемого вычислительного процесса.

1.2.2 Метод Лампорта для спецификации дискретных систем

Для практически важных дискретных систем, в том числе для вычислительных процессов, требуется рассматривать еще более узкий класс множеств всех историй, а именно историй, инвариантных по отношению к преобразованию дублирования одного состояния в соседних отсчетах времени (*stuttering*). Это ограничение, также как и бесконечность историй, связано с требованием корректной передачи поведения композиции нескольких дискретных систем.

Рассмотрим поведение системы, включающей две не взаимодействующие подсистемы. Синтаксически это выражается отсутствием одинаковых переменных в темпоральных формулах, описывающих каждую подсистему в отдельности. Естественно ожидать, что формула для системы в целом — это конъюнкция формул для ее подсистем. То есть формула, описывающая множество историй, в которых обе системы функционируют правильно. Наша система по построению реализует поведение своих подсистем. Логично ожидать, что «реализация» описывается как импликация формул: любая история, допустимая для системы, допустима также для ее подсистем («формула для системы»→«формула для подсистемы»). Теперь конкретизируем наши рассуждения, представив, что первая система периодически увеличивает значение переменной x, а вторая — периодически увеличивает значение переменной y. Так как системы не зависимы, то допустимы истории, в которых значение x увеличивается, а значение y остается неизменным и наоборот. Для сохранения смысла импликации, передающей отношение «реализации», требуется, чтобы описание любой подсистемы и системы в целом допускало повторение одинаковых состояний в соседних отсчетах дискретного времени. Синтаксически это ограничивает рассмотрение всех темпоральных формул только формулами, сводимыми к виду (1.7), которые и называются формулами *TLA*.

$$\Phi \triangleq I \wedge \Box \left[N\right]_f \wedge F. \tag{1.7}$$

Формула (1.7) имеет следующую структуру: I — предикат состояния, описывающий начальное состояние (вычислительного) процесса; N — действие, задающее множество всех атомарных операций дискретной системы; F — множество формул вида WF и SF, определенных для подмножества атомарных операций из N. Дополнительные синтаксические элементы *TLA* определяются через введенные выше обозначения посредством соотношений (1.8)–(1.18).

$$s\,[[\mathit{Enabled}\ A]] \triangleq \exists t \in \mathbf{St} : s\,[[A]]\,t. \tag{1.8}$$

Выражение (1.8) является специальным предикатом, определяющим условие готовности действия A. Действие A готово к исполнению в состоянии s, если существует такое состояние t, в которое можно перейти в результате

выполнения данного действия.

$$f \triangleq (v_1, v_2, v_3, \ldots), f' \equiv (v_1, v_2, v_3, \ldots)' \equiv (v_1', v_2', v_3', \ldots). \tag{1.9}$$

Выражение (1.9) является функцией состояния, состоящей из кортежа переменных. Оно позволяет обобщить действие знака апострофа на кортеж переменных. Данное определение используется для описания специального действия «не изменяются следующие переменные. . . »:

$$\mathit{Unchanged}\ f \triangleq f = f'. \tag{1.10}$$

С использованием приведенных выше обозначений вводится определение для компактной записи действия, допускающего сохранение текущего состояния (1.11).

$$[A]_f \triangleq A \vee (f = f') \equiv A \vee \mathit{Unchanged}\ f. \tag{1.11}$$

В противоположность символу из определения (1.11) вводится символ для обозначения действия, вызывающего безусловную смену состояния (1.12).

$$\langle A \rangle_f \triangleq A \wedge (f \neq f') \equiv A \wedge \neg \mathit{Unchanged}\ f. \tag{1.12}$$

В темпоральных логиках вводится также оператор возможности $\Diamond$. В случае *TLA* — это оператор, который имеет значение «в конечном счете» (в отличие от символа $\Box$ «всегда»). Оператор возможности выражается посредством оператора необходимости $\Box$ на основе отношения двойственности модальных логик, как показано в (1.13).

$$\Diamond F \triangleq \neg \Box \neg F. \tag{1.13}$$

Семантическое определение оператора «в конечном счете» записывается в виде (1.14)). Оно читается следующим образом. Условие «в конечном счете» F выполняется для истории, если в ней имеется некоторый суффикс, для которого темпоральная формула F интерпретируется в истину.

$$\langle s_0, s_1, s_2, \ldots \rangle [[\Diamond F]] \equiv \exists n \in \mathbb{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [[F]]. \tag{1.14}$$

В рассматриваемой темпоральной логике представляют особый интерес две комбинации операторов ($\Box\Diamond$) и ($\Diamond\Box$). Сочетание операторов $\Box\Diamond$ можно прочитать как «периодически» или «время от времени». Так, если формула, записанная после комбинации операторов $\Box\Diamond$, является действием, то ее смысл определяется выражением (1.15).

$$\langle s_0, s_1, s_2, \ldots \rangle \, [[\Box\Diamond A]] \equiv \forall n \in \mathbb{N} : \exists m \in \mathbb{N} : s_{n+m} \, [[A]] \, s_{n+m+1}. \qquad (1.15)$$

Выражение (1.15) может читаться следующим образом: формула истинна $\Box\Diamond A$ для истории, если в любом ее суффиксе имеется пара состояний, смена которых вызвана действием A. Сочетание операторов $\Diamond\Box$ можно прочитать как «постоянно с некоторого времени». Если формула, записанная после комбинации операторов $\Diamond\Box$, является действием, ее смысл определяется выражением (1.16).

$$\langle s_0, s_1, s_2, \ldots \rangle \, [[\Diamond\Box A]] \equiv \exists n \in \mathbb{N} : \forall m \in \mathbb{N} : s_{n+m} \, [[A]] \, s_{n+m+1}. \qquad (1.16)$$

Выражение (1.16) может читаться следующим образом: формула $\Diamond\Box A$ истинна для истории, если история содержит суффикс, в котором каждая пара состояний связана действием A (смена которых вызвана действием A). На основе определений (1.15) и (1.16) формулируются понятия «справедливость в слабом смысле» и «справедливость в сильном смысле», необходимые для описания асинхронных процессов (1.17) и (1.18).

$$\mathrm{WF}_f(A) \triangleq (\Box\Diamond\langle A\rangle_f) \vee (\Box\Diamond\neg \mathit{Enabled}\langle A\rangle_f). \qquad (1.17)$$

$$\mathrm{SF}_f(A) \triangleq (\Box\Diamond\langle A\rangle_f) \vee (\Diamond\Box\neg \mathit{Enabled}\langle A\rangle_f). \qquad (1.18)$$

Подробное объяснение определений (1.8)–(1.18) и их словесная интерпретация приведены в работе [26]. Для понимания модели существенным является смысл выражений (1.11), (1.17) и (1.18). Определение (1.11) читается «либо выполнено действие A, либо состояние системы, описываемое перемен-

ными f, не изменилось». Определение (1.17) называется «справедливостью в слабом смысле» (*weak fairness*). Оно читается следующим образом: «если, начиная с некоторого момента времени, выполнить действие A возможно, и эта возможность сохраняется неопределенно долго, то действие A будет выполнено неопределенно много раз». Определение (1.18) называется «справедливостью в сильном смысле» (*strong fairness*). Оно, в отличие от (1.17), в качестве условия выполнения требует, чтобы, начиная с некоторого момента времени, действие можно было бы выполнить неопределенно много раз.

Возвращаясь к примеру о композиции системы, периодически увеличивающей значение переменной x, и системы, увеличивающей y, можно понять смысл свойства «справедливости». Из формул (1.17)–(1.18) очевидно следует назначение условий F в формуле (1.7): при сохранении инвариантности к *stuttering*–преобразованию исключаются из описания истории, в которых N может выполниться, но не выполняется бесконечное число отсчетов дискретного времени. То есть, формулы (1.17)–(1.18) исключают из описания системы истории, при которых условия выполнения некоторых действий (возникновения событий) соблюдаются, однако состояние системы не изменяется. Это как раз и позволяет исключить из рассмотрения случаи нежелательной самопроизвольной остановки системы.

1.2.3 Построение модели типовых решений на основе метода Лампорта

Для описания типовых решений взяты алгоритмы, структуру которых можно описать ориентированным графом вида

$$G_M \triangleq (P, C), P \subset \mathbb{N}, C \subset \mathbb{N} \times \mathbb{N}, \tag{1.19}$$

в котором вершины P представляют собой процессы, а дуги C каналы, по которым передаются сообщения между процессами. Состояние вычислительного процесса — это значения множества переменных, связанных с процессами и каналами:

$$f \triangleq (i \in P : a_i; (i, j) \in C : e_{i,j}, d_{i,j}), \tag{1.20}$$

где $a_i \in \{1,0\}, b_{i,j} \in \{1,0\}, d_{i,j} \in \{1,0\}$.

Логические переменные a_i обозначают пассивное 0 или активное 1 состояние процесса i; $e_{i,j}$ пассивное 0 или активное 1 состояние канала (i,j); $d_{i,j}$ — направление передачи сообщений по двунаправленным каналам (1 — $i \to j$; 0 — $j \to i$).

В начальном состоянии некоторые каналы находятся в активном состоянии с передачей сообщений в направлении $i \to j$, все процессы находятся в пассивном состоянии:

$$I \triangleq \forall i \in P : \neg a_i \wedge \forall (i,j) \in C : (e_{i,j} \wedge d_{i,j}) \vee (\neg e_{i,j} \wedge \neg d_{i,j}). \tag{1.21}$$

Действия, выполняемые каналами, описываются формулой

$$\begin{aligned} A_c(i,j) \triangleq \{ & e_{i,j} \wedge \neg e'_{i,j} \wedge \left(\left(d_{i,j} \wedge d'_{i,j} \right) \vee \left(\neg d_{i,j} \wedge \neg d'_{i,j} \right) \right) \} \vee \\ & \vee \{ \neg e_{i,j} \wedge e'_{i,j} \wedge \left(\left(d_{i,j} \wedge \neg d'_{i,j} \right) \vee \left(\neg d_{i,j} \wedge d'_{i,j} \right) \right) \}. \end{aligned} \tag{1.22}$$

Действия описывают соседние в истории вычислительного процесса состояния, переменные следующего состояния обозначены апострофом. Согласно формуле (1.22) состояние канала периодически меняется с активного на пассивное, а направление передачи данных с прямого на обратное.

Действия, выполняемые процессами, описываются формулой

$$\begin{aligned} A_p(i) \triangleq & \left\{ \neg a_i \wedge a'_i \wedge \left(\exists! j : e_{i,j} \wedge \neg e'_{i,j} \wedge \neg d'_{i,j} \right) \wedge \neg \left(\exists j : e_{j,i} \wedge \neg e'_{j,i} \wedge d'_{j,i} \right) \right\} \vee \\ & \vee \left\{ \neg a_i \wedge a'_i \wedge \left(\exists! j : e_{j,i} \wedge \neg e'_{j,i} \wedge d'_{j,i} \right) \wedge \neg \left(\exists j : e_{i,j} \wedge \neg e'_{i,j} \wedge \neg d'_{i,j} \right) \right\} \vee \\ & \vee \left\{ a_i \wedge \neg a'_i \wedge \bigvee_{k:(i,k) \in C} \left(\neg e_{i,k} \wedge e'_{i,k} \wedge d'_{i,k} \right) \right\} \vee \\ & \vee \left\{ a_i \wedge \neg a'_i \wedge \bigvee_{k:(k,i) \in C} \left(\neg e_{k,i} \wedge e'_{k,i} \wedge d'_{k,i} \right) \right\} \vee \{ a_i \wedge \neg a'_i \}. \end{aligned} \tag{1.23}$$

Процесс становится активным, когда из смежного с ним канала посту-

пает сообщение. При этом такой канал сам переходит в пассивное состояние. Среди нескольких каналов, готовых передать сообщение в процесс, выбирается произвольный канал. Когда процесс завершит работу и перейдет в пассивное состояние, по одному или нескольким каналам от него могут передаваться сообщения в другие процессы.

С учетом выражений (1.22) и (1.23), действия, выполняемые моделируемым вычислительным процессом, можно представить в виде

$$N \triangleq \bigvee_{(i,j)\in C} A_c\,(i,j) \vee \bigvee_{i\in P} A_p\,(i)\,. \tag{1.24}$$

Способ планирования вычислительного процесса характеризуется выражением

$$F \triangleq (\forall i \in P : \mathrm{WF}_f\,(a_i \wedge \neg a_i')) \wedge \left(\forall\,(i,j) \in C : \mathrm{WF}_f\left(e_{i,j} \wedge \neg e_{i,j}'\right)\right), \tag{1.25}$$

подразумевающим справедливую в слабом смысле стратегию планирования.

Окончательно получаем модель вычислительных процессов для представления типовых решений:

$$\Phi_M \triangleq I \wedge \Box\,[N]_f \wedge F. \tag{1.26}$$

1.2.4 Дискретно-событийное имитационное моделирование типовых решений

Порядок смены состояний вычислительного процесса, удовлетворяющего формуле (1.26), легко воспроизвести дискретно-событийной имитационной моделью. Для этого организуется очередь каналов, находящихся в активном состоянии. На каждом шаге моделирования случайным образом выбирается активный канал. Далее он переводится в неактивное состояние, выполняется действие процесса и отправка сообщений из данного процесса. Блок-схема алгоритма, реализующего данные действия, представлена на рис. 1.1, на рис. 1.2 приведена реализация данного алгоритма.

Покажем, что приведенный на рис. 1.1, 1.2 алгоритм выполнения, ос-

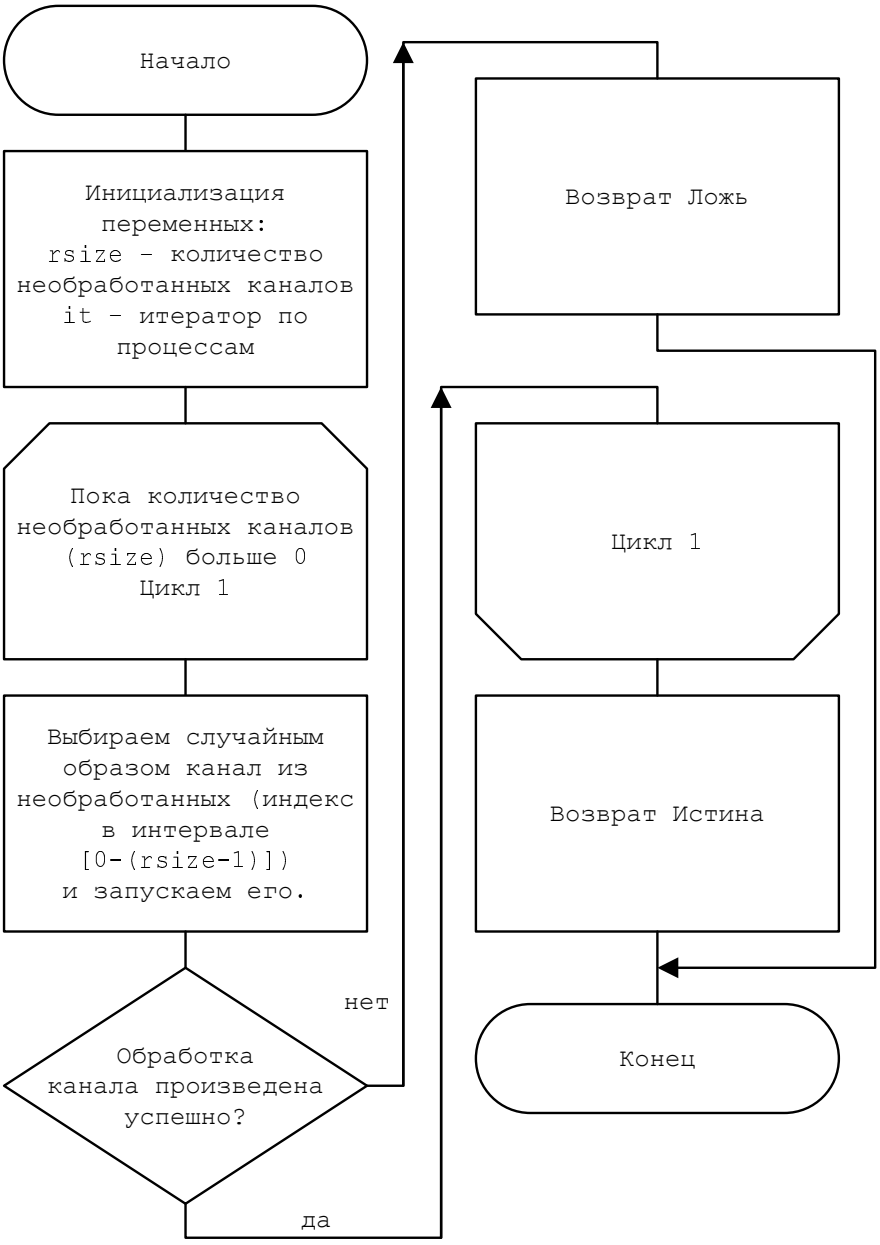


Рисунок 1.1 — Блок-схема алгоритма случайного выбора

нованный на случайном выборе активного канала, реализует вычислительный процесс, описываемый моделью (1.26). Для этого определим для него вид темпоральной формулы Φ_{IMP}.

Очевидно, что множество переменных f_{IMP}, используемых в формуле Φ_{IMP}, включает множество переменных f (1.20). В начальном состоянии системы Φ_{IMP} по построению реализуется начальное состояние модели (1.26) и определяется некоторое начальное состояние переменных, специфичных для алгоритма выполнения модели: $I_{IMP} \triangleq I \wedge C_1$, где C_1 предикат состояния, определённый на множестве переменных $f_{IMP} \setminus f$. Поэтому для предиката на-

```
bool Assemble::run()
{
  size_t rsize;
  std::vector<Channel*>::iterator it;
  while(rsize=ready.size())
  {
    std::vector<Channel*>::difference_type n=rand()%rsize;
    it=ready.begin()+n;
    Channel*c=*it;
    ready.erase(it);
    if(!c->_run())
    {
      return false;
    }
  }
  return true;
}
```

Рисунок 1.2 — Реализация алгоритма случайного выбора

чального состояния I_{IMP} в формуле Φ_{IMP} верно $I_{IMP} \to I$, так как $A \wedge B \to B$.

Рассмотрим вид формулы, описывающей изменение состояния алгоритма выполнения модели. Смысл ограничений, накладываемых последовательным способом исполнения исходной параллельной модели (1.26), заключается в том, что из множества возможных в текущий момент времени переходов из состояния $e_{i,j}$ в состояние $\neg e_{i,j}$ выбирается только один. Эти ограничения выражаются некоторой формулой C_2. Ограничения определяются в виде конъюнкции, поэтому $N_{IMP} \triangleq N \wedge C_2$. Из этого следует, что $\square [N \wedge C_2]_{f_{IMP}} \to \square [N]_f$, так как $\square [N \wedge C_2]_{f_{IMP}} = \square [N]_{f_{IMP}} \wedge \square [C_2]_{f_{IMP}} = \square [N]_f \wedge \square (N \vee f_{IMP} = f'_{IMP}) \wedge \square [C_2]_{f_{IMP}}$.

Способ планирования, основанный на извлечении произвольного активного канала, реализует формулу F (1.25) по следующим соображениям. Если некоторый канал находится в очереди *ready* (рис. 1.2) достаточно долго, то он будет извлечён на выполнение. То есть $\forall (i,j) \in C : \mathrm{WF}_f \left(e_{i,j} \wedge \neg e'_{i,j}\right)$ выполняется. Выражение $\forall i \in P : \mathrm{WF}_f (a_i \wedge \neg a'_i)$ выполняется, если не происходит зацикливания в вызове *c–>_run()*. Зацикливания не происходит, так как в данном вызове выполняется последовательный фрагмент алгоритма численного моделирования, который, очевидно, завершается. Из этого следует, что $F_{IMP} \triangleq F$, то есть $F_{IMP} \to F$.

Так как из $X_1 \to Y_1, X_2 \to Y_2 \ldots X_n \to Y_n$ следует $X_1 \wedge X_2 \wedge \ldots \wedge X_n \to$

$Y_1 \wedge Y_2 \wedge \ldots \wedge Y_n$, то из вышеприведённых рассуждений следует $\Phi_{IMP} \rightarrow \Phi$. Это означает верность исходного утверждения о реализации модели (1.26) алгоритмом на рис. 1.1, 1.2.

Приведённые рассуждения можно распространить и для параллельных алгоритмов реализации модели. Такие реализации используют паттерн «потоковый пул» [77], принципиально отличающийся от имитационного выполнения с точки зрения данного доказательства лишь заменой структуры *std::vector<Channel*>* на очередь готовых каналов.

Таким образом показано, что модель вычислительного процесса (1.26) является реализуемой в виде дискретно-событийной модели, имитирующей асинхронный характер вычислений, и обладает свойством адаптивности под тип параллельной исполняющей среды. Заметим, что рассмотренную модель также можно представить в виде сети Петри (Приложение В). Далее в п. 1.3 на примерах рассматривается вопрос адекватности графовой структуры модели (1.19) информационной структуре используемых на практике алгоритмов численного моделирования.

1.3 Построение визуальных моделей типовых решений

1.3.1 Определение графической нотации на примере алгоритма «разветвление-слияние»

Параллельная система в целом моделируется как композиция элементов модели «процесс» и элементов модели «канал» [9, 10, 28, 29]. В композиции процессов реализуется визуальное представление графа G_M модели (1.26). Интерфейсными элементами процессов, участвующих в композиции, являются порты. Система «разветвление-слияние» $ForkJoin$ в описываемой нотации представлена на рис. 1.3. Родительский процесс p выдает задания дочерним процессам $c1$ и $c2$, а затем выполняет обработку их ответов.

Пометки графа на рис. 1.3 обозначают: $p : Parent$ — процесс p типа $Parent$; $p1 \rightarrow p : Call$ — клиентский порт $p1$ (процесса p) связан с серверным портом p (процесса $c1$) каналом, имеющим тип $Call$. Точка на дуге указывает положение серверного порта.

Каналы. Диаграмма канала описывает объекты, задающие информационные связи между процессами. Диаграмма канала показывает, какие именно сообщения передаются между процессами, и определяет возможный порядок их передачи. Во взаимодействии по каналу участвуют два процесса: клиент и сервер. Возможна передача произвольного числа сообщений в обе стороны. В любой момент времени передается не более одного сообщения.

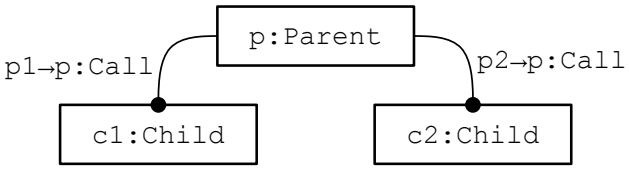


Рисунок 1.3 — Композиция процессов в системе «разветвление-слияние» ($ForkJoin$)

На рис. 1.4 показан протокол взаимодействия между родительским и дочерним процессами в системе «разветвление-слияние», описанный каналом $Call$. Пометки графа на рис. 1.4 обозначают: $s0$ — начальная вершина процесса-клиента; $s1$ — вершина процесса-сервера; $s2$ — вершина процесса-клиента; $call$ и ret — передаваемые сообщения.

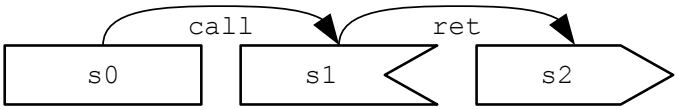


Рисунок 1.4 — Протокол взаимодействия процессов ($Call$) в системе «разветвление-слияние»

Состояние канала — это переменная, хранящая дугу с передаваемым сообщением и признак доставки сообщения. В момент отправки признак имеет значение «отправлено». Рано или поздно признак примет значение «доставлено», начнется обработка данного сообщения, и будет возможен его прием. В начальном состоянии переменная хранит специальное значение, сигнализирующее об отсутствии сообщения.

В начальном состоянии возможна передача сообщений процессом-клиентом по дугам, исходящим из начальной вершины (с пометкой как у $s0$ на рис. 1.4). Новым состоянием канала будет дуга, соответствующая переданному сообщению.

Если состоянием канала является дуга, то возможен прием сообщения, помечающего дугу. Принять сообщение может процесс-клиент, если вершина, в которую входит дуга, вершина процесса-клиента. Иначе — сообщение может принять процесс-сервер. Принимающий сообщение процесс может отправить ответное сообщение из тех, которые помечают дуги, исходящие из рассматриваемой вершины. Аналогично, новым состоянием канала будет дуга, соответствующая переданному сообщению.

Граф на рис. 1.4 показывает, что вначале клиент (родительский процесс p на рис. 1.3) передает дочернему процессу сообщение $call$, затем получает от него сообщение ret. На этом взаимодействие заканчивается.

Процессы. Диаграмма процесса описывает алгоритм обработки поступающих сообщений, исполняемый в течении времени пока $a_i = 1$ в формуле (1.20). Результатом обработки может являться отправка новых сообщений и/или изменение состояния процесса. Процессы являются пассивными, управляемыми сообщениями: алгоритм обработки запускается только при поступлении сообщения. Алгоритм обработки не может быть прерван другим сообщением до его завершения.

Примеры графических обозначений, используемых на диаграммах процессов, показаны на рис. 1.5 и рис. 1.6.

Пометки графов обозначают: $fork$ — начальная вершина, запускающая алгоритм обработки в начальном состоянии; $call$ и ret — получаемые или отправляемые сообщения; $p1$ и $p2$ — клиентские порты процессов; p — серверный порт процесса; $join$, $proc$, $error$ — методы обработки сообщений в процессах; yes, no — передача управления в случае успешного или не успешного завершения процедуры обработки.

