Российский государственный университет нефти и газа имени И.М. Губкина

На правах рукописи

Васильев Алексей Владимирович

Разработка методов и программных средств реализации компонентной вычислительной среды для параллельного и распределённого моделирования режимов систем газоснабжения

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание учёной степени
кандидата технических наук

Научный руководитель
д.т.н., доц.
Сарданашвили Сергей Александрович

Москва – 2013

Оглавление

Введение

Актуальность темы исследования. Единая система газоснабжения (ЕСГ) России управляется распределённой иерархической Автоматизированной системой диспетчерского управления (АСДУ).

В работе диспетчерских служб (ДС) всех уровней активно применяются многочисленные программно-вычислительные комплексы (ПВК) систем поддержки принятия диспетчерских решений (СППДР), а также компьютерные диспетчерские тренажеры, основанные на моделировании стационарных и нестационарных режимов систем газоснабжения (СГ).

В настоящее время реализуется проект модернизации АСДУ ЕСГ России. Для информационного обеспечения всех подсистем АСДУ создаются централизованные источники актуализированных данных и распределённые средства доступа к ним – единое информационное пространство (ЕИП). Для вычислительного обеспечения потребностей АСДУ создан ряд мощных центров обработки данных (ЦОД).

Новые требования предъявляются и к ПВК СППДР:

- быстрое решение наиболее вычислительно-сложных задач моделирования СГ на основе параллельных и распределённых вычислений и с использованием всех доступных вычислительных ресурсов: от персональных компьютеров ДС до ЦОД;
- работа в многопользовательском, распределённом режиме, соответствующем иерархии АСДУ;
- интеграция с единым информационным пространством;
- обеспечение согласованности результатов работы различных ПВК, организация их взаимодействия на основе стандартных протоколов информационного обмена.

Реализацию перечисленных требований кардинально затрудняет ряд исторически сложившихся факторов:

– использование в каждом ПВК собственного расчётного модуля, взаимодействие с которым осуществляется по закрытому интерфейсу, затрудняет согласование результатов работы различных ПВК, их совместное использование, интеграцию с ЕИП;
– ориентация расчётных модулей на последовательную организацию вычислительных процессов, что не соответствует современным требованиям;
– монолитность архитектуры расчётных модулей – длительная разработка на основе процедурного подхода привела к высокой степени взаимопроникновения между их элементами, что крайне затрудняет осуществление модернизации.

Один из путей преодоления проблем, связанных с перечисленными факторами – разработка унифицированного обеспечения иерархической, распределённой СППДР программно-вычислительными ресурсами для решения задач ДС всех уровней на базе единой вычислительной среды – единого вычислительного пространства (ЕВП) АСДУ ЕСГ.

В настоящей диссертации спроектирована и реализована такая вычислительная среда.

Степень разработанности темы исследования. В работе отмечен и проанализирован вклад в областях науки, связанных с темой диссертационного исследования, следующих отечественных и зарубежных учёных:

– Альтшуль А.Д., Берман Р.Я., Бобровский С.А., Вольский Э.Л., Галиуллин З.Т., Григорьев Л.И., Константинова И.М., Леонов Д.Г., Меренков А.Н., Митичкин С.К., Новицкий Н.Н., Сарданашвили С.А., Селезнев В.Е., Ставровский Е.Р., Сухарев М.Г., Чарный И.А., Хасилев В.Я., Швечков В.А., Юфин В.А., Яковлев Е.И. и др. в области математических методов, вычислительных алгоритмов моделирования и оптимизации режимов транспорта газа и их программной реализации в ПВК СППДР;

– Воеводин В.В., Таненбаум Э.С., Флинн М., Хоар Т., Хьюит К., Шохам Й., и др. в области теоретических основ и программных технологий программирования параллельных, распределённых, высокопроизводительных вычислений, создания распределённых систем;
– Буч Г., Гамма Э., Лисков Б., МакКоннел С., Мартин Р., Страуструп Б., Фаулер М., и др. в области объектно-ориентированного программирования, принципов, шаблонов и методологий проектирования программного обеспечения.

Цель и задачи диссертационной работы. Целью диссертационной работы является разработка моделей, методов, алгоритмов и программная реализация мультизадачной вычислительной среды комплекса моделирования режимов систем газоснабжения на основе применения технологий параллельных и распределённых вычислений.

Для достижения поставленной цели были решены следующие задачи:

1. Анализ текущего состояния и направлений развития АСДУ ЕСГ России, единого информационного пространства системы диспетчерского управления ЕСГ, архитектур построения ПВК моделирования режимов СГ, технологий программирования параллельных и распределённых вычислений.
2. Проектирование компонентной среды организации параллельных и распределённых вычислительных процессов решения задач моделирования режимов СГ и алгоритмов создания вычислительных сервисов в этой среде.
3. Разработка архитектурных решений и алгоритмов для вычислительных сервисов решения задач численного моделирования режимов СГ, адаптации модели режимов СГ к фактическим режимам.
4. Программная реализация мультизадачной вычислительной среды и вычислительных сервисов ее поддержки.

5. Проверка корректности и измерение производительности разработанного программного обеспечения в однопользовательском, сетевом, распределённом режимах.
6. Разработка методологии перехода ПВК СППДР к использованию вычислительных сервисов, созданных на базе мультизадачной вычислительной среды, осуществление интеграции с одним из ПВК СППДР в тестовом режиме.

При этом задачи, поставленные в данной диссертации, в области программных средств обеспечения систем поддержки принятия диспетчерских решений АСДУ Единой системы газоснабжения России решаются впервые.

Научная новизна. Основные научные результаты работы состоят в следующем.

1. Разработаны архитектурные решения, методы, алгоритмы и программная реализация компонентной вычислительной среды параллельного и распределённого моделирования режимов СГ, обеспечивающей эффективное использование вычислительных ресурсов в однопользовательском, сетевом, распределённом режимах, механизмы интеграции с ПВК СППДР на основе открытого предоставления вычислительных сервисов.
2. Разработаны специализированные сервисы, управляющие вычислительными процессами и ресурсами, вычислительные сервисы моделирования режимов СГ, позволяющие многократно ускорить решение расчётных задач, реализуя параллельные вычисления общего назначения на графических картах.
3. Разработан вычислительный сервис решения многоуровневых итерационных задач, организующий параллельное моделирование множества однородных и разнородных СГ в распределённой среде; показана его эффективность при использовании «облачных» вычислительных ресурсов *Amazon Web Services Elastic Compute Cloud*.

Практическая ценность работы заключается в создании базовой мультизадачной вычислительной среды для решения наиболее сложных расчётных задач моделирования режимов распределенных СГ большой размерности, позволяющей