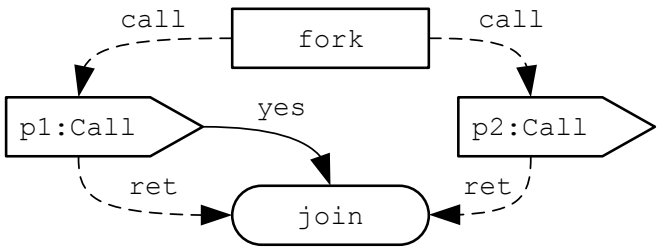


Рисунок 1.5 — Родительский процесс ($Parent$) в системе «разветвление-слияние»

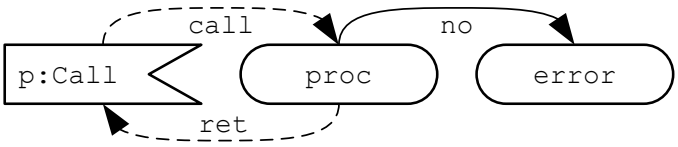


Рисунок 1.6 — Дочерний процесс ($Child$) в системе «разветвление-слияние»

Состояние процесса — это переменная, хранящая значение текущей вершины графа процесса. Когда обработка не выполняется, переменная хранит специальный признак останова. Если имеется сообщение, отправленное процессу, то рано или поздно начнется обработка данного сообщения с вершины-порта (p, $p1$ или $p2$ на рис. 1.5, рис. 1.6). Переменная состояния процесса примет значение соответствующей вершины, и будет возможен прием сообщения.

Правило передачи управления для портов:

а) если имеется исходящая дуга с пометкой yes, то управление передается по ней;

б) если имеется исходящая дуга, помеченная поступившим сообщением, то управление передается по данной дуге.

Правило запуска процедуры обработки сообщений. Процедура запускается, если одновременно:

а) переменная состояния принимает значение вершины, помеченной данной процедурой;

б) возможен прием сообщений, ведущих в эту вершину;

в) возможна отправка сообщений, исходящих из данной вершины.

Отправка исходящих сообщений выполняется, если одновременно:

а) процедура была запущена;

б) процедура вернула признак успешного завершения.

Правило передачи управления для процедур:

а) передача управления по дуге с пометкой yes происходит, если процедура была запущена и вернула признак успешного завершения;

б) иначе происходит передача управления по дуге с пометкой no.

Если нет исходящих из вершин дуг, по которым можно выполнить переход в текущей ситуации, переменная состояния процесса принимает значение

признака останова.

Рассмотрим примеры процессов рис. 1.5 и рис. 1.6. Вычисления начинаются с выполнения метода $fork$ процесса $Parent$. Он формирует и отправляет сообщения $call$ дочерним процессам по портам $p1$ и $p2$. Так как из вершины не исходят дуги с пометками yes или no, на этом обработка заканчивается. Сообщения $call$ попадают в связанные дочерние процессы $Child$ через порт p. Запускается метод $proc$. Его параметрами являются поступившее сообщение $call$ и формируемое для отправки сообщение ret. В случае не успешного завершения метода $proc$ отправка сообщения ret не производится, а запускается метод $error$. Иначе в родительский процесс отправляется сообщение ret. При поступлении сообщения на порт $p1$ в процесс $Parent$ управление немедленно передается в метод $join$. Альтернативную возможность передачи управления реализует порт $p2$. Здесь происходит проверка типа поступившего сообщения. Метод $join$ родительского процесса $Parent$ запускается, когда ответные сообщения ret придут от обоих дочерних процессов.

Отметим некоторые особенности представленной нотации. Диаграммы процессов внешне напоминают блок-схемы алгоритмов. Однако в приведенной нотации имеется несколько типов начальных вершин ($fork$ и $p1$ на рис. 1.5, p на рис. 1.6). Не делается различий между процедурой и принятием решения, что также используется в некоторых нотациях [26]. Поток управления может пройти через вершину-метод, при этом метод может не выполниться, если не соблюдены некоторые условия. Данные свойства позволяют удобно описывать выбор метода обработки поступающих в процесс сообщений.

В отличие от блок-схем точно задан контекст запуска метода. Определено, какие сообщения пересылаются между процессами и их порядок. Это позволяет провести анализ непротиворечивости графического описания алгоритма. Семантика исполнения модели по построению гарантирует отсутствие состояния состязания (*race condition*) при реализации алгоритма в общей памяти и состояния тупика (*deadlock*). Хотя для диффузных моделей возможна остановка в незавершенном состоянии, данная ситуация более проста, чем состояние тупика.

1.3.2 Графическая модель «метод переменных направлений»

Рассмотрим, каким образом может быть осуществлено распараллеливание численных моделей, основанных на методе переменных направлений. Пример также иллюстрирует технологию разработки произвольных последовательно-параллельных программ в нотации *Templet* [9, 10, 28, 29].

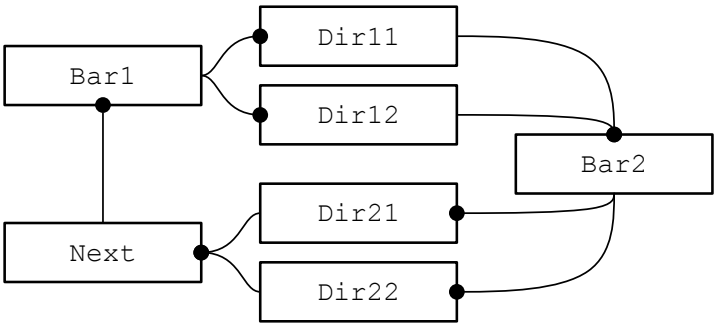
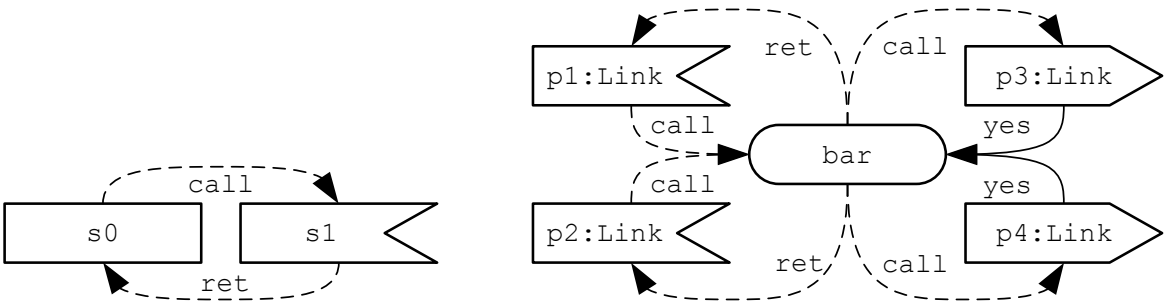


Рисунок 1.7 — Композиция процессов $AltDir1$ в методе переменных направлений с функциональной декомпозицией

Структура процессов для функциональной декомпозиции показана на рис. 1.7 Процессы сверху $dir11$ и $dir12$ выполняют вычисления горизонтальных независимых прогонок на $1/2$ шага, процессы снизу $dir21$ и $dir22$ выполняют вычисления вертикальных независимых прогонок при окончании шага (или наоборот: верхние процессы — вертикальные, нижние — горизонтальные прогонки). Реализация барьера по окончании $1/2$ шага изображена на рис. 1.8(б). Метод bar отправляет сообщения, если закончились все входящие параллельные ветви. Контроль корректности последовательно-параллельной структуры реализуется при помощи ответных сообщений ret в каналах $Link$ на рис. 1.8(а). В программной реализации по модели данные сообщения не обязательны. Другие два барьерных процесса устроены аналогично процессу рис. 1.8(б).

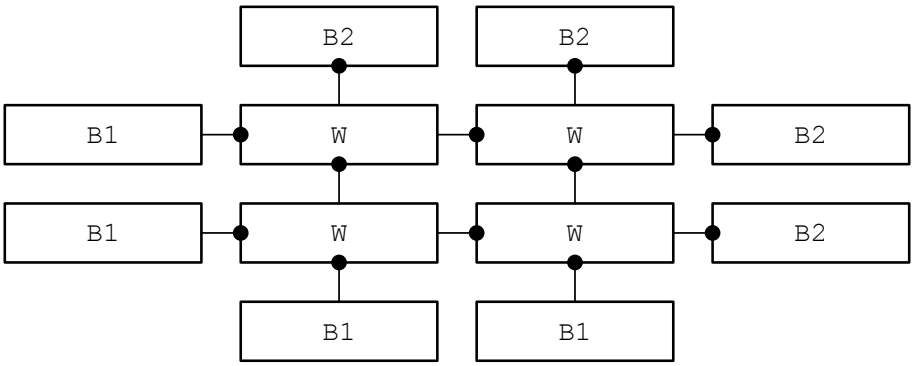
Алгоритм основанный на методе переменных направлений также можно распараллелить в технике декомпозиции по данным. Ниже на рис. 1.9 и рис. 1.10 представлен пример построения параллельных алгоритмов для двумерных сеток из процессорных элементов.

Протокол канала для данного вида декомпозиции аналогичен рис. 1.8(а). Граничные процессы на сетке рис. 1.10 используются для запуска прямых и

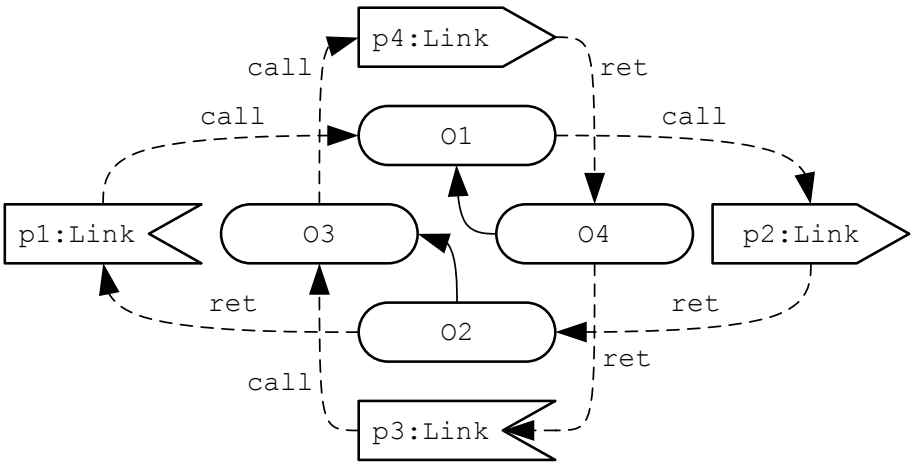


(а) (б)

Рисунок 1.8 — Описание протокола канала $Link$ (а) и барьера Bar (б) в методе переменных направлений с функциональной декомпозицией



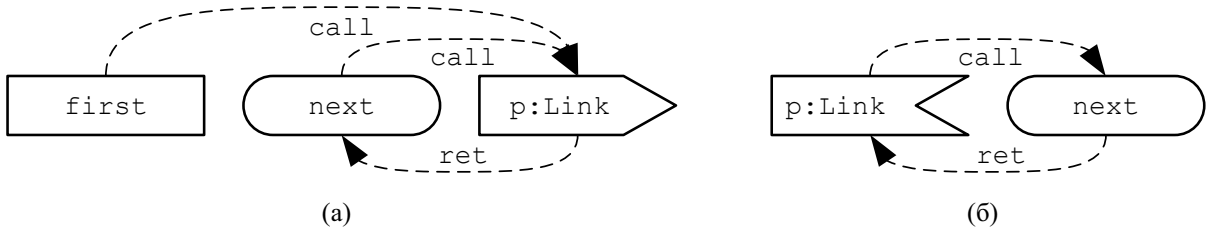
(а)



(б)

Рисунок 1.9 — Композиция процессов $AltDir2$ (а) и рабочий процесс W (б) в методе переменных направлений с декомпозицией по данным

обратных ходов прогонок соответственно. Основу алгоритма составляют рабочие процессы сетки рис. 1.9(б). Метод $O3$ отвечает за выполнение прямого хода вертикальной прогонки, а $O1$ — прямого хода горизонтальной прогонки. Метод $O4$ отвечает за выполнение обратного хода вертикальной прогонки, а



(а) (б)

Рисунок 1.10 — Граничные процессы $B1$ (а) и $B2$ (б) в методе переменных направлений с декомпозицией по данным

$O2$ — обратного хода горизонтальной прогонки.

Правильная очередность запусков методов в процессе рис. 1.9(б) организуется следующим образом. В каждом процессе хранится номер текущего направления прогонки. У каждого метода имеется предусловие, проверяющее можно ли начать обработку. Если обработку можно начать при запуске метода, то изменяется номер текущего направления на следующее, выполняется прогонка по текущему направлению и возвращается значение «истина». Если нельзя начать обработку, метод возвращает значение «ложь». Кроме этого в начале каждого цикла прогонок проверяется условие остановки. Оно может быть реализовано как предусловие метода next рис. 1.10.

1.3.3 Графическая модель «конвейер»

Пример вычислительного «конвейера» иллюстрирует использование линейной регулярной структуры процессов. Он может применяться как в алгоритмах с функциональной декомпозицией, так и в алгоритмах с декомпозицией по данным. Показан способ инициализации и остановки вычислений в предлагаемой нотации. Композиция процессов конвейера рис. 1.11 — это цепочка из процессов типа $Center$, замкнутая с концов процессом типа $Left$ и процессом типа $Right$. Особенностью инициализации вычислений в конвейере является то, что все его ступени, кроме r:Right, должны получить сообщение ret. Это достигается посылкой сообщения ret самому себе из методов $first$, что видно на рис. 1.12(в), рис. 1.12(а), рис. 1.13. В методах $first$ также может выполняться инициализация.

Остановкой вычислений управляют методы next процессов. Для остановки требуется, чтобы метод вернул значение «ложь». Это приводит к тому,

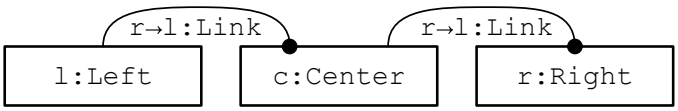


Рисунок 1.11 — Композиция процессов $Pipline$ конвейера

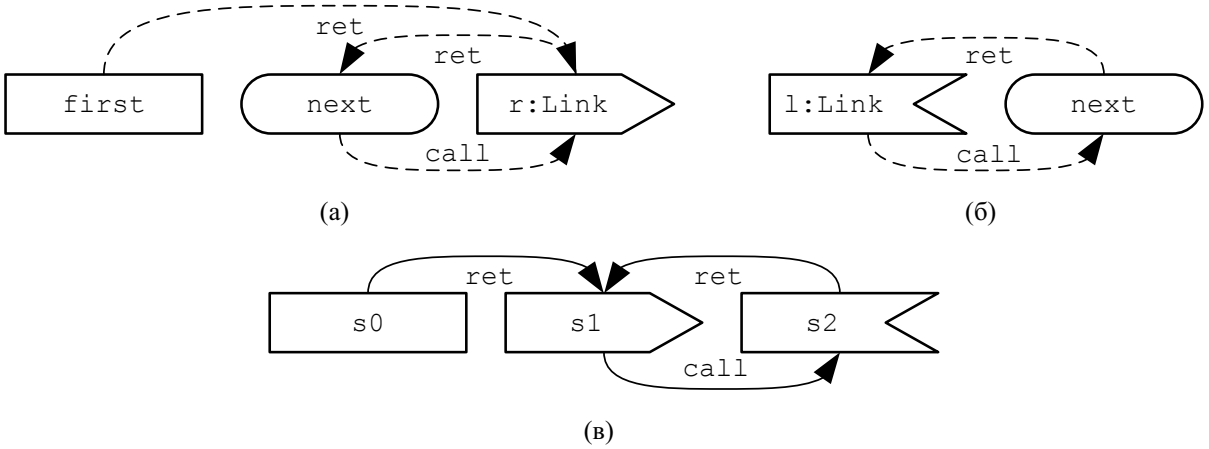


Рисунок 1.12 — Левый $Left$ (а) и правый $Right$ (б) процессы и протокол канала $Link$ (в) конвейера

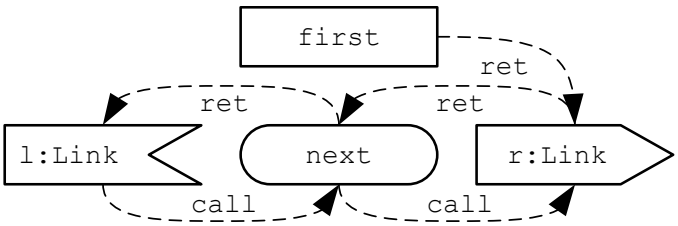


Рисунок 1.13 — Центральный процесс конвейера $Center$

что не отправляются исходящие сообщения: ret — процессу слева и $call$ — процессу справа. При остановке процессы выходят из вычислений от крайнего левого к крайнему правому.

1.3.4 Графическая модель «портфель задач»

Типовыми схемами решения многих переборных и оптимизационных задач являются схемы «управляющий — рабочие». Рассматривается модель одной из общих схем управления вычислениями из данного класса: «вычисление с портфелем задач». Это схема с активными рабочими, реализующая управление *pull*-типа. Здесь иллюстрируется описание более сложного протокола

взаимодействия процессов с несколькими типами сообщений (по сравнению с исходными (1.22) протоколами «запрос-ответ»).

На рис. 1.14 показана композиция процессов схемы «портфель задач», соответствующая графу (1.19). Обработка заданий выполняется по инициативе рабочих процессов $w1, \ldots, wN$, что видно из направления каналов $p \to p1 : Link, \ldots, p \to pN : Link$ от рабочих к управляющему процессу.

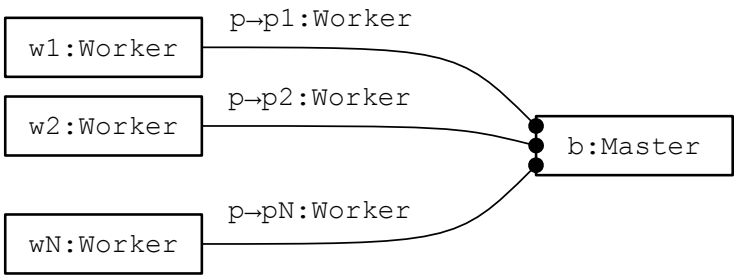


Рисунок 1.14 — Композиция процессов при вычислении с портфелем задач $TaskBag$

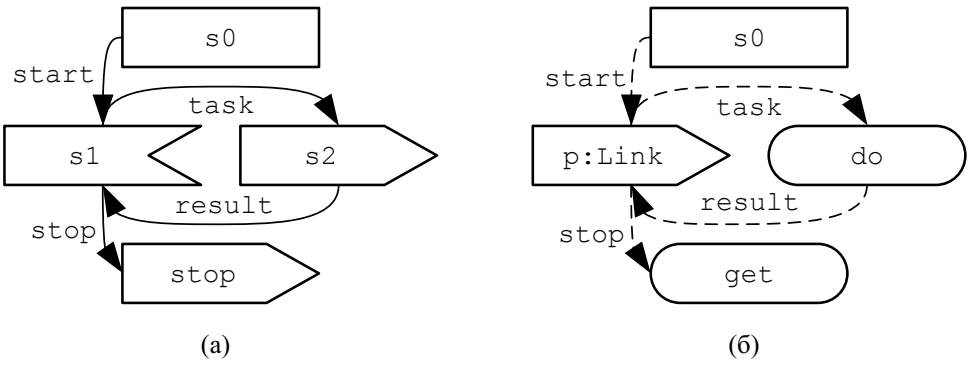


Рисунок 1.15 — Протокол канала $Link$ (а) и рабочий процесс $Worker$ (б) алгоритма вычисления с портфелем задач

Роль рабочих процессов рис. 1.15(б) заключается в периодическом получении задания от управляющего процесса в сообщении $task$ и отправки результата его обработки $result$. Обработка выполняется в методе do. Цикл обработки начинается по инициативе рабочего с отправки сообщения $start$ и заканчивается при получении от управляющего процесса сообщения $stop$. Если потребуется, в методе $stop$ можно выполнить операцию очистки.

На рис. 1.16 показан фрагмент управляющего алгоритма, реализованного в процессе $Master$, относящийся к порту pN. Методы процесса get, put, if_task и $start$ реализуют специфические для решаемой задачи операции:

get — извлечение очередной задачи; put — сохранение результата; if_task — проверку наличия задачи для обработки. Метод $start$ может использоваться для инициализации.

Методы if_active, $wait_N$, $stop_N$, $waiting_N$ являются универсальными для произвольного численного метода. Метод if_active возвращает истинное значение, если имеются активные рабочие процессы. То есть, имеются каналы, не находящиеся в состоянии «доставлено сообщение $start$ или $result$», что в программной реализации определяется простым подсчетом при поступлении и отправке сообщений. Метод $wait_N$ устанавливает признак, что канал N находится в состоянии ожидания появления заданий, метод $waiting_N$ возвращает значение данного признака. Метод $stop_N$ используется при отправке сообщения $stop$ рабочему для начала очистки.

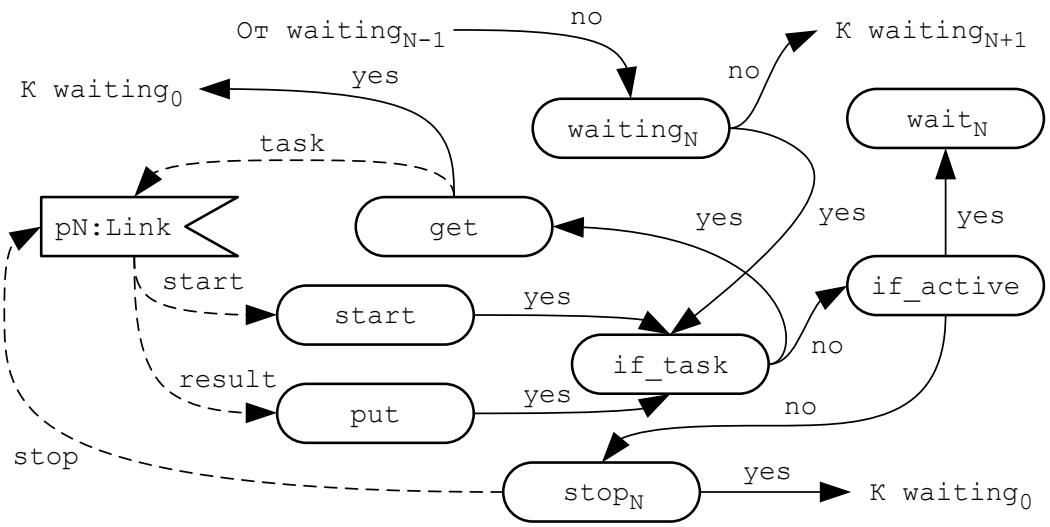


Рисунок 1.16 — Фрагмент диаграммы управляющего процесса $Master$ для алгоритма вычисления с портфелем задач

Остановка вычислений и очистка начинаются, если нет задач и нет активных рабочих процессов. Когда сформирована и отправлена очередная задача, происходит поиск ожидающего рабочего процесса. Если такой процесс найдется и есть еще одна задача для обработки, то выполняется ее формирование и отправка рабочему. В противном случае управляющий процесс либо заканчивает обработку сообщения и ждет ответа от рабочих, либо начинает остановку вычислений и очистку.

1.4 Выводы

а) Рассмотрены методы моделирования параллельных вычислений. Показано, что для современных средств реализации параллельных вычислений применительно к задачам численного моделирования целесообразно использовать модель процессов, типовые решения, автоматизированное распараллеливание и визуализацию моделей.

б) С использованием темпоральной логики Лампорта построена модель процессов для описания типовых решений.

в) Для модели типовых решений выполнено построение монитора моделирования, позволяющего воспроизвести исполнение параллельного типового процесса на последовательном компьютере. Доказано, что данный монитор моделирования реализует модель типовых решений.

г) Для проверки описательных возможностей предложенной модели с использованием метода визуализации *Templet* [29] выполнено построение имитационных моделей типовых решений «портфель задач», «конвейер», «метод переменных направлений».

ГЛАВА 2. СИНТЕЗ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ОБУЧЕНИЯ ГИПЕРРАДИАЛЬНОЙ НЕЙРОННОЙ СЕТИ НА ОСНОВЕ ТИПОВОГО РЕШЕНИЯ «ПОРТФЕЛЬ ЗАДАЧ» ДЛЯ РЕШЕНИЯ ЗАДАЧ ПРОГНОЗИРОВАНИЯ ВРЕМЕННЫХ РЯДОВ

В первой части второй главы приводится понятие нейронных сетей. Ставится задача прогноза временных рядов и обосновывается применение гиперрадиальных нейронных сетей в качестве универсального математического аппарата для решения данной задачи.

Применение гиперрадиальных нейронных сетей сопряжено с высокой вычислительной сложностью алгоритмов обучения. Во второй части главы выполнен синтез параллельного алгоритма обучения данного типа нейронной сети с использованием типового решения «портфель задач» на основе последовательного алгоритма с подбором структуры, что позволило снизить время обучения.

В третьей части главы в серии вычислительных экспериментов демонстрируется высокое быстродействие реализованного параллельного алгоритма обучения на примере решения задачи прогноза временных рядов биржевых котировок.

Результаты главы опубликованы в работах [11–13, 15, 19, 22].

2.1 Решение задачи прогнозирования временных рядов нейронной сетью

2.1.1 Понятие нейронной сети

Интерес к нейронным сетям постоянно возрастает. Они успешно используются в различных областях: технике, бизнесе, медицине, геологии, физике и т.д. Их применение оправдано там, где необходимо решать задачи прогнозирования, аппроксимации, моделирования, классификации, идентификации или управления.

Нейронные сети — это раздел теории искусственного интеллекта, в котором для обработки сигналов используются явления, аналогичные происходящим в нейронах живых существ [78]. Такой подход моделирования обладает

важным преимуществом: он позволяет реализовать сложные функциональные зависимости, что объясняется нелинейной природой нейронных сетей. Линейное моделирование долгое время являлось основным методом моделирования в широком круге решаемых задач, объяснялось это наличием хорошо разработанных процедур оптимизации. Но в задачах, где линейная аппроксимация неудовлетворительна, линейные модели работают плохо. Кроме того, нейронные сети справляются с «проклятием размерности» [79], которое не позволяет моделировать линейные зависимости в случае большого числа переменных.

Нейронные сети обладают способностью обучаться и обобщать полученные знания. То есть, сеть обученная на ограниченном множестве обучающих данных, накапливает и обобщает полученную информацию и способна вырабатывать ожидаемую реакцию на данные, которые в процессе обучения не участвовали [80]. Таким образом она может выявлять зависимости, которые при использовании других методов обработки информации выявить не удается.

Обучение нейронной сети заключено в формировании представительных данных и запуске алгоритма обучения. Алгоритм воспринимает структуру данных автоматически. Исследователю, который хочет использовать нейросетевую модель, необходим определенный уровень эвристических знаний о подготовке данных обучения, выборе архитектуры нейронной сети под конкретную задачу и интерпретацию результатов. Но объем этих знаний гораздо скромнее, чем объем, необходимый при использовании традиционных методов статистики.

Огромный потенциал нейронных сетей заложен в возможности параллельной обработки информации всеми нейронами одновременно. Это своего рода параллельный компьютер, где вычислительными узлами выступают нейроны. Благодаря этому свойству, появляется возможность обработки информации в реальном масштабе времени.

В нейронных сетях прямого распространения сигналы, проходящие через элементы, идут только в прямом направлении. Нейронные сети имеют слоистую архитектуру. Узлы входного слоя принимают поступающие на них входные сигналы. В последующих (скрытых) слоях, нейроны, в соответствии с используемой архитектурой, соединены с элементами предыдущего к ним слоя. Каждый нейрон вычисляет выходное значение, своей функции актива-

ции от входных сигналов, поступивших по установленным соединениям от элементов предыдущего слоя. Структура такого вида имеет устойчивое поведение.

Создание нейронной сети для решения конкретной задачи заключается в определении способов соединения нейронов друг с другом и подборе значения весовых коэффициентов этих связей. Взаимодействие элементов между собой зависит от установленных соединений, а весовой коэффициент определяет меру этого взаимодействия [81].

Нейронная сеть малочувствительна к ошибкам отдельных соединений, это объясняется большим количеством соединений элементов. Если возникают поврежденные соединения, их функции возьмут на себя другие связи в сети.

2.1.2 Постановка задачи прогноза временного ряда нейронной сетью

Временным рядом некоторой измеренной (наблюдаемой) динамической переменной $a\,(t)$ в последовательные моменты времени будем называть массив значений вида (2.1).

$$\{a_i\} : a_i = a(t_i); i \in \mathbb{N}. \tag{2.1}$$

Прогнозирование временного ряда — вычисление величины его будущих значений либо характеристик, позволяющих определить эту величину, на основании анализа известных значений [82, 83]. Величина, подлежащая прогнозу, называется прогнозируемой величиной.

При прогнозировании предполагается, что значение прогнозируемой величины зависит от каких-либо факторов. Один из подходов прогнозирования основан на предположении зависимости прогнозируемой величины от предыдущих значений временного ряда.

Схема решения задачи прогнозирования состоит из следующей последовательности этапов [84].

а) Постановка задачи и сбор исходной информации.

б) Предварительный анализ исходных временных рядов, редукция и уплот-

нение данных.

в) Формирование модели прогнозирования и ее численная оценка.

г) Получение прогноза (непосредственная экстраполяция).

д) Оценка полученного прогноза.

Существует несколько классификаций видов прогнозов по различным признакам. Основными из них являются глубина, используемый метод прогнозирования. Глубина прогноза — это число отсчетов, вычисляемых при прогнозировании. Для каждого временного ряда глубина прогноза находится в собственных диапазонах. В зависимости от глубины выделяют следующие виды прогнозов.

а) Краткосрочный — прогноз глубиной до 3% от наблюдений, но не более чем на 3 шага вперед, для данного типа прогнозов применяют практически все известные методы.

б) Среднесрочный — прогноз глубиной до 5% от объема наблюдений, но не более чем на 10 шагов вперед, для данного типа прогнозов применяют нейросетевые модели [85, 86], модели *АРПСС*, экспоненциального сглаживания.

в) Долгосрочный прогноз — для данного типа прогнозов стандартные методы практически не используются, кроме случаев очень «хороших» рядов.

Большое количество реальных процессов, в виду их нелинейности и хаотичности, не может быть представлено традиционными статистическими моделями [86]. Выходом в данной ситуации является применение аппарата искусственных нейронных сетей. Он позволяет, имея обучающую последовательность, осуществлять классификацию и прогнозирование [78–81].

Нейросетевой подход заключается в моделировании биологических нейронных сетей. Биологические системы способны к адаптации, самоорганизации и обучению, что является преимуществом перед остальными вычислительными системами.

Следует учесть, что на эффективность построения прогноза и обучения нейронной сети сильно влияет ее тип и архитектура [86]. Таким образом, при выборе нейронной сети и ее структуры, необходимо решить следующие задачи.

а) Выбор числа слоев нейронов.

б) Выбор количества нейронов в каждом слое.

В качестве перспективных в применении, можно отметить радиально-базисные нейронные сети. Они обладают рядом преимуществ.

а) Отличающиеся высокой скоростью обучения.

б) Обладают универсальными аппроксимирующими возможностями.

в) Содержат только один скрытый слой.

При этом, задача выбора числа слоев уже является решенной, остается произвести подбор количества нейронов единственного скрытого слоя.

В рамках данной работы для решения задачи прогнозирования была выбрана гиперрадиальная нейронная сеть, с помощью которой производилось построение краткосрочного прогноза.

2.1.3 Нейронная сеть на основе радиально-базисных функций

Нейронные сети на основе радиально-базисных функций, с точки зрения математики, выполняют аппроксимацию путем адаптации нескольких одиночных аппроксимирующих функций к ожидаемым значениям, причем эта адаптация проводится только в локальной области многомерного пространства [78]. При таком подходе отображение всего множества данных представляет собой сумму локальных преобразований, а скрытые нейроны составляют множество базисных функций локального типа.

В таких нейронных сетях, скрытые нейроны реализуют функции, радиально изменяющиеся вокруг выбранного центра и принимающие ненулевые значения только в окрестности этого центра:

$$\phi(X) = \phi\left(\|X - C\|\right), \tag{2.2}$$

где X — входной вектор, C — вектор координат центра.

Подобные функции, определяемые выражением (2.2), называются радиально-базисными функциями. Роль скрытого нейрона нейронной сети на основе радиально-базисных функций заключается в отображении радиального пространства вокруг одиночной заданной точки либо вокруг группы таких точек, образующих кластер. Суперпозиция сигналов, поступающих от всех скрытых нейронов, которая выполняется выходным нейроном, позволяет получить

отображение всего многомерного пространства. Нейрон на основе радиально-базисной функции представляет собой гиперсферу, которая осуществляет шаровое разделение пространства вокруг центральной точки рис. 2.1(б). Для сравнения приведено разделение пространства сигмоидальным нейроном рис. 2.1(а).

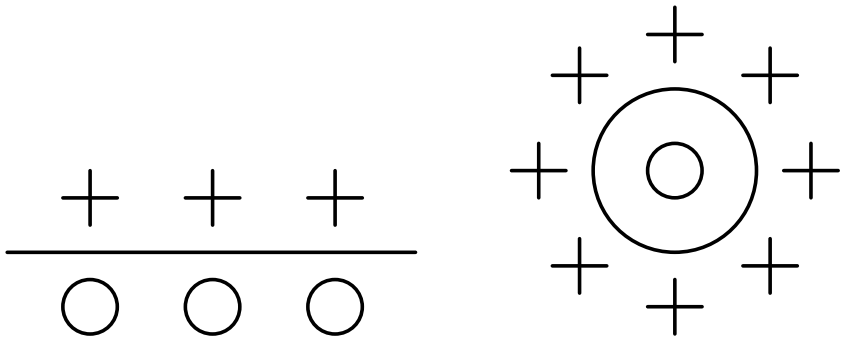


Рисунок 2.1 — Разделение пространства данных нейронами на основе сигмоидальной (а) и радиально-базисной (б) функций

Так как нейроны могут выполнять различные функции, в нейронных сетях данного типа отсутствует необходимость использования большого количества скрытых слоев. Структура типичной нейронной сети на основе радиально-базисных функций включает: входной слой, на который подаются сигналы, описываемые входным вектором; скрытый слой с нейронами на основе радиально-базисных функций; выходной слой, состоящий, как правило, из одного или нескольких линейных нейронов. Функция выходного нейрона сводится исключительно к взвешенному суммированию сигналов, генерируемых скрытыми нейронами.

Математическую основу функционирования нейронной сети на основе радиально-базисных функций составляет теорема Т. Ковера о разделимости образов [87]. Нелинейное преобразование сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов.

Если вектор радиально-базисных функций в N-мерном входном пространстве X ($X_l \in X$) обозначить как:

$$\phi\left(X_l\right) = \left(\phi^{(1)}\left(X_l\right), \phi^{(2)}\left(X_l\right), \ldots, \phi^{(K)}\left(X_l\right)\right)^T, \tag{2.3}$$

то это пространство является нелинейно ϕ-разделяемым на два пространственных класса X^+ и X^- тогда, когда существует такой вектор весов W, что выполняются неравенства

$$\begin{aligned} W^T\phi\left(X_l\right) > 0, X_l \in X^+, \\ W^T\phi\left(X_l\right) < 0, X_l \in X^-. \end{aligned} \tag{2.4}$$

Граница между этими классами определяется уравнением $W^T\phi\left(X_l\right) = 0$.

Ковер доказал, что каждое множество образов, случайным образом размещенных в многомерном пространстве, является ϕ-разделяемым с вероятностью 1 при условии большой размерности ($K > N$) этого пространства. На практике это означает, что применение достаточно большого количества скрытых нейронов K, реализующих радиально-базисные функции $\phi^{(k)}\left(X_l\right)$, гарантирует решение задачи аппроксимации при построении всего лишь двухслойной сети. При этом скрытый слой должен реализовать вектор $\phi\left(X_l\right)$, а выходной слой может состоять из единственного линейного нейрона, выполняющего суммирование выходных сигналов от скрытых нейронов с весовыми коэффициентами, заданными вектором W.

Простейшая нейронная сеть функционирует по принципу многомерной интерполяции, состоящей в отображении L различных входных векторов $X_l, l = \overline{1, L}$ из входного N-мерного пространства во множество из L рациональных чисел $d_l, l = \overline{1, L}$. Для реализации этого процесса необходимо использовать L скрытых нейронов на основе радиально-базисной функции и задать такую функцию отображения $y(X_l)$, для которой выполняется условие интерполяции:

$$y(X_l) = d_l. \tag{2.5}$$

С практической точки зрения использование в разложении большого числа радиально-базисных функций недопустимо, поскольку число обучающих выборок L велико, в результате вычислительная сложность обучающего алгоритма станет чрезмерной, а сама сеть адаптируется к разного рода шумам и нерегулярностям, сопровождающим обучающие выборки. Во избежании возникновения данной проблемы необходимо редуцировать количество весов, что приведет к уменьшению количества радиально-базисных функций.

При этом ищется субоптимальное решение в пространстве меньшей размерности, которое будет с достаточной точностью аппроксимировать точное решение. Если ограничиться K радиально-базисных функциями (рис. 2.2), то аппроксимирующее решение можно представить в виде:

$$y(X_l) = \sum\nolimits_{k=1}^{K} w^{(k)} \phi\left(\left\|X_l - C^{(k)}\right\|\right), \tag{2.6}$$

где $K < L$ а $C^{(k)}, k = \overline{1, K}$ — множество центров, которые необходимо определить.

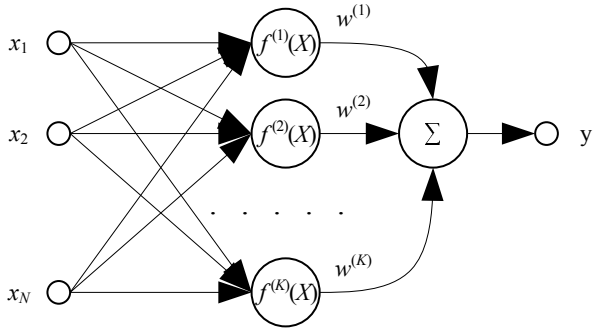


Рисунок 2.2 — Структура нейронной сети на основе радиально-базисных функций

В особом случае, если принять $K = L$ и $C^{(k)} = X_l$, то можно получить точное решение.

Задача аппроксимации заключается в подборе соответствующего количества радиально-базисных функций K вида $\phi\left(\left\|X_l - C^{(k)}\right\|\right)$ и их параметров, а также в подборе вектора весов $W = \left(w^{(1)}, w^{(2)}, \ldots, w^{(K)}\right)$, чтобы решение уравнения (2.5) было наиболее близким к точному. Подбор параметров радиально-базисных функций и значений весов W можно свести к минимизации целевой функции. Используя метрику Евклида, целевую функцию можно записать в форме:

$$E = \sum\nolimits_{l=1}^{L} \left(\sum\nolimits_{k=1}^{K} w^{(k)} \phi\left(\left\|X_l - C^{(k)}\right\|\right) - d_l\right)^2, \tag{2.7}$$

где K — количество нейронов на основе радиально-базисной функции, L — количество обучающих пар, где X_l — это входной вектор, а d_l — соответствующее ему ожидаемое выходное значение.

Если дополнить структуру сети на рис. 2.2 дополнительным параметром

$w^{(0)}$, дающим дополнительную степень свободы, то получим видоизмененную структуру сети (рис 2.3).

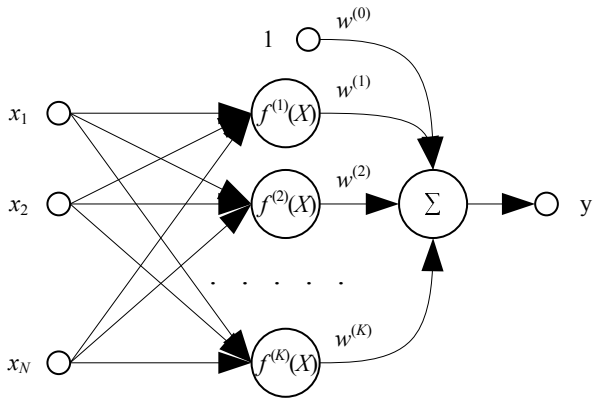


Рисунок 2.3 — Структура нейронной сети на основе радиально-базисных функций с параметром $w^{(0)}$

Преобразовав выражения (2.6) и (2.7) получим соответственно:

$$y(X_l) = w^{(0)} + \sum_{k=1}^{K} w^{(k)} \phi\left(\left\|X_l - C^{(k)}\right\|\right), \tag{2.8}$$

$$E = \sum_{l=1}^{L} \left(w^{(0)} + \sum_{k=1}^{K} w^{(k)} \phi\left(\left\|X_l - C^{(k)}\right\|\right) - d_l\right)^2. \tag{2.9}$$

В качестве функций активации нейронной сети далее будем использовать функцию Гаусса:

$$\phi\left(\left\|X_l - C^{(k)}\right\|\right) = exp\left(\frac{\left\|X_l - C^{(k)}\right\|}{2\left(\sigma^{(k)}\right)^2}\right) \tag{2.10}$$

где $\sigma^{(k)}$ — параметр, от значения которого зависит ширина функции.

2.1.4 Гиперрадиальная нейронная сеть

Требования к количеству скрытых нейронов можно смягчить применением в нейронах матрицы диаметров (масштабирующей матрицы) $Q^{(k)}$ размерности $N \times N$, реализующей масштабирование сигналов на входе k-й радиальной функции. Коэффициенты масштабирующей матрицы, связанные с соответствующими компонентами вектора X_l, представляют собой еще од-

ну группу параметров, подлежащих подбору и облегчающих аппроксимацию обучающих данных гиперрадиальной (*HRBF — Hyper Radial Basis Function*) нейронной сетью. За счет увеличения количества подбираемых параметров требуемая точность может быть достигнута сетью при меньшем числе нейронов [78].

Принципы функционирования гиперрадиальной нейронной сети описываются выражениями (2.11)–(2.15). Изначально на входные узлы сети подается входной вектор данных $X_l = (x_{l1}, x_{l2}, \ldots, x_{lN})$ для обучения, который является l-й строкой матрицы обучающих данных X размера $L \times N$, где L — количество обучающих пар, N — количество входных параметров обучения. Данные от этих узлов сдвигаются следующей функцией:

$$v_j^{(k)}(X_l) = x_{lj} - c_j^{(k)}, \tag{2.11}$$

где $C^{(k)} = \left(c_1^{(k)}, c_2^{(k)}, \ldots, c_N^{(k)}\right)$ — вектор центров радиальной функции k-ого нейрона. Далее результат сдвига масштабируется следующей функцией:

$$z_i^{(k)}(X_l) = \sum\nolimits_{j=1}^{N} q_{ij}^{(k)} \cdot v_j^{(k)}(X_l), \tag{2.12}$$

где $Q^{(k)} = \left(q_{ij}^{(k)}\right)$ — матрица диаметров радиальной функции k-ого нейрона размера $N \times N$. Затем квадраты масштабированных сигналов суммируются:

$$u^{(k)}(X_l) = \sum\nolimits_{i=1}^{N} \left(z_i^{(k)}(X_l)\right)^2. \tag{2.13}$$

И подаются как аргумент функции активации k-ого нейрона:

$$f^{(k)}(X_l) = \exp\left(-\frac{u^{(k)}(X_l)}{2}\right). \tag{2.14}$$

Получаем выход нейронной сети:

$$y(X_l) = w^{(0)} + \sum\nolimits_{i=1}^{K} w^{(k)} f^{(k)}(X_l), \tag{2.15}$$

где K — количество нейронов сети, $W = (w^{(0)}, w^{(1)}, \ldots, w^{(k)})$ — вектор весов выходного слоя.

Целевая функция, которая подлежит минимизации во время обучения, имеет следующий вид:

$$E(X, D) = \frac{1}{2} \sum_{l=1}^{L} (y(X_l) - d_l)^2, \tag{2.16}$$

где d_l — l-й элемент вектора отклика D длины L, X_l и d_l — это обучающая пара.

Потоковый граф описанной гиперрадиальной нейронной сети (2.11)–(2.15) представлен на рис. 2.4.

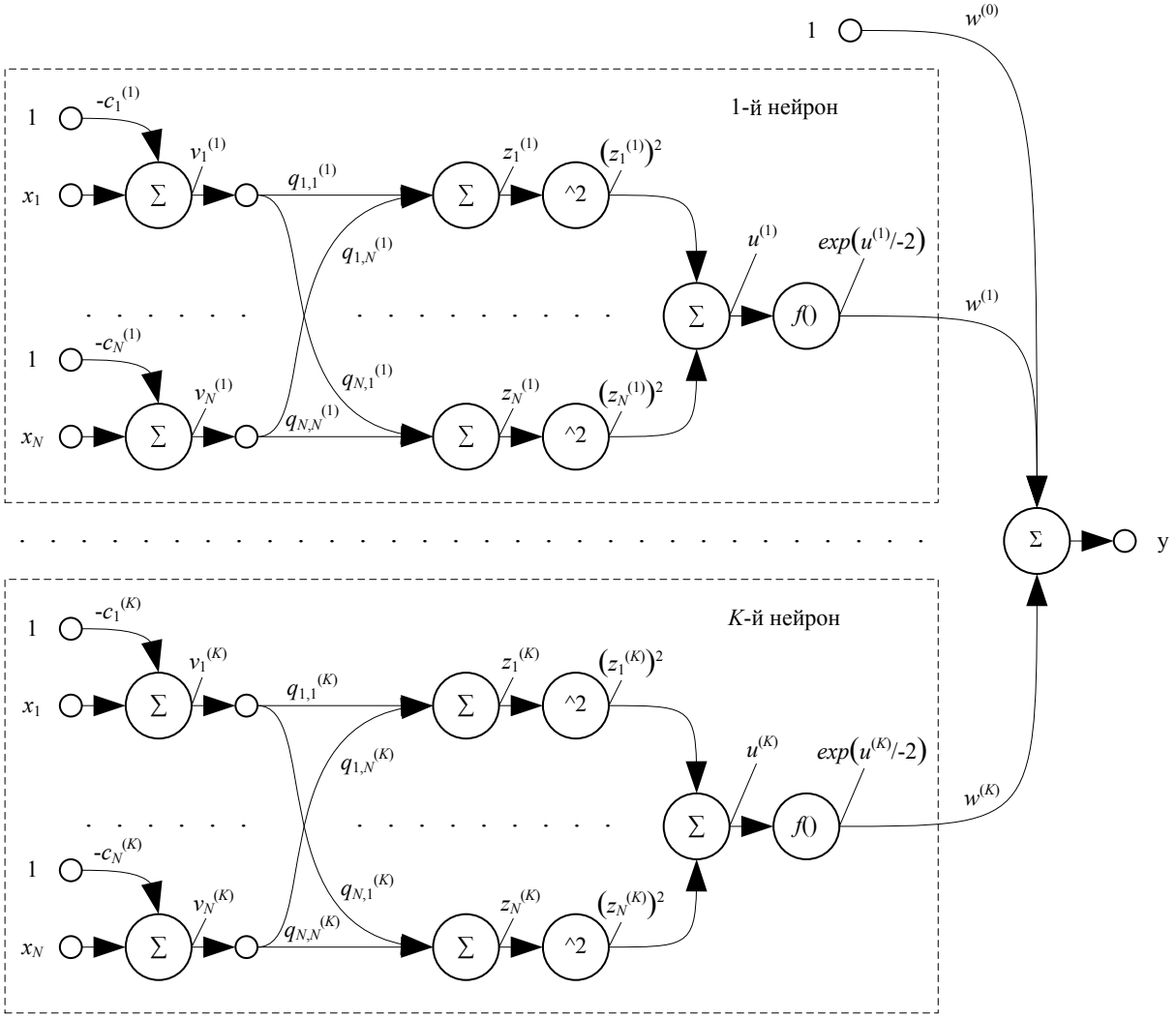


Рисунок 2.4 — Потоковый граф гиперрадиальной нейронной сети

2.2 Разработка алгоритма обучения гиперрадиальной нейронной сети

2.2.1 Принцип обучения

Задача обучения нейронной сети — это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена [79]. Задачу обучения можно разделить на две подзадачи.

а) Установка начальных параметров нейронной сети.

б) Минимизация целевой функции.

Установка начальных параметров, в свою очередь, состоит из следующих процедур.

а) Выбор количества нейронов. Выбор количества нейронов — задача нетривиальная. Она считается основной проблемой, возникающей при корректном решении задачи аппроксимации. Количество нейронов гиперрадиальной нейронной сети либо подбирают эвристически, либо используют метод ортогонализации Грэма-Шмидта [78].

б) Выбор линейных параметров сети (значений весовых коэффициентов выходного слоя). Обычно выбираются случайно из интервала $[-1; 1]$.

в) Выбор нелинейных параметров нейронов (центры $C^{(k)}$ и диаметры $Q^{(k)}$). Их можно выбирать случайным образом или используя алгоритмы самоорганизации.

Целевая функция, задаваемая разработчиком, должна оценивать конфигурацию нейронной сети. Задача минимизации этой функции заключается в модификации свободных параметров нейронной сети и относится к классу оптимизационных задач. Процесс минимизации целевой функции — процесс итеративный, с каждой новой итерацией нейронная сеть все лучше справляется с решаемой ею задачей.

Существует несколько методов минимизации целевой функции.

а) Алгоритмы, основанные на обратном распространении ошибки [78, 80, 81]. Метод обратного распространения ошибки — это способ быстрого расчета градиента. Данный класс алгоритмов использует ошибку на вы-

ходе сети для расчета частных производных по весовым коэффициентам выходного слоя, затем расчет повторяется для нижележащего слоя, и так далее, пока не дойдет до входных элементов сети.

б) Гибридный алгоритм обучения [78, 79]. Для подбора весовых коэффициентов выходного слоя используется метод псевдоинверсии [88], а для нелинейных параметров используются обычные градиентные алгоритмы наискорейшего спуска.

в) Вероятностные алгоритмы [89]. Как следует из названия, данный класс алгоритмов использует вероятностный подбор параметров сети.

2.2.2 Обучение с подбором оптимальной архитектуры

Разработанный автором алгоритм в основе своей содержит гибридный алгоритм обучения. В нем был применен подход к каскадному наращиванию сети. Он заключается в последовательном добавлении нейронов по одному (рис. 2.5), и дообучении нейронной сети. Такую стратегию формирования структуры в сочетании с реализацией алгоритма обучения можно трактовать как выполнение подбора оптимальной архитектуры нейронной сети. Она схожа с алгоритмом формирования сети каскадной корреляции Фальмана [90].

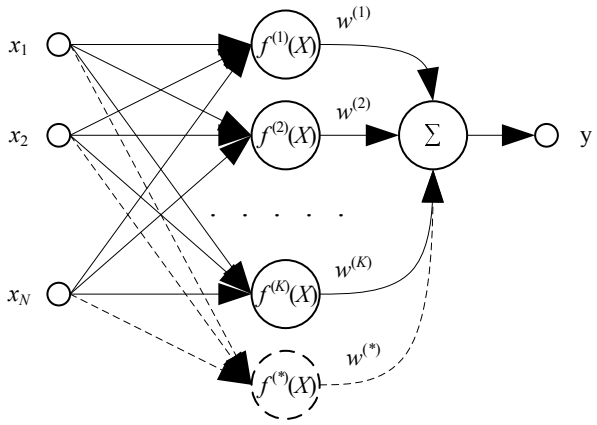


Рисунок 2.5 — Добавление нейрона в нейронную сеть

На начальном этапе сеть состоит из одного единственного нейрона, обученного гибридным алгоритмом. Затем в нейронную сеть добавляется нейрон, начальные параметры которого выбираются случайно. После чего, для этого

нового нейрона, запускается процедура «имитации отжига» [81], в завершении которой выполняется процедура обучения гибридным алгоритмом всей нейронной сети. Если не достигнута заданная погрешность обучения и не превышено максимальное количество нейронов — шаг добавления нейрона в сеть повторяется.

Таким образом, после выполнения такого алгоритма обучения, на выходе получаем гиперрадиальную нейронную сеть с локально-оптимальной архитектурой.

2.2.3 Синтез параллельного алгоритма обучения на основе типового решения «портфель задач»

Разработанный алгоритм обучения нейронной сети основан на модифицированном гибридном алгоритме описанном выше, состоит в использовании метода грубой силы и выполняет обучение и подбор количества нейронов скрытого слоя. Он заключается в итеративном добавлении нейронов, различающихся случайным положением центра, в идентичные копии нейронных сетей и дообучение этих копий. На каждой итерации алгоритма производится выбор лучшей копии сети (по критерию достижения минимального значения целевой функцией) и ее копирование для замены худших копий. Далее итерация добавления повторяется. Критерием остановки алгоритма является либо достижение заданного предельного количества нейронов, либо достижение заданного значения целевой функции.

Параллельный алгоритм обучения реализован на языке *C++* (стандарт *ANSI C++*) с использованием библиотеки *LAPACK* [91]. Он заключается в параллельном выполнении процессов добавления нейронов в идентичные копии нейронных сетей и их дообучение. Применение типового решения «портфель задач» рис. 1.14–1.16 для распараллеливания состоит в определении методов get, put, if_task. В «портфеле задач» для управления вычислениями хранятся:

а) K — текущее количество нейронов в скрытом слое;

б) K_{max} — максимальное число нейронов в скрытом слое;

в) W_1 — число выданных задач, W_2 — решенных задач;

г) W_{max} — максимальное количество задач (попыток добавления нейрона);

д) E_{end} — значение, при достижении целевой функцией которого процесс обучения останавливается;

е) Ψ_K — коэффициенты обученной нейронной сети с K нейронами (включают вектора центров, матрицы диаметров и вектор весов выходного слоя);

ж) Ψ_{K+1} — коэффициенты обученной нейронной сети с $K+1$ нейронами;

з) E_{K+1} — значение целевой функции сети Ψ_{K+1}.

Задача определяется коэффициентами Ψ'_K и положением центра добавляемого нейрона ψ'. Результат ее вычисления — коэффициенты Ψ'_{K+1} и значение ошибки E'_{K+1}. Как и в последовательном случае, при решении задачи производится уточнение коэффициентов сети Ψ'_{K+1} гибридным алгоритмом.

В начальном состоянии ($start$) определяются постоянные алгоритма K_{max}, W_{max}, E_{end}; переменные $W_1 := 0$, $W_2 := 0$, $K := 2$, $\Psi_K := \Psi_2$ (параметры сети с двумя нейронами), Ψ_{K+1} — не определены, $E_{K+1} \gg E_{end}$.

Проверка наличия задачи (if_task). Возможна выборка очередной задачи на обработку, если одновременно выполняются следующие условия. Не достигнута заданная ошибка обучения: $E_{K+1} > E_{end}$; Остались не выданные задачи: $W_1 < W_{max}$.

Алгоритм извлечения задачи (get). Присвоить $\Psi'_K := \Psi_K$; определить случайное положение центра нейрона ψ'. Увеличить счетчик запущенных задач $W_1 := W_1 + 1$.

Алгоритм помещения результата вычисления задачи (put). Если $E'_{K+1} < E_{K+1}$, то определить $E_{K+1} := E'_{K+1}$ и $\Psi_{K+1} := \Psi'_{K+1}$. Присвоить $W_2 = W_2 + 1$. Если $W_2 = W_{max}$, то закончено обучение слоя. В этом случае, если $K < K_{max}$, переходим к следующему слою: $K := K + 1$; $\Psi_K := \Psi_{K+1}$, $W_1 := 0$; $W_2 := 0$.

Алгоритм завершается, когда будут обучены все сети с числом нейронов до K_{max} включительно, либо будет достигнуто заданное значение ошибки обучения $E_{K+1} \leq E_{end}$.

2.2.4 Методика синтеза параллельных алгоритмов с использованием моделей типовых решений

Обобщим описанный способ построения параллельного алгоритма в виде методики синтеза, использующей нисходящую декомпозицию исходного последовательного алгоритма на основе графической модели типовых решений. Методика включает выполнение пяти последовательных шагов.

Шаг 1. Выбор подходящего типового решения.

На данный выбор оказывают влияние следующие факторы. Во-первых, выбор определяется особенностями алгоритма: что более присуще решаемой проблеме — параллелизм по данным или функциональный параллелизм. Например, рассмотренный «метод переменных направлений» допускает оба типа параллелизма. «Портфель задач» и «разветвление-слияние» используют функциональный параллелизм; конвейер — пример параллелизма по данным.

Во-вторых, различают частные и общие типовые решения. Если для распараллеливания подходят несколько решений, то выбор определяется компромиссом между сложностью адаптации исходного кода под выбранное типовое решение и эффективностью распараллеливания. Обычно специализированное типовое решение даёт больший выигрыш в ускорении, но требует больших усилий по адаптации кода алгоритма.

В-третьих, сложность адаптации зависит от того, планируется ли использование систем с распределенной памятью. При использовании распределённой памяти, обычно, требуются большие усилия для адаптации алгоритма под выбранное типовое решение.

«Портфель задач» является универсальным типовым решением, пригодным как для вычислений в общей, так и в распределенной памяти. Он хорошо подходит для оптимизационных и переборных задач. Если в исходном последовательном алгоритме есть участки кода, соответствующие «задачам», выбор следует остановить на данном решении. Очевидной «задачей» в алгоритме обучения нейронной сети является часть, соответствующая гибридному алгоритму уточнения коэффициентов нейронной сети.

Шаг 2. Определение соответствия между элементами графической модели типового решения и исходным последовательным алгоритмом.

Графическое представление типового решения, во-первых, задаёт декомпозицию алгоритма на последовательные процедуры и определяет типы входных и выходных данных, с которыми эти процедуры работают.

Во вторых, графическое представление задаёт информационные зависимости и порядок запуска процедур.

В-третьих, процедуры алгоритма скомпонованы в процессы. Такие процедуры работают с локальными данными процесса.

Уточнение элементов декомпозиции следует начинать с выделения процессов в исходном коде и определения локальных переменных состояния процессов. Например, распараллеливание алгоритма обучения нейронной сети мы начали с описания состояния портфеля и задачи. При удачном соответствии типового решения и исходной задачи (что имеет место для примера обучения нейронной сети и примеров, рассмотренных в главе 3), процедуры имеют несложный, обозримый код.

Шаг 3. Рефакторинг исходного кода последовательного алгоритма.

Рефакторинг — это изменение структуры кода алгоритма с сохранением функциональной и информационной семантики. Для выполнения данного шага берется шаблон кода типового решения. Данный шаблон представляет собой последовательный алгоритм с выделенными участками, в которые пользователю следует добавить фрагменты из исходного распараллеливаемого алгоритма. Эти части содержат объявления пользовательских типов данных и процедуры, идентифицированные на шаге 2.

Заметим, что на данной стадии пользователь оперирует с последовательным кодом. Шаблон кода типового решения не содержит никаких методов коммуникации и синхронизации. Предполагается, что пользователь понимает семантику конструкций языка программирования, на котором реализован исходный последовательный алгоритм.

Более подробно программная реализация шаблонов и состав библиотеки программного комплекса рассмотрены в главе 3. Отметим, что код шаблона типового решения может либо синтезироваться по графическому представлению (в виде *XML*-спецификаций) специальными утилитами, либо представлять собой рукописный повторно используемый код.

После выполнения рефакторинга нужно убедиться в функциональной идентичности исходного и преобразованного кода. Обычно это достигается с

использованием процедуры регрессионного тестирования. То есть тесты, которые успешно проходила исходная программа, должны быть также успешно пройдены преобразованной программой.

Шаг 4. Автоматическое распараллеливание.

Данный шаг является автоматизированным. Требуется лишь выбрать отображение алгоритма на параллельную архитектуру из возможных вариантов для данного типового решения, имеющихся в библиотеке программного комплекса.

В силу универсальности, для типового решения «портфель задач» доступны эффективные отображения на архитектуры *Windows* и *POSIX* в общей памяти и на архитектуру *MPI* в распределённой памяти.

Шаг 5. Тестирование производительности.

Требуется определить, достигается ли желаемый выигрыш в скорости выполнения программы. Для этого описываемые в диссертации реализации типовых решений содержат код замера времени выполнения, а специальная *web*-система *Templet Web* (`http://templet.ssau.ru`) позволяет автоматически выполнить запуск параллельной программы на суперкомпьютере «Сергей Королёв» и на тестовых окружениях.

При достижении желаемого ускорения далее проводятся вычислительные эксперименты. Если желаемое ускорение не достигнуто, необходимо проверить следующие возможности. Если тестировалось отображение на распределенную архитектуру *MPI*, рекомендуется повторить шаги 4 и 5 в общей памяти для архитектуры *POSIX* на, по-возможности, большем числе процессоров. При достижении значимого ускорения можно попробовать вернуться к распределённому варианту отображения, задав меньшее количество узлов.

Также следует увеличить размерность решаемой задачи. При проведении тестирования производительности необходимо учитывать, что значимое сокращение времени счёта удаётся получить для программ, исходное время счёта которых составляет минуту и более. Предлагаемая технология распараллеливания предназначена для уменьшения исходного времени счёта с порядка одного часа, до времени счёта порядка одной минуты.

Если не удалось достичь желаемого ускорения, возможно требуется распараллеливание, учитывающее тонкие особенности алгоритма, и техника типовых решений не подходит. Данная методика ориентирована на инкремент-

ное распараллеливание, затрагивающее незначительное изменение структуры кода, возможно добавление небольшого объёма связывающего кода, но не предусматривает переписывания исходного алгоритма. Цель методики — сокращение общего времени распараллеливания и проведения вычислительных экспериментов, а не получение максимального ускорения при распараллеливании алгоритма. Однако, как показано в приведённых результатах вычислительных экспериментов, за счёт учёта особенностей вычислительных процессов, описываемых типовыми решениями, удаётся достичь высокой эффективности распараллеливания.

2.3 Экспериментальное исследование алгоритма обучения

Эксперимент по обучению гиперрадиальной нейронной сети параллельным алгоритмом на основе типового решения «портфель задач» был проведен на кластере «Сергей Королёв» (`http://hpc.ssau.ru/`). Запуск задач производился на узлах 3-х типов с 3-мя различными типами процессоров соответственно.

В рамках данного эксперимента на быстродействие влияло количество доступных вычислительных ядер на узле. Остальные параметры ощутимого влияния не оказывали, что подтверждается экспериментальными результатами.

Данные для обучения нейронной сети, построения прогноза и анализа алгоритма были взяты из открытой базы данных информационного агентства «РосБизнесКонсалтинг» (`http://export.rbc.ru/`). Использовались дневные курсы валют за период 2010–2013 годов. Условия экспериментов состояли в следующем.

а) Варьировалось количество логических процессов P «портфеля задач» (объявлено в конкретном эксперименте). Количество логических процессов — это параметр, влияющий на масштабируемость параллельного алгоритма.

б) Предельное количество нейронов K_{max} во всех экспериментах равнялось 100.

в) Количество попыток добавления нейронов W_{max} во всех экспериментах

равнялось 100.

Для осуществления сравнительного анализа были произведены замеры времени исполнения последовательного и параллельного алгоритмов. Для минимизации воздействия сторонней динамической нагрузки процессоров на измеряемое время проводилось по 20 запусков алгоритма с фиксированными параметрами. При этом, полученное время исполнения параллельного алгоритма $t_{pr_i}(P)$ было усреднено:

$$t_{pr}(P) = \frac{\sum_{i=1}^{20} t_{pr_i}(P)}{20}. \tag{2.17}$$

Для времени исполнения последовательного алгоритма t_{sq_i}, было найдено минимальное значение $t_{sq_{min}} = \min_{i=\overline{1,20}} t_{sq_i}$. И на его основе рассчитано эталонное время исполнения параллельного алгоритма

$$t_{std}(P) = \frac{t_{sq_{min}}}{P}. \tag{2.18}$$

Средне-квадратичное отклонение рассчитывалось для $t_{pr_i}(P)$.

На узлах с процессорами *Intel®Xeon®X5560* количество логических процессов изменялось в интервале $P = \overline{1,20}$. Данные, полученные при проведении эксперимента, приведены на рис. 2.6, соответствующее средне-квадратичное отклонение изображено на рис. 2.7.

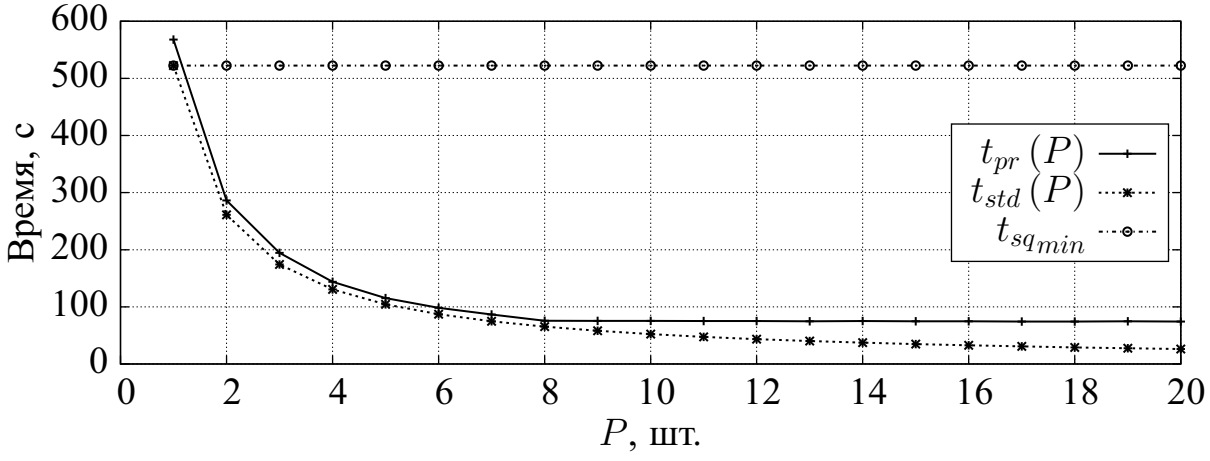


Рисунок 2.6 — Зависимость времени исполнения от количества логических процессов P (8-ядер)

На узлах с процессорами *Intel®Xeon®X5570* количество логических процессов изменялось в интервале $P = \overline{1,30}$. Данные, полученные при прове-

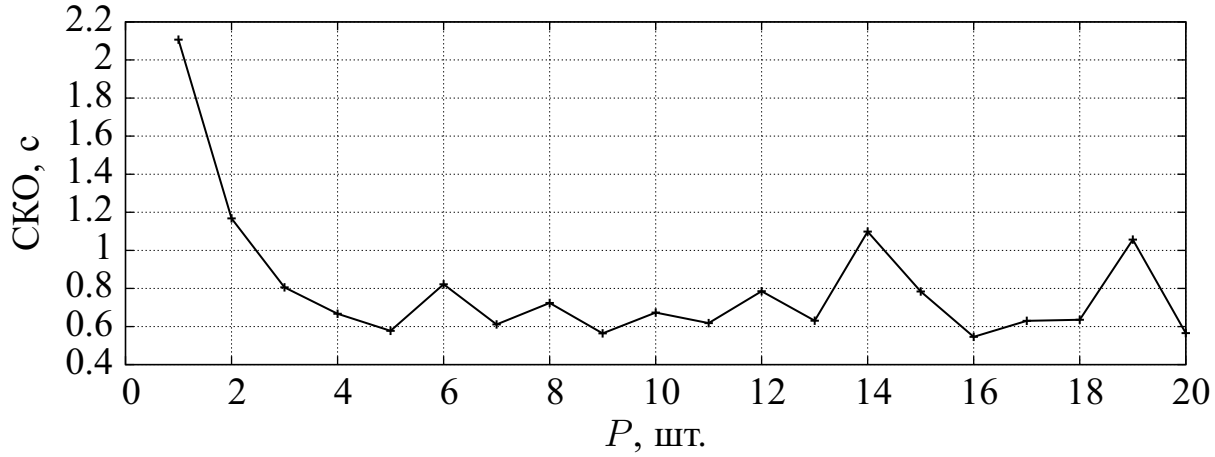


Рисунок 2.7 — Зависимость СКО от количества логических процессов P (8-ядер)

дении эксперимента, приведены на рис. 2.8, соответствующее средне-квадратичное отклонение изображено на рис. 2.9.

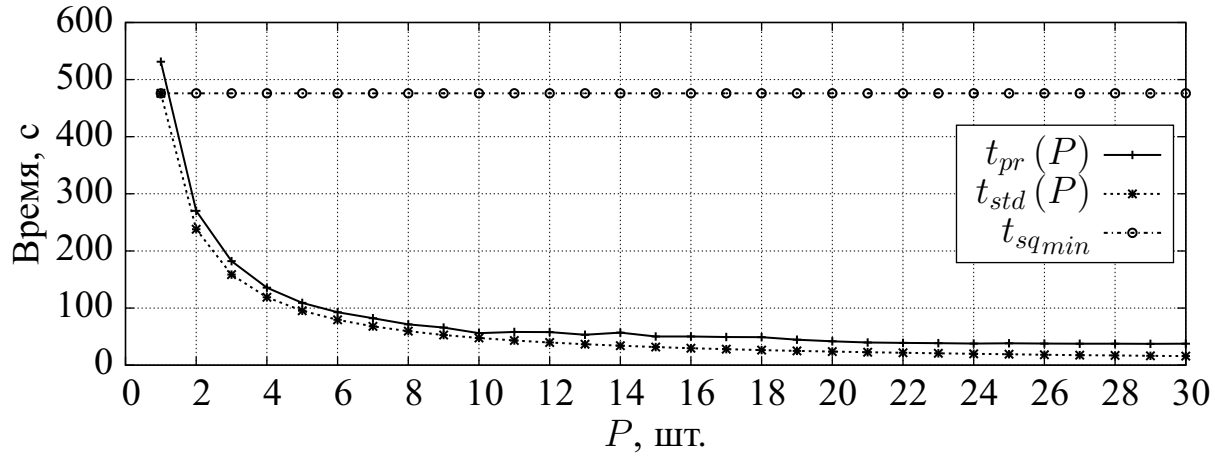


Рисунок 2.8 — Зависимость времени исполнения от количества логических процессов P (12-ядер)

На узлах с процессорами *Intel®Xeon®E5-2665* количество логических процессов изменялось в интервале $P = \overline{1, 40}$. Данные, полученные при проведении эксперимента, приведены на рис. 2.10, соответствующее средне-квадратичное отклонение изображено на рис. 2.11.

Анализируя графики, изображенные на рис. 2.6–2.11, можно сделать вывод, что усредненное время исполнения параллельного алгоритма (2.17), разработанного с использованием типового решения «портфель задач», лишь немного больше эталонного времени исполнения параллельного алгоритма

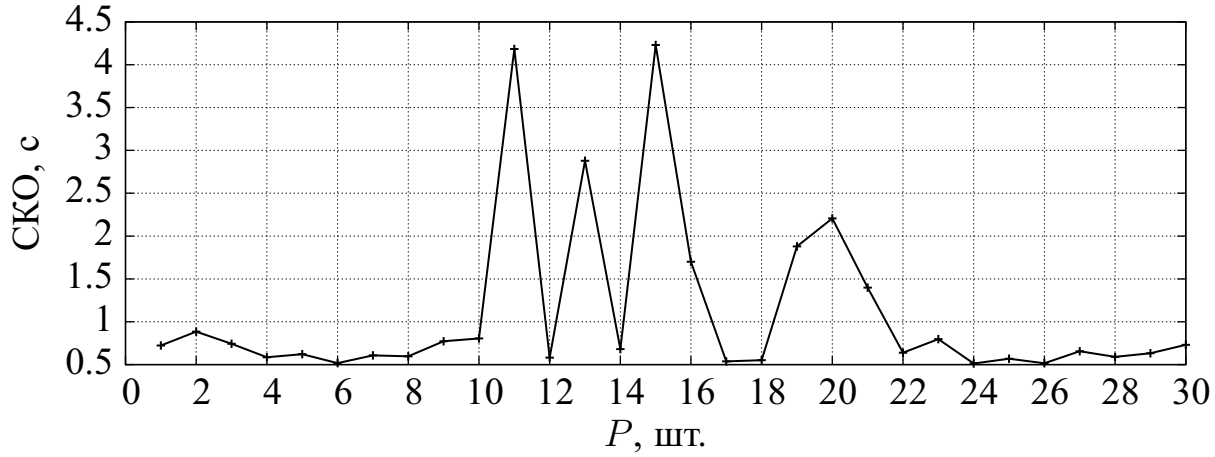


Рисунок 2.9 — Зависимость СКО от количества логических процессов P (12-ядер)

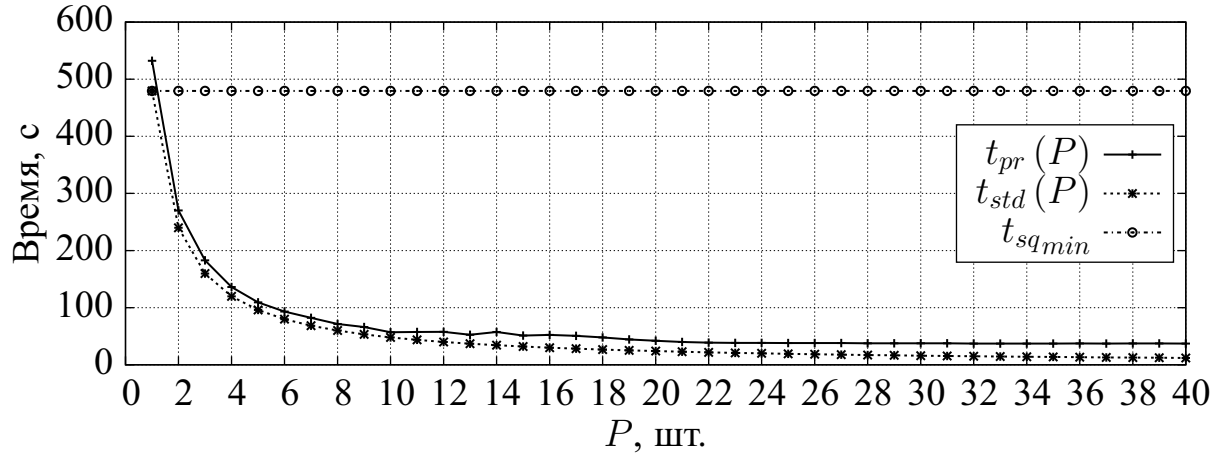


Рисунок 2.10 — Зависимость времени исполнения от количества логических процессов P (16-ядер)

(2.18).

Также можно заметить, что при достижении количеством логических процессов P значения приблизительно равного удвоенному количеству физических вычислительных ядер, уменьшение времени исполнения параллельного алгоритма практически прекращается. Это объясняется наличием технологии *Hyper-threading* в процессорах (удвоенное количество вычислительных потоков по отношению к количеству ядер). Если продолжать увеличивать количество логических процессов, то в связи с ростом накладных расходов время исполнения начнёт возрастать. Следовательно, число логических процессов равное количеству вычислительных потоков можно считать оптимальным,

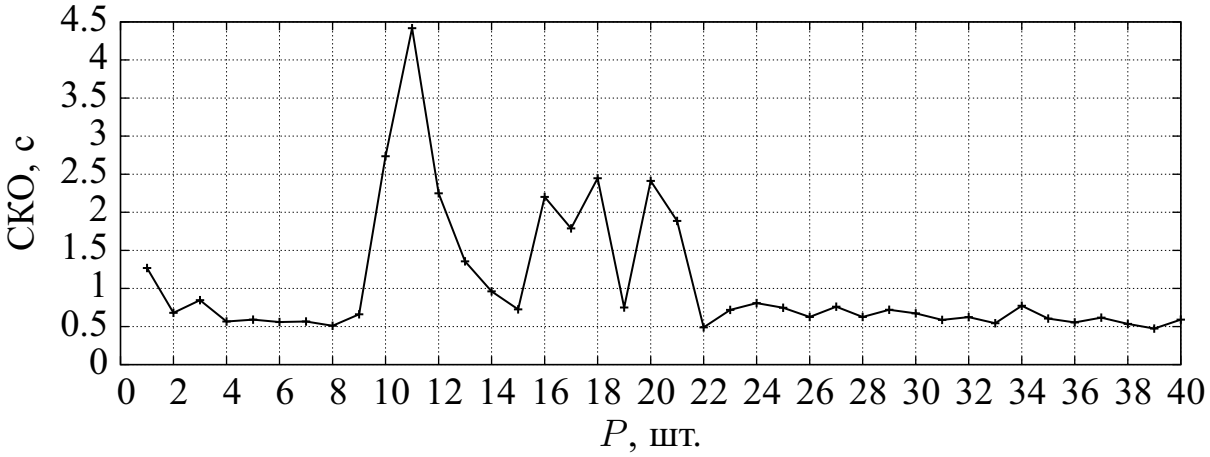


Рисунок 2.11 — Зависимость СКО от количества логических процессов P (16-ядер)

и рекомендовать для использования в экспериментах с применением типового решения «портфель задач».

Отметим, что данный алгоритм является распределенным по построению и может исполняться как в системах с разделяемой, так и в системах с распределенной памятью. А техника его распараллеливания является универсальной и может применяться для решения подобных задач с использованием типового решения «портфель задач».

2.4 Выводы

а) Показано, что методы прогнозирования на основе радиально-базисных нейронных сетей являются наиболее простыми в использовании.

б) Рассмотрены методы обучения гиперрадиальных нейронных сетей. Показано, что основными проблемами обучения являются высокая трудоемкость и необходимость подбора оптимальной структуры сети.

в) Предложена методика синтеза параллельного алгоритма на примере алгоритма обучения гиперрадиальной нейронной сети с использованием модели типового решения «портфель задач».

г) На основе типового решения «портфель задач» синтезирован параллельный алгоритм обучения гиперрадиальной нейронной сети, позволяющий уменьшить время обучения с использованием параллельной *ЭВМ* и решающий проблему подбора оптимальной структуры сети.

д) В вычислительных экспериментах с разработанным параллельным алгоритмом показано, что сокращения времени вычисления составило: на 8-ми ядерной (16 потоков) системе примерно в 7 раз; на 12-ти ядерной (24 потока) системе примерно в 12 раз; на 16-ти ядерной (32 потока) системе примерно в 13 раз.

ГЛАВА 3. ПОСТРОЕНИЕ ПРОГРАММНОГО КОМПЛЕКСА ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ НА ОСНОВЕ ТИПОВЫХ РЕШЕНИЙ И ОПИСАНИЕ ЕГО ПРИМЕНЕНИЙ

В первой части третьей главы дается обзор программного обеспечения параллельных вычислений. В обзоре показано, какие средства параллельного программирования пригодны для реализации типовых решений.

Во второй части главы рассматривается предложенная архитектура программного комплекса для применения типовых решений в численном моделировании. Описывается структура библиотеки типовых решений, реализующая «портфель задач» и другие типовые решения.

В третьей части главы показано применение программного комплекса на примерах распараллеливания программ численного моделирования, предоставленных авторами работ [18, 92] из разных предметных областей [93–95]. Рассмотрено применение моделей типовых решений «метод переменных направлений», «конвейер», «портфель задач», реализованных в библиотеках программного комплекса, в задачах исследования нелинейных динамических процессов, распространении плоской световой волны, моделирования фазовых траекторий динамических систем.

Результаты главы опубликованы в работах [17, 18, 21].

3.1 Обзор программного обеспечения параллельных вычислений

3.1.1 Прикладные интерфейсы операционных систем

Параллельное приложение для мультипроцессорного компьютера можно написать, воспользовавшись средствами интерфейса прикладного программирования (*API*) операционной системы. Такой выбор обусловлен желанием максимально оптимизировать код программы и упростить интеграцию вычислительной части приложения с компонентами пользовательского интерфейса. Необходимость данного метода программирования вызвана также тем, что современные персональные компьютеры и рабочие станции требуют поддержки симметричной мультипроцессорной обработки для эффективного исполнения (многоядерные процессоры и технология *Hyper-threading technology*).

В функциональном смысле интерфейсы разных операционных систем похожи, так как основаны на результатах теории синхронизации процессов, разработанной в 1960–1980 годах. Рассмотрим средства мультипроцессорной обработки *Windows API* [96], применяемые в операционных системах семейства *Windows* [97].

Мультипроцессорная обработка в *Windows API* может быть реализована двумя способами: на уровне процессов и на уровне потоков (или нитей) исполнения. Первый способ позволяет изолировать адресные пространства компонентов приложения, что повышает надежность. Второй способ позволяет более эффективно реализовать взаимодействие через разделяемую память. Особенностью межпроцессного взаимодействия в *Windows API* является использование семантики доступа к файлам. Обмен информацией между процессами реализуется как чтение или запись в файл с последовательным доступом. Межпроцессное взаимодействие осуществляется с использованием почтовых ящиков (*mail slot*), каналов (*pipe*), файлов, отображенных на память, и сокетов.

Для синхронизации имеется набор примитивов, каждый из которых эффективен в том или ином случае. К числу таких примитивов относятся функции спин-блокировок, критическая секция, мьютекс, семафор и событие. Особенностью реализации примитивов *Windows API* является использование объектно-ориентированного подхода, вследствие чего, ко всем объектам синхронизации, которые могут находиться в сигнальном состоянии, применяются одинаковые функции синхронизации. Это упрощает *API* и позволяет организовывать точки объединения подпроцессов (нитей) без введения дополнительных функций.

Мультипроцессорная обработка может осуществляться с использованием механизмов асинхронного программирования, например, асинхронного вызова процедур. В интерфейсе *Windows API* имеются также методы изменения стандартной стратегии планирования потоков, влияющие на приоритеты планирования и на способ выбора процессора для исполнения нити. При необходимости можно реализовать собственную стратегию планирования с использованием волокон (*fiber*). Общее число функций синхронизации составляет более 70. В целом можно заключить, что *Windows API* содержит развитые средства синхронизации, позволяющие выполнить тонкую оптими-

зацию мультипроцессорного приложения для операционных систем семейства *Windows*.

Проблемой при использовании *Windows API* является сложность переноса кода приложений на другие операционные системы. Однако, если не рассматривается параллелизм на уровне программных инструкций «можно организовать поддержку параллелизма средствами библиотек, которые будут приближаться к встроенным средствам параллелизма, как по эффективности, так и по удобству применения» [98]. На данном эмпирическом факте основаны работы по стандартизации *API* средств, выполняемые в рамках стандарта *POSIX* [99] (*Portable Operation System Interface*), который в сочетании с языковыми средствами *C++* применяется для программирования параллелизма. Функции мультипроцессорной обработки являются частью единого стандарта спецификаций *UNIX* (*Single UNIX Specifications Standart*) *IEEE Std.1003.1-2013*. Данный стандарт преследует цель переносимости кода приложения и представляет разработчикам *ПО* единый набор *API*-функций, которые должны поддерживаться каждой *UNIX*-системой.

Аналогично *Windows* параллелизм в *UNIX* может реализовываться как на уровне процессов, так и на уровне потоков. Для работы с процессами имеется функция *spawn* и семейство *exec* функций. Для работы с потоками используют функции библиотеки *POSIX Threads*.

В качестве особенностей работы с процессами можно выделить отличающуюся от *Windows API* семантику порождения дочерних процессов, когда новый процесс исполняет тот же самый код, что и родительский процесс, с точки ветвления. При этом порожденный процесс наследует дескрипторы некоторых объектов (например, файлов), но содержит индивидуальные значения переменных. Также процессы имеют средства работы с сигналами *UNIX*.

Нити *POSIX* могут синхронизироваться мьютексами, семафорами и условными переменными. К особенностям использования мьютексов можно отнести наличие блокировок для чтения и записи. Условные переменные являются аналогом событий в *Windows API*, однако они используются совместно с мьютексом. Для объединения потоков имеются специальные функции. Расширенная семантика реализована в планировщике потоков путем определения областей конкуренции.

Благодаря похожей семантике многопоточных операционных систем,

объектно-ориентированные приложения можно сделать переносимыми путем создания интерфейсных классов для примитивов синхронизации. Также возможен непосредственный запуск *POSIX* приложений в версиях *Windows* на ядре *NT*.

3.1.2 Стандартные библиотеки и библиотеки времени исполнения языков программирования

Стандарт *OpenMP*. Альтернативный подход к разработке сред программирования для мультипроцессорного компьютера основан на стандартизации конструкций, встраиваемых в компиляторы последовательных языков. Примером такого подхода является стандарт *OpenMP* [100].

Стандарт *OpenMP* основан на простой модели потока управления, использующей операторы ветвления и объединения процессов. Оператор ветвления порождает группу параллельных процессов, за которым должен следовать оператор объединения. После объединения продолжает исполнение только один исходный процесс. Каждый подпроцесс далее может подобным образом порождать новые подпроцессы. Другим принципом стандарта является обеспечение условной компиляции для встраиваемых в язык конструкций и функций управления.

Стандарт регламентирует следующие элементы среды разработки: управляющие структуры, окружения данных, синхропримитивы и библиотеку времени исполнения.

Основной управляющей структурой является рассмотренная конструкция для порождения параллельных процессов. Примерами других структур являются критические секции; секции, в которые может входить только один процесс из группы; секции, в которые входит только главный процесс из группы; секции, задающие порядок их прохождения процессами из группы и так далее.

Окружением данных в *OpenMP* называют переменные, определяющие контекст исполнения процесса. Когда порождается новый процесс можно при помощи атрибутов определить, какие переменные будут разделяться с головным процессом, процессами той же группы, а также не будут разделяться с

другими процессами. При помощи атрибутов можно задавать различные варианты разделения переменных.

Синхронизация в *OpenMP* может быть явной и неявной. Примером неявной синхронизации являются управляющие структуры. К явным средствам относятся специальные синхропримитивы атомарных операторов, операторов синхронизации состояний переменных, барьерной синхронизации.

В дополнение к встраиваемым в язык директивам стандарт предусматривает набор функций времени исполнения. Эти функции используются для управления режимами работы приложения. Возможно задание динамического порождения процессов, запрет или разрешение вложенного параллелизма.

Использованный в *OpenMP* подход регламентирует средства синхронизации, а также задает модель исполнения и структуру кода приложения. При этом приложение может кодироваться более компактно, но за счет потери гибкости программирования. Модель хорошо подходит для вычислительных приложений, содержащих распараллеливаемые циклы.

Библиотеки стандарта *MPI* и *PVM*. Интерфейс передачи сообщений *MPI* представляет собой набор из около 300 функций управления параллельными вычислениями на основе семантики передачи сообщений [101]. К числу основных понятий модели *MPI* относятся процесс, группа процессов и коммуникатор.

Все процессы исполняют код одной и той же программы (модель *SPMD*), но имеют уникальные идентификаторы в группе. Обычно все процессы порождаются одновременно, хотя версии 2 и 3 стандарта оговаривают механизм динамического порождения процессов. Группа представляет собой совокупность процессов, каждый из которых имеет внутри группы уникальное имя и взаимодействует с другими процессами через коммуникатор группы. Коммуникатор служит коммуникационной средой взаимодействия и используется для синхронизации. Коммуникаторы бывают внутригрупповые и межгрупповые. Каждый коммуникатор имеет собственное коммуникационное пространство. Сообщения, отправленные через разные коммуникаторы, не оказывают влияния друг на друга. В *MPI* имеются функции управления памятью для буферизации сообщений и хранения внутренних объектов (например, коммуникаторов). Гарантируется, что программы, написанные с использованием *MPI*, будут без настройки выполняться на любом компьютере. Для этого сама

программа содержит код инициализации, в котором определяется исходный коммуникатор, включающий все запущенные процессы.

Автоматизация при использовании *MPI* заключается в том, что библиотека обеспечивает гарантированную доставку сообщения адресату, автоматическую буферизацию на стороне отправителя и получателя сообщения, а также выполнение групповых обменов сообщениями с учетом различных топологий коммуникационных графов. Групповые операции включают барьерную синхронизацию, широковещательную передачу одного сообщения, сбор данных из всех процессов группы в один процесс, сбор данных из всех процессов во все процессы.

Аппарат коммуникаторов позволяет разрабатывать библиотеки параллельных программ, которые могут представлять собой либо специализированные коммуникаторы, либо специализированные подпрограммы, связанные через внутригрупповой коммуникатор.

Библиотека *PVM* (*Parallel Virtual Machine*) [102] является еще одним стандартным средством организации параллельных вычислений на основе модели передачи сообщений. В библиотеке реализована надежная доставка сообщений типа точка-точка и групповая доставка сообщений. Особенностью *PVM* являются средства динамического порождения программ, работа в гетерогенном окружении и средства для написания отказоустойчивых приложений. Приложение в общем случае соответствует модели *MPMD*, причем разные программы, образующие приложение, могут быть скомпилированы для разных платформ. В этом случае среда исполнения выполняет преобразование передаваемых сообщений с использованием формата *XDR* [103].

Обе библиотеки предоставляют средства для создания переносимого кода. Однако уровень автоматизации программирования достаточно низкий, что требует высокой квалификации и усложняет программирование. При написании кода с использованием этих библиотек для повышения уровня программирования рекомендуется инкапсулировать код *MPI/PVM* внутри объектов [104].

Языковые средства. Для написания параллельных и распределенных программ могут использоваться языки программирования общего назначения либо специализированные языки.

В языках программирования общего назначения средствами задания параллелизма являются библиотеки или препроцессорные директивы. Наибо-

лее распространенными языками общего назначения, используемыми для написания вычислительных программ в исследовательских целях, в настоящее время являются языки *C* и *Fortran*. Стандартным способами задания параллелизма в языках *C* и *Fortran* являются библиотеки *MPI* [101] и директивы *OpenMP* [100], причем они могут применяться совместно. Данный подход имеет следующие достоинства. Обеспечивается переносимость программ, так как компиляторы языков *C* и *Fortran* доступны на любой вычислительной программно-аппаратной платформе. В связи с тем, что эти языки используются достаточно давно, для них разработано большое количество библиотек численных методов (как последовательных, так и параллельных), которые можно применять в более сложных специализированных моделях. Также важно, что языки *C* и *Fortran* связаны с объектно-ориентированными языками *C++* [98] и *Fortran 90* [105]. В большинстве случаев процедурные библиотеки переносимы на уровне исходных кодов из *C* в *C++* и из ранних версий языка *Fortran* в объектно-ориентированные версии. Использование объектной парадигмы позволяет повысить уровень абстрагирования, что может потребоваться при написании сложных многокомпонентных и, в том числе, распределенных моделей. Также это упрощает интеграцию с другими подсистемами и приложениями, например, подсистемой визуализации результатов моделирования. Кроме этого, последние версии стандарта *C++ ISO/IEC 14882:2014*содержат средства параллельного программирования. Другие последовательные языки программирования общего назначения (*Pascal*) применяются, как правило, при создании несложных численных моделей.

Проблемой использования языков *C*, *Fortran* и средств на основе *MPI* или *OpenMP* является сложность программирования, как алгоритмов, так и кода для управления вычислительным процессом, ввода исходных данных и представления результатов моделирования. С целью упрощения программирования параллельных и распределенных вычислений создаются специализированные языки. Они могут реализовываться как полностью автономные проекты [106] или являться препроцессорами широко используемых языков, например *C++* [107,108]. К особой группе относятся встроенные языки специализированных математических пакетов (*Math Lab*, *MathCAD*), с помощью которых возможно организовать параллельные вычисления, ввод данных и визуализацию результатов. Языки могут использовать как явную [109], так и неявную

форму [34] представления параллелизма и синхронизации, быть ориентированными на распределенную [110] или параллельную обработку [111]. Язык может использоваться автономно [112] или требовать определение низкоуровневых операций на другом языке [113] как, например, *IDL CORBA* [114]. Среди специализированных языков можно выделить языки, определяющие вычислительную модель в целом (*T-технология*) [59] и накладывающие ограничения (или предлагающие специальные) на методы коммуникации и синхронизации [115].

В промышленных разработках в области параллельного и распределенного программирования часто используются платформо-независимые среды с динамической компоновкой *Java* от *ORACLE* [116] и *.NET* от *Microsoft* [117]. Ключевыми для распределенного программирования в данных технологиях являются следующие возможности:

а) механизмы прозрачного перемещения, как состояния объектов, так и их кода с одной машины на другую;
б) модели безопасности, позволяющие организовать контроль использования ресурсов;
в) возможность генерации кода заглушек *RPC* «налету», устраняющая необходимость в использовании сторонних технологий на основе *IDL*;
г) динамическая генерация целевого кода из переносимого при загрузке классов, повышающая эффективность исполнения.

Еще одним направлением в области промышленного программирования является использование *XML* [118] как основы для разработки предметно-ориентированных языков, в том числе и для распределенного и параллельного программирования. Данный подход позволяет автоматизировать лексический, синтаксический и частично семантический анализ предметно-ориентированного языка. Кроме этого, подход к проектированию языка на основе *XML* позволяет воспользоваться готовым визуальным редактором или разработать собственный редактор языка.

Исходя из обзора средств разработки параллельных алгоритмов п. 3.1.1 и п. 3.1.2, можно сделать следующие выводы.

а) Для систем, ориентированных на *UNIX*, которые составляют большую часть парка высокопроизводительных систем, предпочтение в реализации типовых решений в общей (разделяемой) памяти следует отдать ин-

терфейсу *POSIX Threads*.

б) Библиотеки стандарта *OpenMP* непригодны в силу его преимущественной ориентации на приложения с параллельными циклами. Как показано в главе 1, типовые решения имеют более сложную структуру.

в) Для реализации типовых решений в системах *Windows* с целью потенциально более высокой производительности следует использовать непосредственно *API* операционной системы *Windows*. Данный программный интерфейс имеет более гибкие средства программирования по сравнению с другими средствами на этой операционной системе.

г) Для исполнения в распределенной памяти в первую очередь следует использовать библиотеки стандарта *MPI*. Этот выбор связан с распространенностью библиотек как на суперкомпьютерах, так и на кластерных системах.

3.1.3 Средства управления параллельными вычислениями

Современными средствами управления параллельными вычислениями являются *Grid*-системы и облачные технологии.

В работе [119] термин «*Grid*» определяется как аппаратно-программная инфраструктура, которая обеспечивает надежный, устойчивый, повсеместный и недорогой доступ к высокопроизводительным компьютерным ресурсам. С учетом социальных и политических аспектов функционирования *Grid* Фостер, Кессельман и Тьюке [120] развивают данное определение. *Grid*-компьютинг — это скоординированное разделение ресурсов и решение задач в динамически меняющихся виртуальных организациях со многими участниками. Ключевая концепция состоит в достижении договоренности о разделении ресурсов между поставщиками и потребителями и использование полученного пула ресурсов для различных целей. Отмечается, что упомянутое разделение ресурсов заключается не просто в обмене файлами, а в прямом доступе к компьютерам, программному обеспечению, данным и другим ресурсам. В работе [120] также говорится о важности стандартных протоколов как о средстве, которое обеспечивает взаимодействие и общность инфраструктуры.

Приведенные выше определения можно представить в виде списка из

трех критериев, выделяющих *Grid*-систему среди других распределенных систем.

Grid-система координирует использование ресурсов при отсутствии централизованного управления этими ресурсами. *Grid* интегрирует и координирует ресурсы и пользователей, которые находятся в разных местах (например, персональный компьютер пользователя и центральный компьютер, компьютеры нескольких компаний или административных отделений одной компании), и направляет участникам уведомления о гарантиях, страховке, платежах, членстве и т.д. В противном случае система является локальной системой управления.

Grid-система использует стандартные, открытые, универсальные протоколы и интерфейсы. *Grid* строится на базе многоцелевых протоколов и интерфейсов, позволяющих решать такие фундаментальные задачи как аутентификация, авторизация, обнаружение ресурсов и доступ к ресурсам. В противном случае мы имеем дело со специализированной прикладной системой.

Grid-система должна нетривиальным образом обеспечивать высококачественное обслуживание. *Grid* позволяет использовать входящие в его состав ресурсы таким образом, чтобы обеспечивалось высокое качество обслуживания. Это касается, например, таких параметров, как время отклика, пропускная способность, доступность и надежность. Совместное использование ресурсов различных типов делает выгоду от использования комбинированной системы значительно выше, чем от суммы ее отдельных частей.

Разработка *Grid*-систем является одним из ведущих направлений исследовании в области распределенных вычислений. Другими подходами являются метакомпьютинг, использование вычислительных кластеров, пиринговые вычисления (*P2P*) и вычисления в сети Интернет.

Под метакомпьютингом [121] обычно понимается направление, разрабатываемое в начале 1990 годов, заключающееся в объединении суперкомпьютеров *США* высокоскоростными сетями передачи данных. Ключевыми проектами данного направления являются проект *I-WAY* и проект *FAFNER* (*Factoring via Network-Enabled Recursion*).

Цель проекта *I-WAY* [122] заключалась в объединении суперкомпьютеров с использованием только существующих каналов связи. Одной из инноваций *I-WAY* являлась разработка концепции брокера ресурсов. Этот проект

— предшественник современных *Grid*-систем. Проект *FAFNER* представлял собой решение вычислительно трудной задачи факторизации больших чисел. Принципы организации вычислений проекта *FAFNER* в дальнейшем использовались во многих подобных проектах.

Кластерные вычисления представляют собой альтернативу мейнфреймам и суперкомпьютерам. Кластеры собираются из стандартных компонентов. Первый кластер такого типа назывался *Beowulf* [123]. Сейчас многие коммерческие фирмы поставляют готовые кластерные системы.

Примером пиринговых вычислений является система *Napster*. Эта файлообменная система объединяла произвольные компьютеры с подключением к Интернет (при установке специального программного обеспечения). Несмотря на использование центрального сервера, обмен данными между клиентами сети происходил напрямую.

Вычисления в Интернет или так называемые *@home* проекты являются концептуальными последователями системы *FAFNER* и используют вычислительные мощности простаивающих компьютеров, подключенных к Интернет. Для таких приложений характерен низкий объем трафика. Они реализуются в виде программ «хранителей экранов». Данный способ является наиболее простым при организации мультисайтовых распределенных вычислений.

С точки зрения удобства предоставления доступа к вычислительным ресурсам в настоящее время преимущества имеют облачные технологии. Управление запуском приложений в *Grid*-системах требует специальных навыков, что противоречит концепции удобства использования типовых решений специалистом прикладной области. Кроме этого *Grid*-системы также, как и пакетные системы отдельных суперкомпьютеров, предусматривают сложный регламент оформления доступа. Это вносит дополнительные неудобства и сдерживает применение предлагаемой в работе технологии для разовых расчетов. Поэтому альтернативой *Grid*-технологии для организации доступа к высокопроизводительной системе является «облачный» сервис типа *PaaS* (платформа как сервис) [124].

3.2 Архитектура программного комплекса

3.2.1 Принципы реализации программного комплекса

Для развертывания системы типовых решений автором выбран сервис *PaaS*-сервис *Templet Web* [125]. В исходной конфигурации сервис *Templet Web* позволяет развертывать и запускать произвольные приложения. В диссертации были разработаны пакеты программ, позволяющие запускать типовые решения «портфель задач», «конвейер» и другие способом, удобным для специалиста предметной области. Дальнейшее описание архитектуры ведется на примере типового решения «портфель задач», использованного для реализации параллельного алгоритма обучения нейронной сети. Исходный код типового решения и алгоритма обучения нейронной сети приведен в приложении. Код зарегистрирован в Роспатенте [20].

Сервис *Templet Web* (`http://templet.ssau.ru`) предназначен для коллективной разработки приложений научных вычислений на базе суперкомпьютерного центра Самарского государственного аэрокосмического университета имени академика С.П. Королёва.

Сервис имеет следующие функциональные возможности. Позволяет установить разработанные типовые решения («портфель задач», «конвейер» и другие) для использования в прикладных проектах высокопроизводительных приложений. Сервис позволяет организовать командную работу над проектом с применением системы контроля версий *Subversion*. Имеется возможность доступа к общим окружениям развертывания (вычислительным системам по протоколу *SSH*) и запущенным задачам. Одно и тоже приложение можно запустить как в тестовом, так и в целевом окружении. Во время продолжительных вычислений сервис выполняет отслеживание работы приложения и уведомляет о статусе задач в целевых окружениях.

В сервисе поддерживаются несколько типов окружения: окружение *Linux*, суперкомпьютер «Сергей Королев».

В качестве окружения *Linux* могут использоваться любые настроенные компьютеры пользователя с доступом из сети интернет по протоколу *SSH*. Имеется тестовое окружение *Linux*, развернутое на виртуальной машине су-

перкомпьютерного центра *СГАУ*. Рабочими окружениями являются окружения суперкомпьютера «Сергей Королев» *СГАУ*. Для их настройки используется учетная запись, предоставленная суперкомпьютерным центром *СГАУ*, при этом настраивать подключение *SSH* не требуется. После выполнения настроек дальнейшее взаимодействие с сервисом выполняется через *web*-интерфейс.

Сервис *Templet Web*, включающий набор схем вычислительных методов, упрощает создание параллельного приложения. Инструментарий позволяет запускать приложение в нескольких режимах: отладка, эмуляция, приложение *Windows*, *POSIX* приложение, распределённое *MPI* приложение. В режиме эмуляции существует возможность оценить максимальное ускорение (оптимистичную оценку) для большого числа процессоров без реального запуска на суперкомпьютере. Далее показано как реализуются эти возможности.

3.2.2 Описание библиотеки типовых решений

Состав библиотеки типовых решений программного комплекса показан в таблице 3.1. Первые три схемы демонстрируют возможности языка *Templet* при описании логики процессов. Для решения прикладных задач предназначены схемы 4–6. Применение данных схем не требует от пользователя специальных знаний в области параллельного программирования.

Часть кода программ написана на языке разметки *XML*. Этот код при работе в интегрированной среде (например, *Visual Studio*) либо генерируется по графическому образу процессов и каналов, представленных в главе 1, в специальном редакторе, либо пишется в *XML*-редакторе с поддержкой схем *XML*-документов. В данном случае редактор автоматически генерирует имена тэгов, атрибутов и разделители. Размер частей *XML* файла с учетом возможностей автоматического завершения приведен в последнем столбце таблицы 3.2.

Алгоритм генерации кода реализован в препроцессоре *Templet* [28], совместная работа которого с *IDE Microsoft Visual Studio* организуется, как описано ниже. Код программы пишется на языке *C++*, что предусмотрено в текущей реализации. Модулем в данном случае является пара файлов: заголовочный файл с расширением (**.h*); файл реализации с расширением (**.cpp*). Код этих файлов помещается в общее пространство имен. Каждому модулю в про-

Таблица 3.1 — Состав библиотеки программного комплекса

№ п/п	Кодовое имя схемы	Описание
Схемы с детальным описанием процессов и каналов на языке *Templet XML*		
1	*Map*	Схема «применить ко всем». Фиксированное число независимых подзадач со сборкой результата.
2	*TaskBag*	Схема «портфель задач». Разновидность схемы «управляющий — рабочие» с пассивным управляющим процессом.
3	*Pipeline*	Схема типа «конвейер» с двунаправленным потоком данных между ступенями.
Проблемно-ориентированные схемы		
4	*TBag*	«Портфель задач» для оптимизационных алгоритмов. Используется для исследования алгоритмов балансировки нагрузки имитационным методом.
5	*ADI*	Схема, реализующая численный метод переменных направлений. Используется для моделирования диссипативных систем.
6	*PLine*	Схема «конвейер» для моделирования электромагнитных процессов путем численного решения уравнений Максвелла.

екте *IDE* соответствует файл с *XML*-спецификацией, который тоже включается в проект (обычно в папку ресурсов приложения). Для файла *XML*-спецификации в проекте указывается способ его обработки. Перед сборкой проекта (*pre-built*) такой файл должен быть обработан *Templet*-препроцессором, который генерирует фрагменты кода, соответствующие модели программирования в файлах модуля. Также для файла *XML*-спецификации указывается редактор. Им может быть как штатный текстовый или *XML*-редактор, так и специальный графический редактор, визуализирующий модель программирования при помощи графических обозначений.

В процессе итеративной разработки, состоящей из чередующихся шагов редактирования, сборки и запуска программы, пользователь редактирует как *XML*-спецификацию модуля, так и код внутри модуля. Препроцессор синхронизирует *XML*-спецификацию и код модуля при каждой сборке. Вопрос кон-

троля ошибок при такой синхронизации решен следующим образом. Сначала проверяется, является ли *XML*-файл правильно построенным. Далее в процессе разбора *SAX*-парсером проверяются правила вложенности для тегов, описывающих структурные элементы модели программирования. Атрибуты могут быть пропущены и автоматически заменены значениями по умолчанию, если в препроцессоре активирована функция автоматического ввода (*autocomplete*). В препроцессоре имеется опция преобразования кода *XML*-спецификации, восстанавливающая все атрибуты, и форматирующая файл *XML*-спецификации с учетом вложенности тегов. Никакого контроля при генерации больше не выполняется. Семантика модели (например, соблюдение протокола канала) проверяется во время исполнения путем генерации соответствующих *assert*-инструкций в коде модуля. Если пользователь сформировал блоки, не соответствующие структурным элементам модели, они выносятся в отдельное место в файле модуля. Если в *XML*-спецификацию был добавлен новый элемент, для которого не было пользовательского блока, то генерируется пустой блок с комментарием, указывающим на необходимость его определения. Таким образом, основной контроль выполняется компилятором и системой исполнения (в случае *assert*-вызовов) целевого языка программирования. В текущей реализации это компилятор языка *C++*.

Описанная схема работы обеспечивает уверенность в правильном функционировании кода параллельной программы, так как весь критический код является открытым для анализа и отладки, а технически сложные части анализа выполняются надежным штатным компилятором.

В основе представленной системы автоматизации параллельного программирования лежит система визуального программирования *Graphplus* [9, 10, 28]. Усовершенствования коснулись модели программирования, в которую была добавлена концепция каналов, используемая в операционной системе *Singularity* и языке программирования *Sing#* [126]. Подвергся пересмотру метод определения семантики *XML*-спецификации. В отличие от работ [24, 25], где семантика модели описана в терминах темпоральной логики, применен метод эталонной реализации, упрощающий понимание системы программистом. В язык *XML*-спецификации введена возможность описания типовых решений (схем, паттернов, шаблонов) [2].

Данной системой используется парадигма языково-ориентированного

программирования, в рамках которой система может рассматриваться как пример доменно-ориентированного языка для параллельных вычислений. Система предназначена для реализации и использования паттернов параллельного программирования [127] в более гибкой форме, чем в соответствующих каркасных библиотеках. В частности, как в работе [7], единая вычислительная модель служит целям композиции типовых решений. Параллелизм в представленной системе вводится не как расширение языка или собственный язык, а в форме библиотеки времени исполнения [98], однако для обеспечения переносимости определяется лишь эталонная реализация. Система занимает промежуточное положение между языком и каркасной библиотекой, построена на основе препроцессора, и в этом похожа на систему программирования *Qt* [128]. Однако она не имеет жесткой привязки к языку программирования *C++*, не выполняет даже лексический разбор языковых конструкций, что значительно упрощает дизайн.

Общеупотребительными являются несколько способов введения семантики параллельного исполнения в язык программирования. Каркасная библиотека, например *TBB* на языке *C++* [129], жестко связывает решение с языком программирования. Расширение последовательного языка программирования требует написания как минимум лексического анализатора, что может оказаться неприемлемо сложным для практического применения, хотя в некоторых случаях удается построить компактные расширения, например *T++* [59]. Использование распараллеливающих директив в форме комментариев, например *OpenMP* [100] или *DVM* [63], тоже предполагает наличие сложного компилятора, но кроме этого, заставляет пользователя вручную приводить структуру кода в соответствие с моделью программирования этих средств. Использование приема генерации исходного кода вместо его анализа снимает данную проблему.

Семантика кода в представленной системе основана на модели процессов. Данный выбор обусловлен широкими описательными возможностями, императивностью, удобством декомпозиции сложных систем на ее основе. Несмотря на достоинства функциональной модели, и в том числе возможности описания некоторых паттернов параллелизма в ее терминах, такой подход более универсальный. Использование модели процессов диффузного типа [130] позволяет выполнить ее реализацию базовыми средствами процедур-

но-ориентированных языков и избегать манипулирования со стеком вызовов.

3.2.3 Описание реализации программного комплекса

Программный комплекс обучения и оптимизации структуры гиперрадиальной нейронной сети с использованием типового решения «портфель задач» [20] написан на языке *C++* (стандарт *ANSI C++*) с использованием парадигмы объектно-ориентированного программирования. Он реализует следующий функционал:

а) нормализация/денормализация входных и выходных данных;
б) обучение нейронной сети с подбором оптимальной структуры;
в) моделирование работы нейронной сети;
г) сохранение обученной нейронной сети в файл.

На этапе проектирования программного комплекса был проведен анализ предметной области. В результате была спроектирована диаграмма классов *UML* для разрабатываемой системы рис. 3.1.

а) Класс *Task* — абстрактный класс, представляющий передаваемые данные между процессами «портфеля задач».
б) Класс *TaskBagTask* — класс наследующий класс *Task* и непосредственно реализующий передаваемые данные между процессами «портфеля задач».
в) Класс *TBag* — абстрактный класс, представляющий «портфель задач».
г) Класс *TaskBag* — класс наследующий, класс *TBag* и непосредственно реализующий «портфель задач» со всем функционалом прикладной задачи.
д) Класс *CData* — класс, реализующий матрицу числовых данных.
е) Класс *CNeuron* — класс, реализующий гиперрадиальный нейрон.
ж) Класс *CNeuralNetwork* — класс, реализующий матрицу гиперрадиальную нейронную сеть.

Метод *PseudoInverse()* класса *CData*, реализующий псевдоинверсию матрицы *_Data*, заслуживает отдельного рассмотрения. Он реализован с использованием библиотеки *LAPACK* [91] версии 3.4.0. Данная библиотека, с использованием компилятора *GFortran* из состава *GNU Compiler Collection*, была со-

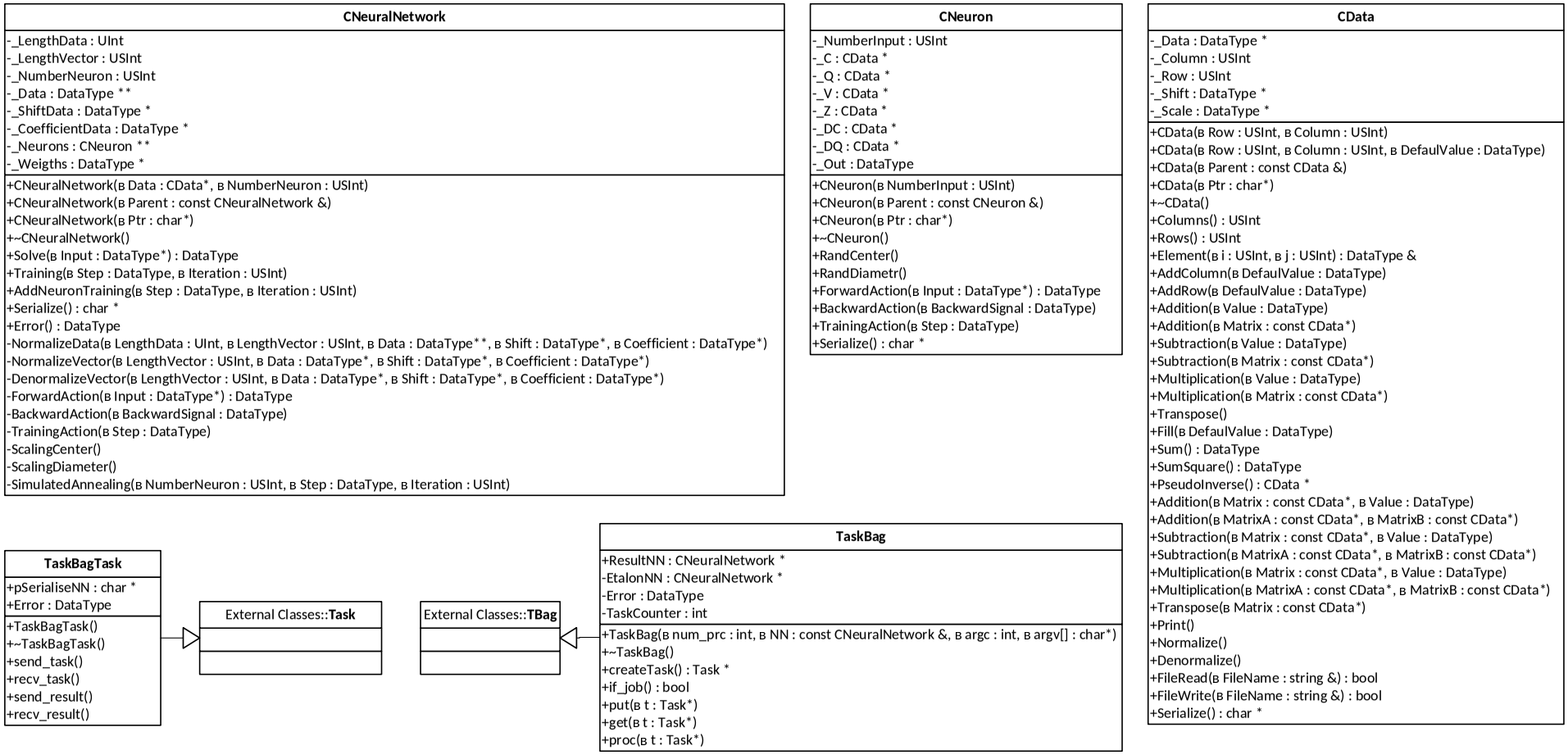


Рисунок 3.1 — Диаграмма классов *UML* программного комплекса

брана утилитой *CMAKE* под платформы *Windows API* и *POSIX*. В результате сборки были получены статические библиотеки, подключаемые к программному комплексу.

Так как алгоритм обучения нейронной сети, заложенный в программный комплекс, должен функционировать в системах и с общей памятью, и с распределенной памятью, необходимо было предусмотреть передачу объектных данных между процессами. Для такого обмена в классах *CData*, *CNeuron* и *CNeuralNetwork* реализованы методы сериализации/десериализации данных.

Структура модулей реализованного программного комплекса имеет вид 3.2.

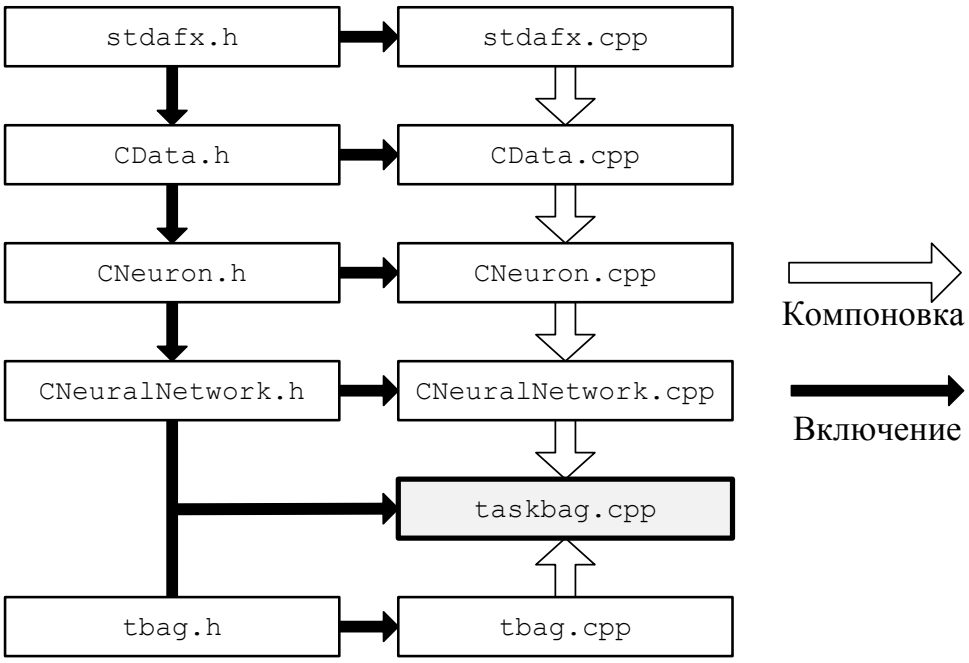


Рисунок 3.2 — Структура модулей программного комплекса

Функционал модулей.

а) Модули *stdafx.cpp* и *stdafx.h* — общие модули с описанием типов и подключением библиотек.

б) Модули *CData.cpp* и *CData.h* — модули с прототипом и реализацией класса *CData*.

в) Модули *CNeuron.cpp* и *CNeuron.h* — модули с прототипом и реализацией класса *CNeuron*.

г) Модули *CNeuralNetwork.cpp* и *CNeuralNetwork.h* — модули с прототипом и реализацией класса *CNeuralNetwork*.

д) Модули *tbag.cpp* и *tbag.h* — модули библиотеки времени исполнения с прототипами и реализацией классов *Task* и *TBag*.

е) Модуль *taskbag.cpp* — главный модуль программы, реализующий «портфель задач» (классы *TaskBagTask* и *TaskBag*).

В качестве формата файлов входных данных в программном комплексе использован формат **.csv* (*Comma-Separated Values*), который предназначен для хранения табличных данных. В нем значения колонок разделяются символом «,», а каждая строка файла — это одна строка таблицы. Этот формат прост для чтения и редактирования, и многие финансовые системы экспортируют данные именно в нем.

Рассчитаем степень автоматизации кодирования на основе сравнения объемов повторно используемого кода типового решения и кода алгоритма обучения нейронной сети. Для модулей, реализующих нейронную сеть (*CData*, *CNeuron* и CNeuralNetwork) объем реализованного кода в сумме составил 36626 байт. Объем управляющего кода (*tbag* и *taskbag*) для *Windows API* составил 17004 байта, для *API POSIX* 16960 байт и отладочный 15225 байт, всего — 49189 байт. Таким образом снижение трудоемкости кодирования можно оценить как отношение общего объема кода к объему прикладного кода $85815/36626 \approx 2,34$.

3.3 Экспериментальная оценка эффективности распараллеливания задач математического моделирования с использованием типовых решений

3.3.1 Оценка сложности реализации численных моделей по объему ручного и автоматического кодирования

Косвенную оценку трудоемкости программирования при использовании программного комплекса дает сопоставление размеров рукописного и сгенерированного кода, представленное в таблицах 3.2 и 3.3.

Размер рукописного кода относится только к алгоритму схемы, а не к численному методу. По таблицам видно, что управляющий алгоритм составляет значительную долю кода. Эффект от автоматизации заключается в его

Таблица 3.2 — Размеры последовательных версий схем программ

№ п/п	Кодовое имя схемы	Размер кода *C++*, байты	Размер кода *XML*, байты
1	*Map*	11044	1820
2	*TaskBag*	15225	2933
3	*Pipeline*	10049	2962
4	*TBag*	3040	213
5	*ADI*	4711	229
6	*PLine*	2752	213

повторном использовании.

При работе над прикладным алгоритмом пользователь системы работает только с кодом, размер которого указан в таблице 3.2. Из сравнения таблицы 3.2 с таблицей 3.3 видно, что размер последовательного кода, который приходится анализировать при отладке несколько меньше. Однако следует учитывать, что код из таблицы 3.3 содержит системные вызовы синхронизации. Его ручная разработка требует квалификации в области параллельного программирования.

Таблица 3.3 — Размеры кода на языке *C++* для *Windows API* и для *API POSIX*

№ п/п	Кодовое имя схемы	Размер кода на *C++* для *Windows API*, байты	Размер кода на *C++* для *API POSIX*, байты
1	*Map*	12823	12779
2	*TaskBag*	17004	16960
3	*Pipeline*	11828	11784
4	*TBag*	4502	4504
5	*ADI*	7705	8237
6	*PLine*	3906	4309

3.3.2 Результаты применения типового решения «метод переменных направлений» в задаче исследования нелинейных динамических процессов

Для решения задачи нелинейной динамики использовалась схема «метод переменных направлений» (*ADI* таблица 3.1) из набора библиотек программ-

ного комплекса, реализующая метод переменных направлений в общей памяти. Алгоритм схемы предусматривает чередование построчного и постолбцового сканирования области заданного размера. Элементарные операции над ячейкой области на четных и нечетных обходах, а также структура данных зависят от конкретного алгоритма. Рассматривалась параллельная реализация численного метода, описанного в работе [94]. В данной реализации решалась задача нелинейной динамики на области размером 5000×5000 по пространству и 10 отсчетов по времени.

Вычислительные эксперименты проводились для двух вариантов развертывания параллельного кода под архитектуры *Windows API* и *POSIX*. Во всех экспериментах учитывалось время инициализации и очистки. Для задачи была проведена оценка времени решения, реального ускорения и эффективности распараллеливания.

Для архитектуры *Windows API* использовалась многоядерная настольная система *Samsung RF710* со следующими характеристиками: процессор *Intel*® *Core*^{TM}*i5 M460*, 2,53GHz, 4 системных потока; оперативная память 8Гб; жесткий диск 120Гб; операционная система *Windows 7 Ultimate 64 bits*. Значения временных характеристик решения задачи для различного количества логических процессов P приведены в таблице 3.4. Здесь и далее эффективность определяется как отношение ускорения к числу системных потоков.

Таблица 3.4 — Временные характеристики решения задачи нелинейной динамики на архитектуре *Windows API*

Количество логических процессов P, шт.	Время исполнения, с	Ускорение	Эффективность, %
1	122,850	1,00	25,0
2	71,504	1,72	43,0
3	62,857	1,95	48,9
4	56,995	2,15	53,9
5	56,869	2,16	54,0
6	54,980	2,23	55,9
7	53,756	2,28	57,1
8	53,741	2,27	57,1
9	53,724	2,29	57,2
10	53,022	2,32	57,9

Для большей наглядности результатов эксперимента был построен график зависимости ускорения от количества логических процессов P (рис. 3.3).

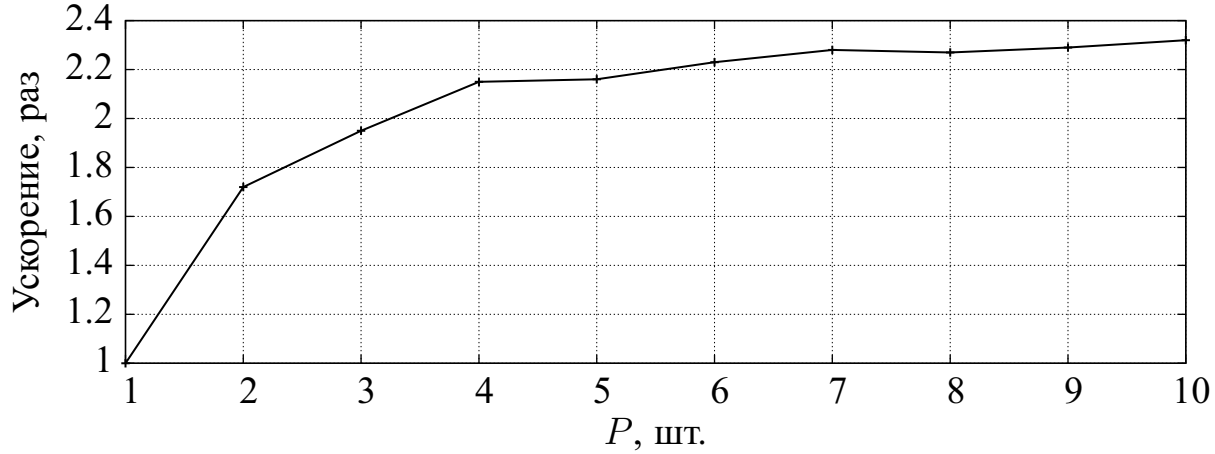


Рисунок 3.3 — Зависимость ускорения от количества логических процессов P при решении задачи нелинейной динамики на архитектуре *Windows API*

Для архитектуры *POSIX* использовался узел кластера «Сергей Королёв» (`http://hpc.ssau.ru/`) со следующими характеристиками: тип узла *IBM BladeCenter HS22*; 2 процессора *Intel®Xeon®X5560*, 2,8GHz, 8MB *Cache*, 8 системных потоков; оперативная память 12Гб; жесткий диск 76Гб или 146Гб *SAS* 10K 16Mb *Cache*; тип системной сети *InfiniBand DDR*, *QDR*; операционная система *Red Hat Enterprise Linux 5.6*. Значения временных характеристик решения задачи для различного количества логических процессов P приведены в таблице 3.5.

Для большей наглядности результатов эксперимента был построен график зависимости ускорения от количества логических процессов P (рис. 3.4).

3.3.3 Результаты применения типового решения «конвейер» в задаче о распространении плоской световой волны

Для задачи о распространении плоской световой волны на основе численного решения уравнений Максвелла, Хайрутдиновым А.Р. [18] было проведено распараллеливание последовательного алгоритма из работы [93]. Для этого использовалась схема «конвейер» (*PLine* таблица 3.1) из набора библиотек программного комплекса. В ней расчетная область представляет со-

Таблица 3.5 — Временные характеристики решения задачи нелинейной динамики на архитектуре *POSIX*

Количество логических процессов P, шт.	Время исполнения, с	Ускорение	Эффективность, %
1	76,494	1,00	12,5
2	72,730	1,05	13,1
3	34,807	2,20	27,4
4	29,239	2,62	32,7
5	23,261	3,29	41,1
6	23,603	3,24	40,5
7	22,967	3,33	41,6
8	22,759	3,36	42,0
9	31,221	2,45	30,6
10	52,438	1,46	18,2

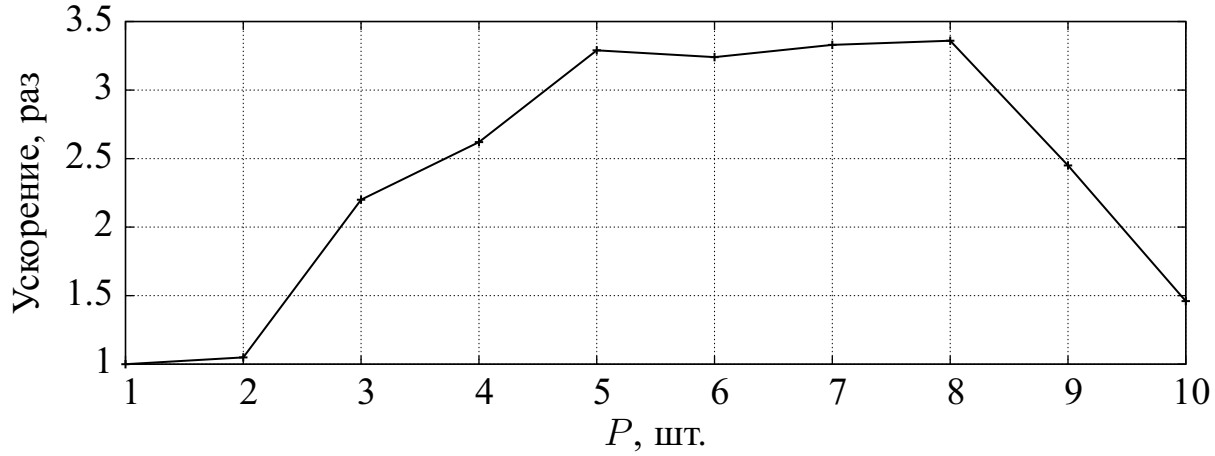


Рисунок 3.4 — Зависимость ускорения от количества логических процессов P при решении задачи нелинейной динамики на архитектуре *POSIX*

бой цепочку ячеек, над которыми последовательно выполняется элементарная операция. Фронт вычислений распространяется слева направо. Элементарная операция не может запускаться в соседних ячейках. В рассмотренной реализации ячейка хранит 3 вектора проекций электромагнитного поля, размер сеточной области в экспериментах составлял 50×201, количество итераций численного метода равно 799.

Вычислительные эксперименты проводились для двух вариантов развертывания параллельного кода под архитектуры *Windows API* и *POSIX*. Во всех

экспериментах учитывалось время инициализации и очистки. Для задачи была проведена оценка времени решения, реального ускорения и эффективности распараллеливания.

Для архитектуры *Windows API* использовалась многоядерная настольная система *Samsung Q210* со следующими характеристиками: процессор *Intel® Core™2 Duo*, 2GHz, 2 системных потока; оперативная память 3Гб; жесткий диск 250Гб; операционная система *Windows Vista Home 32 bits*. Значения временных характеристик решения задачи для различного количества логических процессов P приведены в таблице 3.6.

Таблица 3.6 — Временные характеристики решения задачи о распространении плоской световой волны на архитектуре *Windows API*

Количество логических процессов P, шт.	Время исполнения, с	Ускорение	Эффективность, %
1	1,271	1,00	50,0
2	1,167	1,09	54,5
3	0,824	1,54	77,0
4	0,665	1,91	95,5
5	0,725	1,75	87,5
6	0,796	1,60	80,0
8	0,801	1,59	79,5

Для большей наглядности результатов эксперимента был построен график зависимости ускорения от количества логических процессов P (рис. 3.5).

Для архитектуры *POSIX* использовался узел кластера Сергей Королёв (`http://hpc.ssau.ru/`) со следующими характеристиками: тип узла *IBM BladeCenter HS22*; 2 процессора *Intel®Xeon®X5560*, 2,8GHz, 8MB *Cache*, 8 системных потоков; оперативная память 12Гб; жесткий диск 76Гб или 146Гб *SAS* 10K 16Mb *Cache*; тип системной сети *InfiniBand DDR*, *QDR*; операционная система *Red Hat Enterprise Linux 5.6*. Значения временных характеристик решения задачи для различного количества логических процессов P приведены в таблице 3.7.

Для большей наглядности результатов эксперимента был построен график зависимости ускорения от количества логических процессов P (рис. 3.6).

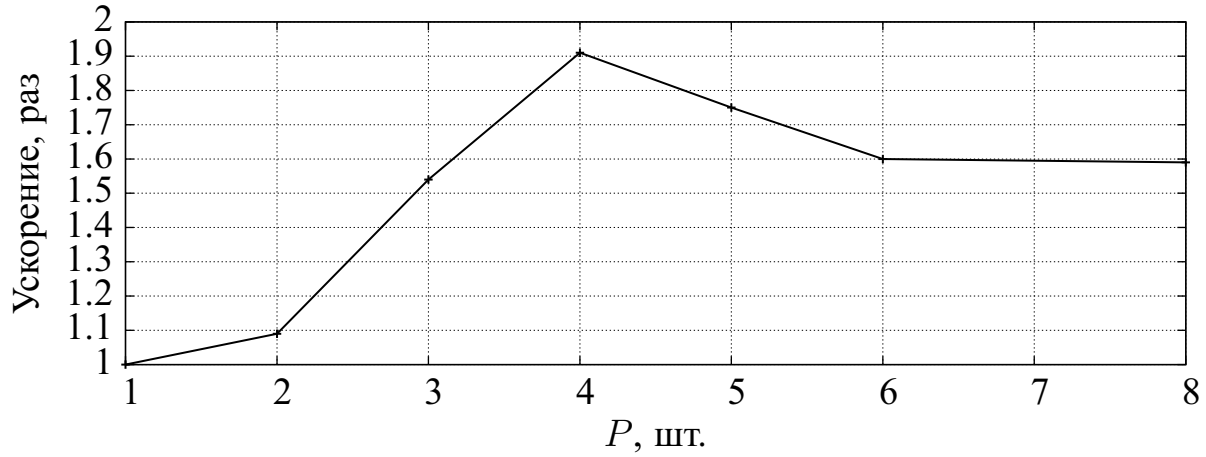


Рисунок 3.5 — Зависимость ускорения от количества логических процессов P при решении задачи о распространении плоской световой волны на архитектуре *Windows API*

Таблица 3.7 — Временные характеристики решения задачи о распространении плоской световой волны на архитектуре *POSIX*

Количество логических процессов, шт.	Время исполнения, с	Ускорение	Эффективность, %
1	0,906	1,00	12,5
2	0,814	1,11	13,9
4	0,473	1,92	24,0
8	0,282	3,21	40,1
16	0,194	4,67	58,4
20	0,184	4,92	61,5
24	0,186	4,87	60,9
28	0,235	3,85	48,1
32	0,311	2,91	36,4

3.3.4 Результаты применения типового решения «портфель задач» в задаче моделирования фазовых траекторий динамических систем

Для анализа параметров многомерных динамических систем с помощью сечений Пуанкаре авторами работы [92] использовалась схема «портфель задач» (*TBag* таблица 3.1), разработанная в диссертации. Задача анализа многомерных динамических систем с помощью сечений Пуанкаре содержит неза-

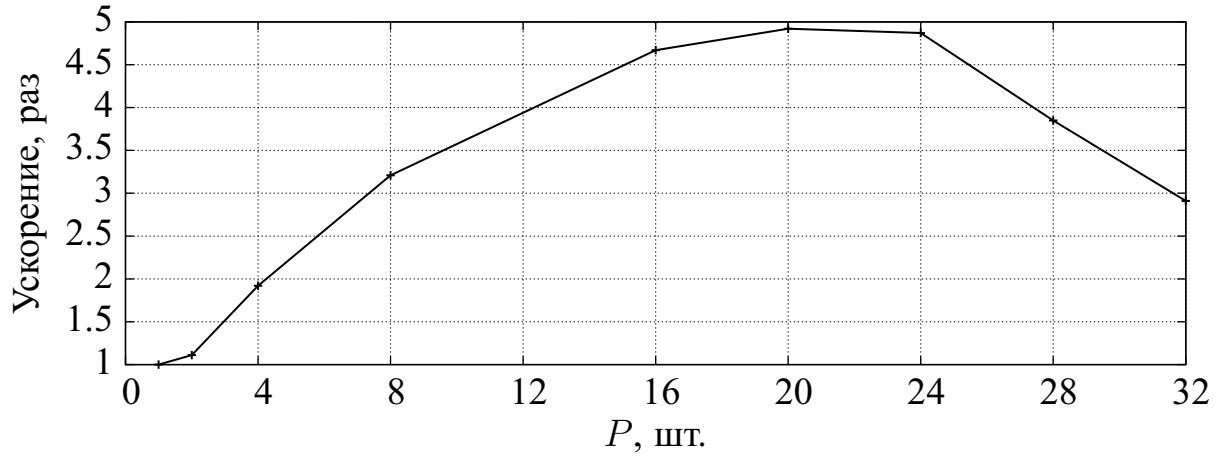


Рисунок 3.6 — Зависимость ускорения от количества логических процессов P при решении задачи о распространении плоской световой волны на архитектуре *POSIX*

висимые по данным этапы вычисления координат пересечения траекторий с плоскостью сечения [95]. Распараллеливание вычисления одной траектории является крайне сложной задачей, так как независимыми по данным в каждый момент времени являются не более нескольких десятков машинных инструкций. Распараллеливание производилось по блокам вычисления траекторий в связи с высокими затратами на синхронизацию в случае распараллеливания вычислений одной траектории.

Решение задачи состояло из вычисления сечения 100–200 траекторий. Некоторого ускорения вычислений можно добиться, увеличивая число потоков до числа траекторий. Но для эффективного использования вычислительных ресурсов необходимо, чтобы число траекторий было в 2–10 раз больше числа потоков вычислений, и это накладывало ограничения на масштабируемость системы.

Достигнутая длительность вычисления 160 траекторий на 20 узлах кластера в 120 раз меньше, чем у последовательном варианта, при этом использовались 160 потоков вычислений. Результат построения сечения траекторий приведен на рис. 3.7.

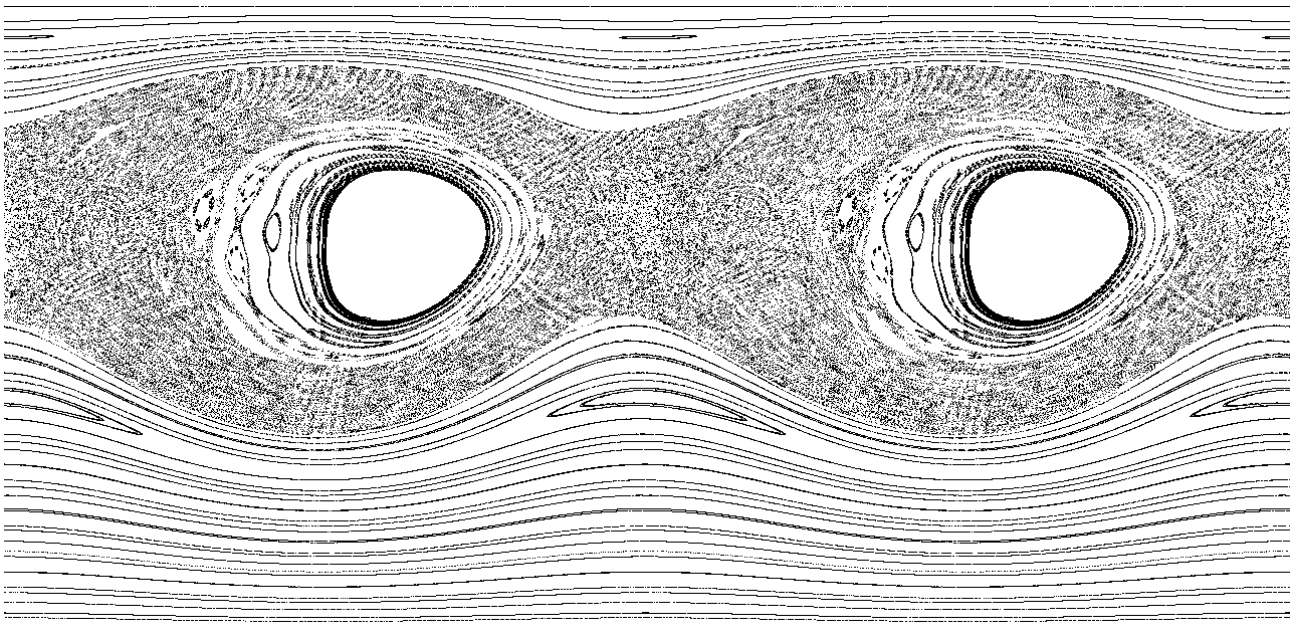


Рисунок 3.7 — Вид сечения Пуанкаре

3.4 Выводы

а) На основе обзора программного обеспечения параллельных вычислений показано, что наиболее целесообразными методами реализации типовых решений в виде параллельных библиотек являются: *Windows API*, *POSIX threads*, *MPI*.

б) Предложена и рассмотрена реализация программного комплекта автоматизации параллельных вычислений на основе *web*-сервиса для суперкомпьютерного центра *СГАУ*. Описан состав и характеристики библиотеки программного комплекса.

в) Продемонстрировано снижение трудоемкости кодирования численных методов при использовании библиотеки типовых решений по сравнению с ручным кодированием для задачи обучения гиперрадиальной нейронной сети примерно равное 2,34.

г) В вычислительных экспериментах обоснована практическая целесообразность применения предложенного в работе подхода. При распараллеливании задачи нелинейной динамики с использованием типового решения «метод переменных направлений» на рабочей станции получено ускорение 2,28 и эффективность 57,1%; на узле суперкомпьютера —

ускорение 3,36 и эффективность 42%. В задаче о распространении плоской световой волны на рабочей станции получено ускорение 1,91 и эффективность 95,5%; на узле суперкомпьютера – ускорение 4,92 и эффективность 61,5%. В задаче моделирования фазовых траекторий динамических систем на суперкомпьютере с использованием типового решения «портфель задач» достигнуто 120 кратное ускорение при вычислении 160 траекторий на 20 восьмиядерных узлах (эффективность 75%).

ЗАКЛЮЧЕНИЕ

Основные выводы и результаты, полученные в работе.

а) Выполнен анализ актуальных проблем разработки программных комплексов высокопроизводительных научных вычислений. Показана перспективность развития подхода, основанного на применении типовых решений.

б) Разработана математическая модель вычислительного процесса для описания различных типовых решений в области высокопроизводительных научных вычислений.

в) Разработана графическая модель алгоритма для типового решения «портфель задач» на основе математической модели вычислительного процесса и графической нотации *Templet*.

г) Предложена методика синтеза параллельного алгоритма на примере алгоритма обучения гиперрадиальной нейронной сети с использованием модели типового решения «портфель задач».

д) Разработан и внедрён в учебно-исследовательский процесс программный комплекс «портфель задач» для использования на высокопроизводительных вычислительных системах в составе сервиса *Templet Web* (`http://templet.ssau.ru`).

е) Проведены экспериментальные исследования, подтверждающие эффективность применения программного комплекса. В частности, в задаче обучения геперрадиальной нейронной сети ускорение составило порядка 13 раз на 16-ти ядерном узле при этом снижение сложности разработки, оцененное по объёму ручного кодирования, составило 2,34 раза.

СПИСОК ЛИТЕРАТУРЫ

1. Марчук Г. И., Котов В. Е. Проблемы вычислительной техники и фундаментальные исследования // Автом. и вычисл. техн. — 1979. — № 2. — С. 3–14.
2. Приемы объектно-ориентированного проектирования. паттерны проектирования / Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес // СПб.: Питер. — 2001. — Т. 4. — С. 3.
3. Cole M. Algorithmic Skeletons: Structured Management of Parallel Computation. — Cambridge, MA, USA : MIT Press, 1991. — ISBN: 0-262-53086-4.
4. MacDonald S. Design patterns in enterprise // Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research. — CASCON '96. — IBM Press, 1996. — P. 25. — URL: http://dl.acm.org/citation.cfm?id=782052.782077.
5. Generating parallel program frameworks from parallel design patterns / Steve MacDonald, Duane Szafron, Jonathan Schaeffer, Steven Bromling // Euro-Par 2000 Parallel Processing / Ed. by Arndt Bode, Thomas Ludwig, Wolfgang Karl, Roland Wismuller. — Springer Berlin Heidelberg, 2000. — Vol. 1900 of Lecture Notes in Computer Science. — P. 95–104. — URL: http://dx.doi.org/10.1007/3-540-44520-X_13.
6. From patterns to frameworks to parallel programs / S. MacDonald, J. Anvik, S. Bromling et al. // Parallel Computing. — 2002. — Vol. 28, no. 12. — P. 1663–1683. — URL: http://dx.doi.org/10.1016/S0167-8191(02)00190-4.
7. Берзигияров П. К. Программирование на типовых алгоритмических структурах с массивным параллелизмом // Вычислительные методы и программирование. — 2001. — Т. 2, № 2.
8. Kraeva M. A., Malyshkin V. E. Assembly technology for parallel realization of numerical models on mimd-multicomputers // Future Generation Computer Systems. — 2001. — Vol. 17, no. 6. — P. 755–765. — I: PaCT. II: {HPC} applications. URL: http://dx.doi.org/10.1016/S0167-739X(00)00058-3.
9. Востокин С. В. Графическая объектная модель параллельных процессов и ее применение в задачах численного моделирования. — Самара : Изд-во СНЦ РАН, 2007. — С. 286.

10. Востокин С. В. Визуальное моделирование в разработке параллельных алгоритмов. Метод и программные средства. Информатика, ИТ. — LAP LAMBERT Academic Publishing, 2011.
11. Литвинов В. Г. Нейросетевой аппроксиматор на основе радиально-базисных функций с масштабирующей матрицей // X Королёвские чтения: Всероссийская молодёжная научная конференция с международным участием, Самара, 6–8 октября 2009 года: Тезисы докладов / Самарский государственный аэрокосмический университет. — Самара : Изд-во Самарского государственного аэрокосмического университета, 2009. — С. 293.
12. Литвинов В. Г., Солдатова О. П. Методы построения оптимальной структуры hrbf нейронной сети для решения задачи прогнозирования // Перспективные информационные технологии для авиации и космоса (ПИТ-2010). Труды Международной конференции с элементами научной школы для молодежи / Самарский государственный аэрокосмический университет. — Самара : Изд-во Самарского государственного аэрокосмического университета, 2010. — С. 252–254.
13. Литвинов В. Г. Работа комплекса типовых решений параллельных вычислений на основе модели «граф плюс» на суперкомпьютерных системах // Тезисы докладов XXXVII Самарской областной студенческой научной конференции / Самарский государственный аэрокосмический университет. — Самара : ООО «ИПК «Право», 2011. — С. 275.
14. Визуальное моделирование параллельных алгоритмов в процессно-ориентированной нотации templet / С. В. Востокин, А. Р. Хайрутдинов, В. Г. Литвинов, Макагонова Д. Д. // Труды Шестой Международной конференции «Параллельные вычисления и задачи управления» PACO'2012 / Институт проблем управления им. В. А.Трапезникова РАН. — Т. 1. — М. : Изд-во ИПУ РАН, 2012. — С. 260–269. — URL: `http://paco2012.ipu.ru/procdngs/E208.pdf`.
15. Представление алгоритмов высокопроизводительных вычислений в нотации templet / С. В. Востокин, В. Г. Литвинов, Д. Д. Макагонова, А. Р. Хайрутдинов // Перспективные информационные технологии в научных исследованиях, проектировании и обучении (ПИТ 2012): труды научно-технической конференции с международным участием и элемен-

тами научной школы для молодежи, посвященной 40-летию кафедры информационных систем и технологий СГАУ / Под ред. С. А. Прохоров ; Самарский государственный аэрокосмический университет. — Самара : Изд-во СНЦ РАН, 2012. — С. 74–76.

16. Востокин С. В., Литвинов В. Г. Применение типовых решений для распараллеливания алгоритмов численного моделирования // Перспективные информационные технологии (ПИТ 2013): труды Международной научно-технической конференции / Под ред. С. А. Прохоров ; Самарский государственный аэрокосмический университет. — Самара : Изд-во СНЦ РАН, 2013. — С. 184–187.
17. Востокин С. В., Хайрутдинов А. Р., Литвинов В. Г. Программный комплекс параллельного программирования graphplus templet // Вестник Самарского государственного технического университета: Серия «Физико-математические науки». — 2011. — Т. 25, № 4. — С. 146–153. — URL: http://dx.doi.org/10.14498/vsgtu1027.
18. Востокин С. В., Хайрутдинов А. Р., Литвинов В. Г. Применение комплекса параллельного программирования graphplus templet в моделировании // Программные продукты и системы. — 2012. — № 3. — С. 14–18.
19. Литвинов В. Г. Разработка и применение вычислительной модели типовых решений: пример использования «портфеля задач» для обучения нейронной сети hrbf // Вестник Самарского государственного технического университета: Серия «Физико-математические науки». — 2014. — Т. 36, № 3. — С. 183–195. — URL: http://dx.doi.org/10.14498/vsgtu1341.
20. Литвинов В. Г. Программа обучения и оптимизации структуры гиперрадиальной нейронной сети с использованием типового решения «портфель задач». / свидетельство о государственной регистрации программ для эвм №2014661813, выданное федеральной службой по интеллектуальной собственности. зарегистрировано в реестре программ для эвм 13.11.2014. — 2014. — URL: http://www1.fips.ru/Archive/EVM/2014/2014.12.20/DOC/RUNW/000/002/014/661/813/document.pdf.
21. Литвинов В. Г. Моделирование динамики пространственно-распределенных систем типа «реакция-диффузия» с использованием

паттерна «портфель задач» на суперкомпьютере // Королёвские чтения: Международная молодежная конференция, посвященная 50-ти летию первого полета человека в космос, Самара, 4–6 октября 2011 года: Тезисы докладов / Самарский государственный аэрокосмический университет. — Самара : Изд-во Самарского государственного аэрокосмического университета, 2011. — С. 282.

22. Литвинов В. Г. Экстраполяция поведения фондового рынка искусственной нейронной сетью на основе радиально базисных функций // Тезисы докладов XXXVI Самарской областной студенческой научной конференции / Самарский государственный аэрокосмический университет. — Т. 1. — Самара : ООО «ИПК «Право», 2010. — С. 307–308.
23. González-Vélez H., Leyton M. A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers // Software: Practice and Experience. — 2010. — Vol. 40, no. 12. — P. 1135–1160. — URL: http://dx.doi.org/10.1002/spe.1026.
24. Востокин С. В. Спецификация модели параллельных и распределенных вычислений graphplus на основе логики tla // Известия СНЦ РАН. — 2006. — Т. 8, № 3. — С. 875–881.
25. Востокин С. В. Спецификация визуальной модели параллельных и распределенных вычислений на основе логики tla // Труды Четвертой Международной конференции «Параллельные вычисления и задачи управления» PACO'2008 / Институт проблем управления им. В. А.Трапезникова РАН. — М. : Изд-во ИПУ РАН, 2008. — С. 1338–1348.
26. Lamport L. The temporal logic of actions // ACM Trans. Program. Lang. Syst. — 1994. — May. — Vol. 16, no. 3. — P. 872–923. — URL: http://dx.doi.org/10.1145/177492.177726.
27. Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. — ISBN: 032114306X.
28. Востокин С. В. Система автоматизации параллельного программирования graphplus templet // Труды Пятой Международной конференции «Параллельные вычисления и задачи управления» PACO'2010 / Институт проблем управления им. В. А.Трапезникова РАН. — М. : Изд-во ИПУ РАН, 2010. — С. 1143–1156.

29. Востокин С. В. Templet — метод процессно-ориентированного моделирования параллелизма // Программные продукты и системы. — 2012. — № 3. — С. 11–14.
30. Элементы параллельного программирования / В. А. Вальковский, В. Е. Котов, А. Г. Марчук, Н. Н. Миренков. — Радио и связь, 1983.
31. Поспелов Д. А. Введение в теорию вычислительных систем. — Сов. радио, 1972.
32. Baer J.-L., Bovet D. P. Compilation of arithmetic expressions for parallel computations. // IFIP Congress (1). — 1968. — P. 340–346.
33. Ramamoorthy C. V., Park J. H., Li H. F. Compilation techniques for recognition of parallel processable tasks in arithmetic expressions // IEEE Trans. Comput. — 1973. — November. — Vol. 22, no. 11. — P. 986–998. — URL: http://dx.doi.org/10.1109/T-C.1973.223635.
34. Андрианов А. Н., Ефимкин К. Н., Задыхайло И. Б. Непроцедурный язык для решения задач математической физики // Программирование. — 1991. — № 2. — С. 80–94.
35. Chandy K. M. Parallel program design // Opportunities and Constraints of Parallel Computing / Ed. by Jorge L. C. Sanz. — Springer US, 1989. — P. 21–24. — URL: http://dx.doi.org/10.1007/978-1-4613-9668-0_6.
36. Milner R. A calculus of communicating systems. — springer-Verlag Berlin, 1980. — Vol. 92.
37. Owicki S., Gries D. An axiomatic proof technique for parallel programs i // Acta Informatica. — 1976. — Vol. 6, no. 4. — P. 319–340. — URL: http://dx.doi.org/10.1007/BF00268134.
38. Pnueli A. The temporal logic of programs // Foundations of Computer Science, 1977., 18th Annual Symposium on. — 1977. — October. — P. 46–57. — URL: http://dx.doi.org/10.1109/SFCS.1977.32.
39. Pnueli A. The temporal semantics of concurrent programs // Semantics of Concurrent Computation / Ed. by Gilles Kahn. — Springer Berlin Heidelberg, 1979. — Vol. 70 of Lecture Notes in Computer Science. — P. 1–20. — URL: http://dx.doi.org/10.1007/BFb0022460.
40. Schwartz R., Melliar-Smith P. From state machines to temporal logic: Specification methods for protocol standards // Communications, IEEE Transac-

tions on. — 1982. — December. — Vol. 30, no. 12. — P. 2486–2496. — URL: http://dx.doi.org/10.1109/TCOM.1982.1095451.

41. Sulistio A., Yeo C. S., Buyya R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools // Software: Practice and Experience. — 2004. — Vol. 34, no. 7. — P. 653–673. — URL: http://dx.doi.org/10.1002/spe.585.
42. Нестеренко Б. Б., Марчук В. А. Основы асинхронных методов параллельных вычислений. — Наук. думка, 1989.
43. Котов В. Е., Нариньяни А. С. Асинхронные вычислительные процессы над памятью. — Кибернетика, 1966.
44. Thiagarajan P. S. Some aspects of net theory // The Analysis of Concurrent Systems / Ed. by B. T. Denvir, W. T. Harwood, M. I. Jackson, M. J. Wray. — Springer Berlin Heidelberg, 1985. — Vol. 207 of Lecture Notes in Computer Science. — P. 26–54. — URL: http://dx.doi.org/10.1007/3-540-16047-7_34.
45. Питерсон Д. Теория сетей Петри и моделирование систем. — Мир, 1984.
46. Chapter on distributed computing : Rep. / DTIC Document ; Executor: Leslie Lamport, Nancy Lynch : 1989.
47. Dijkstra E. W. The structure of the “the” multiprogramming system // Classic Operating Systems / Ed. by PerBrinch Hansen. — Springer New York, 2001. — P. 223–236. — URL: http://dx.doi.org/10.1007/978-1-4757-3510-9_12.
48. Hoare C. A. R. Monitors: An operating system structuring concept // Commun. ACM. — 1974. — October. — Vol. 17, no. 10. — P. 549–557. — URL: http://dx.doi.org/10.1145/355620.361161.
49. Хоар Ч. Взаимодействующие последовательные процессы. — Рипол Классик, 1989.
50. Spf: Structured parallel fortran / John Darlington, Yi-ke Guo, Hing Wing To, Jin Yang // In: PCW’96 — Intl. Parallel Computing Workshop. Japan. Kawasaki. — 1996. — P. 1–6.
51. Bratvold T. Skeleton-based parallelization of functional programs : Ph. D. thesis / TA Bratvold ; Dept. Of Computing and Electrical Engineering. Edinburgh: Heriot-Watt University. — 1994.
52. Botorog G. H., Kuchen H. Skil: an imperative language with algorithmic

skeletons for efficient distributed programming // High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on. — 1996. — August. — P. 243–252. — URL: http://dx.doi.org/10.1109/HPDC.1996.546194.

53. Serot J., Ginhac D., Derutin J.-P. Skipper: A skeleton-based parallel programming environment for real-time image processing applications // Parallel Computing Technologies / Ed. by Victor Malyshkin. — Springer Berlin Heidelberg, 1999. — Vol. 1662 of Lecture Notes in Computer Science. — P. 296–305. — URL: http://dx.doi.org/10.1007/3-540-48387-X_31.
54. Pelagatti S. Structured development of parallel programs. — Taylor & Francis Abington, 1998. — Vol. 102.
55. Skie: A heterogeneous environment for {HPC} applications / B. Bacci, M. Danelutto, S. Pelagatti, M. Vanneschi // Parallel Computing. — 1999. — Vol. 25, no. 13–14. — P. 1827–1852. — URL: http://dx.doi.org/10.1016/S0167-8191(99)00072-1.
56. Singh A., Schaeffer J., Szafron D. Views on template-based parallel programming // Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research. — CASCON '96. — IBM Press, 1996. — P. 35. — URL: http://dl.acm.org/citation.cfm?id=782052.782087.
57. Design patterns for parallel programming / Stephen Siu, Mauricio De Simone, Dhrubajyoti Goswami, Ajit Singh // PDPTA. — Vol. 96. — 1996. — P. 230–240.
58. Feo J. T., Cann D. C., Oldehoeft R. R. A report on the sisal language project // Journal of Parallel and Distributed Computing. — 1990. — Vol. 10, no. 4. — P. 349–366. — Data-flow Processing. URL: http://dx.doi.org/10.1016/0743-7315(90)90035-N.
59. Абрамов С. М., Адамович А. И., Коваленко М. Р. Т-система — среда программирования с поддержкой автоматического динамического распараллеливания программ // Программирование. — 1999. — Т. 25, № 2. — С. 100–107.
60. Bert 77: Automatic and efficient parallelizer for fortran. — 2014. — URL: http://www.plogic.com/bert.html.
61. Антонов А. С., Воеводин В. В. Эффективная адаптация последова-

тельных программ для современных векторно-конвеерных и массивно-параллельных супер-эвм // Программирование. — 1996. — Т. 4. — С. 37–51.

62. High performance fortran. — 2014. — URL: http://hpff.rice.edu/.
63. Крюков В. А. Отладка dvm-программ // Препринты Института прикладной математики им. МВ Келдыша РАН. — 1999. — № 0. — С. 56–1.
64. Beguelin A., Dongarra J. J. Graphical development tools for network-based concurrent supercomputing // Proceedings of the 1991 ACM/IEEE Conference on Supercomputing. — Supercomputing '91. — New York, NY, USA : ACM, 1991. — P. 435–444. — URL: http://dx.doi.org/10.1145/125826.126059.
65. Beguelin A. L., Nutt G. J. Collected papers on Phred. — Department of Computer Science, University of Colorado, 1991.
66. Rost J. "d2r": A dynamic dataflow representation for task scheduling // SIGPLAN Not. — 1993. — August. — Vol. 28, no. 8. — P. 109–116. — URL: http://dx.doi.org/10.1145/163114.163128.
67. Коварцев А. Н. Автоматизация разработки и тестирования программных средств. — Самара, 1999.
68. Калентьев А. А., Тюгашев А. А. Ипи/cals технологии в жизненном цикле комплексных программ управления // Самара: Изд-во Самар. науч. цента РАН.-2006. — 2006.
69. Browne J. C., Azam M., Sobek S. Code: a unified approach to parallel programming // Software, IEEE. — 1989. — July. — Vol. 6, no. 4. — P. 10–18. — URL: http://dx.doi.org/10.1109/52.31648.
70. Paralex: An environment for parallel programming in distributed systems / Özalp Babaoğlu, Lorenzo Alvisi, Alessandro Amoroso et al. // Proceedings of the 6th International Conference on Supercomputing. — ICS '92. — New York, NY, USA : ACM, 1992. — P. 178–187. — URL: http://dx.doi.org/10.1145/143369.143406.
71. Traversat B. NEPTUNE: The Application of CourseGrain Data Flow Methods to Scientific Parallel Programming : Ph. D. thesis / B Traversat ; The Florida State University. — 1990.
72. Loyall J. P. Specification of Concurrent Systems Using Graph Grammars : Ph. D. thesis / J. P. Loyall ; Dept. of Comp. Sci, Univ. of Illinois, Urbana-

Champaign. — 1991.

73. Bailey D. A., Cuny J. E., Loomis C. P. Paragraph: Graph editor support for parallel programming environments // International Journal of Parallel Programming. — 1990. — Vol. 19, no. 2. — P. 75–110. — URL: http://dx.doi.org/10.1007/BF01407832.
74. Object-oriented modeling and design / James Rumbaugh, Michael Blaha, William Premerlani et al. — Prentice-hall Englewood Cliffs, 1991.
75. Object-oriented software engineering: a use case driven approach / Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard // [New York]: ACM Press; Wokingham, Eng.; Reading, Mass.: Addison-Wesley Pub.,— c1992, Revised printing. — 1992. — Vol. 1.
76. Herlihy M. P. Impossibility and universality results for wait-free synchronization // Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing. — PODC '88. — New York, NY, USA : ACM, 1988. — P. 276–290. — URL: http://dx.doi.org/10.1145/62546.62593.
77. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects / Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. — John Wiley & Sons, Inc., 2013. — Vol. 2.
78. Осовский С. Нейронные сети для обработки информации. Пер. с польского. — М.: Финансы и статистика, 2004. — С. 344.
79. Haykin S. S. Neural Networks and Learning Machines. Neural networks and learning machines. — Prentice Hall, 2009. — P. 906. — ISBN: 9780131471399.
80. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. — М.: Мир, 1992.
81. Роберт К. Основные концепции нейронных сетей/Пер. с англ. — М.: Издательский дом «Вильямс», 2001.
82. Андерсон Т. Статистический анализ временных рядов. — Мир, 1976.
83. Крисилов В. А., Чумичкин К. В., Кондратюк А. В. Представление исходных данных в задачах нейросетевого прогнозирования // М.: МИФИ. — 2003. — С. 184–191.
84. Ханк Д., Райтс А. Д., Уичерн Д. У. Бизнес-прогнозирование. — М.: Издательский дом «Вильямс», 2003.

85. Солдатова О. П., Семенов В. В. Применение нейронных сетей для решения задач прогнозирования // Электронный журнал «Исследовано в России». — 2006. — Т. 135. — С. 1261–1269.
86. Тихонов Э. Е. Методы прогнозирования в условиях рынка: учебное пособие. — Невинномысск: Изд-во СевКавГТУ, 2006.
87. Cover T. M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition // Electronic Computers, IEEE Transactions on. — 1965. — June. — Vol. EC-14, no. 3. — P. 326–334. — URL: http://dx.doi.org/10.1109/PGEC.1965.264137.
88. Golub G. H., Reinsch C. Singular value decomposition and least squares solutions // Numerische Mathematik. — 1970. — Vol. 14, no. 5. — P. 403–420. — URL: http://dx.doi.org/10.1007/BF02163027.
89. Tarassenko I., Roberts S. Supervised and unsupervised learning in radial basis function classifiers // IEE Proceedings — Vision, Image and Signal Processing. — 1994. — August. — Vol. 141. — P. 210–216. — URL: http://digital-library.theiet.org/content/journals/10.1049/ip-vis_19941324.
90. Fahlman S. E., Lebiere C. The cascade-correlation learning architecture // Advances in Neural Information Processing Systems 2 / Ed. by David S. Touretzky. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1990. — P. 524–532. — URL: http://dl.acm.org/citation.cfm?id=109230.107380.
91. Linear algebra package. — 2014. — URL: http://www.netlib.org/lapack/.
92. Востокин С. В., Дорошин А. В., Артамонов Ю. С. Программный комплекс templet. организация прикладных вычислений на базе суперкомпьютера «сергей королёв» // Управление движением и навигация летательных аппаратов: Сборник трудов XVII Всероссийского семинара по управлению движением и навигации летательных аппаратов: Часть I. — 2014.
93. Головашкин Д. Л., Сойфер В. А. Анализ прохождения электромагнитного излучения через дифракционную линзу // Автометрия. — 1999. — Т. 35, № 6. — С. 119–121.
94. Курушина С. Е. Аналитическое исследование и численное моделирова-

ние контрастных диссипативных структур в поле флуктуаций динамических переменных // Известия вузов. Прикладная нелинейная динамика. — 2009. — Т. 6. — С. 125–138. — URL: http://elibrary.ru/item.asp?id=13022218.

95. Doroshin A. V. Heteroclinic dynamics and attitude motion chaotization of coaxial bodies and dual-spin spacecraft // Communications in Nonlinear Science and Numerical Simulation. — 2012. — Vol. 17, no. 3. — P. 1460 – 1474. — URL: http://dx.doi.org/10.1016/j.cnsns.2011.07.031.
96. Рихтер Д. Windows для профессионалов: Программирование для Windows 95 и Wndows NT 4 на базе Win32 API/Пер. с англ. — М.: Издательский отдел «Русская редакция» ТОО «Channel Trading Ltd», 1997.
97. Solomon D. A., Russinovich M. Inside Microsoft Windows 2000. — 3rd edition. — Redmond, WA, USA : Microsoft Press, 2000. — ISBN: 0735610215.
98. Страуструп Б. Язык программирования C++, 3-е изд./Пер. с англ. — СПб-М.: Невский Диалект-Издательство БИНОМ, 1999.
99. Butenhof D. R. Programming with POSIX Threads. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. — ISBN: 0-201-63392-2.
100. Dagum L., Menon R. Openmp: an industry standard api for shared-memory programming // Computational Science & Engineering, IEEE. — 1998. — Vol. 5, no. 1. — P. 46–55.
101. Message passing interface forum. — 2014. — URL: http://www.mpi-forum.org/.
102. Pvm (parallel virtual machine). — 2014. — URL: http://www.csm.ornl.gov/pvm/.
103. Xdr: External data representation standard. — 2014. — URL: http://www.ietf.org/rfc/rfc1832.txt.
104. Хьюз К., Хьюз Т. Параллельное и распределенное программирование с использованием C++ Пер. с англ. — М.: Издательский дом «Вильямс», 2004.
105. Ellis T. M. R., Philips I. R., Lahey T. M. Fortran 90 Programming. International computer science series. — Addison-Wesley, 1994. — ISBN: 9780201544466.

106. Virding R., Wikström C., Williams M. Concurrent Programming in ERLANG (2Nd Ed.) / Ed. by Joe Armstrong. — Hertfordshire, UK, UK : Prentice Hall International (UK) Ltd., 1996. — ISBN: 0-13-508301-X.
107. Chandy K. M., Kesselman C. Compositional c++: Compositional parallel programming // Languages and Compilers for Parallel Computing / Ed. by Utpal Banerjee, David Gelernter, Alex Nicolau, David Padua. — Springer Berlin Heidelberg, 1993. — Vol. 757 of Lecture Notes in Computer Science. — P. 124–144. — URL: http://dx.doi.org/10.1007/3-540-57502-2_44.
108. Kale L. V., Krishnan S. Charm++: a portable concurrent object oriented system based on c++ // SIGPLAN Not. — 1993. — October. — Vol. 28, no. 10. — P. 91–108. — URL: http://dx.doi.org/10.1145/167962.165874.
109. Lehmann D., Rabin M. O. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem // Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL '81. — New York, NY, USA : ACM, 1981. — P. 133–138. — URL: http://dx.doi.org/10.1145/567532.567547.
110. Courtney A. Phantom: An interpreted language for distributed programming // Proceedings of the USENIX Conference on Object-Oriented Technologies on USENIX Conference on Object-Oriented Technologies (COOTS). — COOTS'95. — Berkeley, CA, USA : USENIX Association, 1995. — P. 7–7. — URL: http://dl.acm.org/citation.cfm?id=1268098.1268106.
111. Modula-3 resource page. — 2014. — URL: http://modula3.org/.
112. Limited I. Occam programming manual. Prentice-Hall international series in computer science. — Prentice-Hall International, 1984.
113. Gelernter D. Generative communication in linda // ACM Trans. Program. Lang. Syst. — 1985. — January. — Vol. 7, no. 1. — P. 80–112. — URL: http://dx.doi.org/10.1145/2363.2433.
114. Common object request broker architecture (corba). — 2014. — URL: http://www.omg.org/spec/CORBA/.
115. Bal H. E., Kaashoek M. F., Tanenbaum A. S. Orca: a language for parallel

programming of distributed systems // Software Engineering, IEEE Transactions on. — 1992. — March. — Vol. 18, no. 3. — P. 190–205. — URL: http://dx.doi.org/10.1109/32.126768.

116. Java technology. — 2014. — URL: http://www.oracle.com/technetwork/java/index.html.
117. Microsoft .net. — 2014. — URL: https://www.microsoft.com/net.
118. Extensible markup language (xml). — 2014. — URL: http://www.w3.org/XML/.
119. Foster I., Kesselman C. The Grid 2: Blueprint for a New Computing Infrastructure. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2003. — ISBN: 1558609334.
120. Foster I., Kesselman C., Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations // International journal of high performance computing applications. — 2001. — August. — Vol. 15, no. 3. — P. 200–222. — URL: http://dx.doi.org/10.1177/109434200101500302.
121. Smarr L., Catlett C. E. Metacomputing // Grid Computing. — John Wiley & Sons, Ltd, 2003. — P. 825–835. — ISBN: 9780470867167. — URL: http://dx.doi.org/10.1002/0470867167.ch37.
122. Overview of the i-way: Wide-area visual supercomputing / Thomas A. Defanti, Ian Foster, Michael E. Papka et al. // The International Journal of Supercomputer Applications and High Performance Computing. — 1996. — Vol. 10, no. 2/3. — P. 123–131. — URL: http://dx.doi.org/10.1177/109434209601000201.
123. Beowulf: Параллельные вычисления на linux-кластерах. — 2014. — URL: http://parallel.ru/computers/reviews/beowulf.html.
124. Mell P., Grance T. The nist definition of cloud computing // National Institute of Standards and Technology. — 2009. — Vol. 53, no. 6. — P. 50.
125. Артамонов Ю. С. Коллективная разработка приложений для высокопроизводительных вычислений // Перспективные информационные технологии в научных исследованиях, проектировании и обучении (ПИТ 2012): труды научно-технической конференции с международным участием и элементами научной школы для молодежи, посвященной 40-летию кафедры информационных систем и технологий СГАУ / Под ред. С. А. Прохоров ; Самарский государственный аэрокосмический университет. — Са-

мара : Изд-во СНЦ РАН, 2012. — С. 64–67.

126. Hunt G. C., Larus J. R. Singularity: Rethinking the software stack // SIGOPS Oper. Syst. Rev. — 2007. — April. — Vol. 41, no. 2. — P. 37–49. — URL: http://dx.doi.org/10.1145/1243418.1243424.
127. Шмидт Д., Хьюстон С. Программирование сетевых приложений на C++. — М.: ООО «Бином-Пресс», 2004. — Т. 2. — С. 400.
128. Земсков Ю. В. Qt4 на примерах. — БХВ-Петербург, 2008. — С. 608. — ISBN: 9785977502566.
129. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. — O'Reilly Media, Inc., 2007. — ISBN: 0596514808.
130. Dijkstra E. W., Scholten C. S. Termination detection for diffusing computations // Information Processing Letters. — 1980. — Vol. 11, no. 1. — P. 1–4. — URL: http://dx.doi.org/10.1016/0020-0190(80)90021-6.

ПРИЛОЖЕНИЕ А. Акты внедрения

УТВЕРЖДАЮ
Проректор по науке и инновациям СГАУ
д.т.н., профессор
_____А.Б. Прокофьев
«___»_____2011 г.

АКТ

об использовании результатов научных исследований
аспиранта Литвинова Владимира Геннадьевича,
выполненных под руководством профессора
кафедры информационных систем и технологий
Востокина Сергея Владимировича
в научной деятельности факультета летательных аппаратов ФГБОУ ВПО
«Самарский государственный аэрокосмический университет имени
академика С.П. Королева (национальный исследовательский университет)»
(СГАУ)

Комиссия Самарского государственного аэрокосмического университета в составе декана факультета летательных аппаратов СГАУ, д.т.н., профессора Кирпичёва В.А, заместителя декана факультета летательных аппаратов, к.т.н., доцента кафедры теоретической механики Алексеева А.В., доцента кафедры конструкции и проектирования летательных аппаратов Кременецкой М.Е., ассистента кафедры конструкции и проектирования летательных аппаратов, к.т.н. Черняева А.В. составила настоящий акт о том, что результаты научных исследований аспиранта Литвинова В.Г., выполненных под руководством профессора кафедры информационных систем и технологий Востокина С.В. на тему «Методы распараллеливания и эффективного отображения на мультипроцессорные архитектуры с GPU ускорителем и учетом операций ввода вывода для схем типа «портфель задач» использованы в научно-исследовательской работе кафедры конструкции и проектирования летательных аппаратов.

Библиотека TBag программного комплекса автоматизации параллельного программирования Graphplus templet для кластера «Сергей Королев» (режим доступа http://graphplus.ssau.ru/) использовалась в качестве примеров распараллеливания ресурсоемких оптимизационных задач в рамках НИР по реализации на территории Самарской области инновационных и научно-технических проектов, направленных на содействие реализации

программы развития Государственного образовательного учреждения высшего профессионального образования «Самарский государственный аэрокосмический университет имени академика С. П. Королёва (национальный исследовательский университет)» в 2011-2013 годах.
Госконстракт № 283 от 09.09.2011.
Мероприятие «Разработка технологий и кадровое сопровождение цифрового проектирования и перепроектирования изделий авиационно-космической техники (в том числе ракет-носителей «Союз» и двигателей семейства НК)».
Раздел 1 «Разработка экспериментально-цифровой технологии проектирования агрегатов аэрокосмической техники из композиционных материалов».

Председатель комиссии
декан факультета летательных аппаратов СГАУ,
д.т.н., професор В.А. Кирпичев

Члены комиссии:

Зам. декана факультета летательных аппаратов,
доцент кафедры ТМ, к.т.н. А.В.Алексеев

Доцент кафедры КиПЛА, к.т.н. М.Е. Кременецкая

Ассистент кафедры КиПЛА, к.т.н. А.В. Черняев

У Т В Е Р Ж Д А Ю

Проректор по информатизации СГАУ

_____ Пашков Д.Е.

А К Т

Об использовании результатов диссертационной работы Литвинова В.Г. «Разработка и применение типовых решений для распараллеливания алгоритмов численного моделирования»

Настоящим подтверждаю, что Литвинов Владимир Геннадьевич выполнял работы в качестве участника проекта «Разработка комплекса технологий использования ресурсов суперкомпьютера «Сергей Королёв» в целях развития инновационной и научно-образовательной среды университета». Работы по данному проекту выполнены в 2013 году в рамках мероприятия 3.3 Программы развития СГАУ «Разработка и реализация в учебном процессе и научных исследованиях инновационных информационных технологий и подготовка методических материалов по использованию суперкомпьютерной и инфотелекоммуникационной грид-среды университета».

Результаты диссертации Литвинова В.Г. использованы в разделе «Разработка пакета программ автоматизации поисково-переборных расчётов и методики применения сервиса параллельных вычислений в инженерных расчётах» указанного проекта. Литвиновым В.Г. разработаны: пакет программ автоматизации поисково-переборных расчётов и методика их применения в инженерных расчётах.

Пакет программ автоматизации поисково-переборных расчётов находится в публичном доступе в сети интернет по адресу templet.ssau.ru/templet. Компоненты программного комплекса, разработанные Литвиновым В.Г., успешно прошли этап тестовой эксплуатации и с 2015 года используются для проведения лабораторных практикумов по курсам «Параллельное программирование», «Высокопроизводительные вычислительные системы», «Методы параллельных вычислений» в Самарском государственном аэрокосмическом университете им. академика С.П. Королёва (НИУ) и Самарском государственном техническом университете.

Руководитель проекта,
заместитель начальника
управления информатизации
и телекоммуникаций СГАУ
к.т.н., доцент _____ Д.В. Еленев

13.04.2015.

ПРИЛОЖЕНИЕ Б. Свидетельство о регистрации программы

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2014661813

Программа обучения и оптимизации структуры гиперрадиальной нейронной сети с использованием типового решения «портфель задач»

Правообладатель: ***Литвинов Владимир Геннадьевич (RU)***

Автор: ***Литвинов Владимир Геннадьевич (RU)***

Заявка № **2014619770**

Дата поступления **30 сентября 2014 г.**

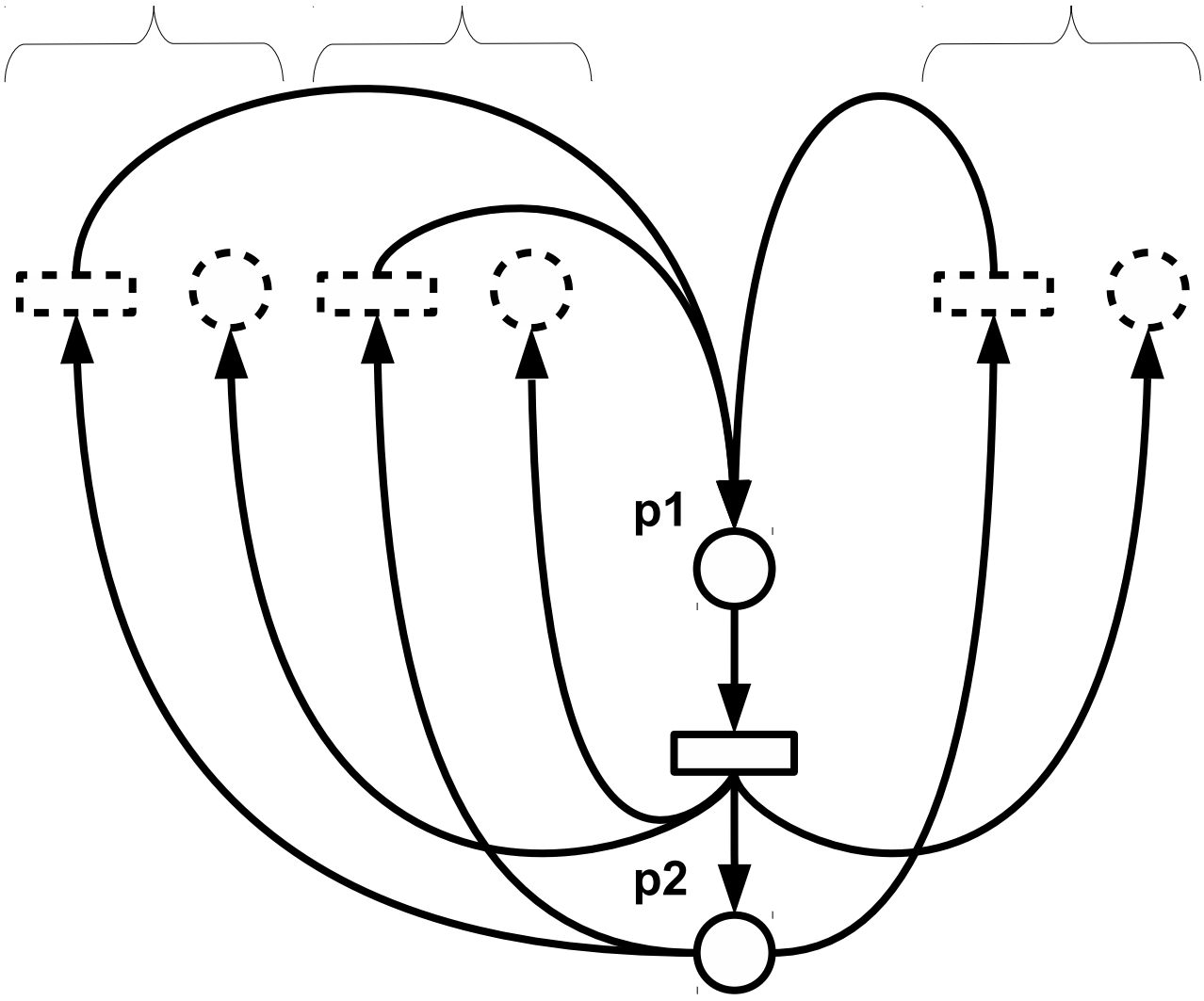
Дата государственной регистрации в Реестре программ для ЭВМ ***13 ноября 2014 г.***

Врио руководителя Федеральной службы по интеллектуальной собственности

Л.Л. Кирий

Представление процессов в виде сети Петри

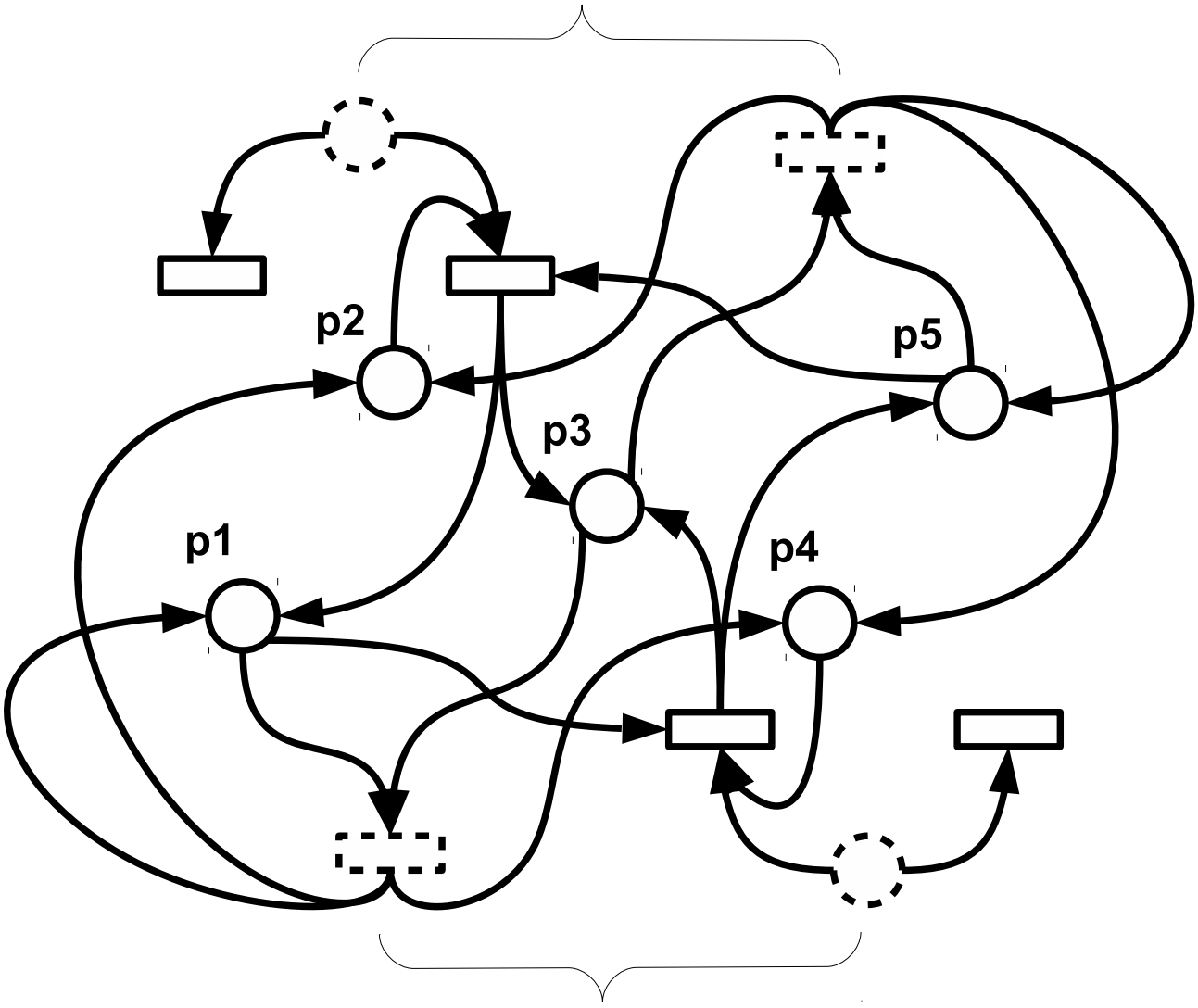
Пары переход-место сопрягаются с местом
и переходом входящих и/или исходящих дуг



Начальное состояние процесса: p1=0 ^ p2=1

Представление каналов в виде сети Петри

Сопрягается с местом и переходом вершины,
из которой исходит дуга, обозначающая канал



Сопрягается с местом и переходом вершины,
в котороруюю входит дуга, обозначающая канал

Активное начальное состояние канала:

p1=1 ∧ p2=0 ∧ p3=1 ∧ p4=0 ∧ p5=0

Пассивное начальное состояние канала:

p1=1 ∧ p2=1 ∧ p3=0 ∧ p4=1 ∧ p5=0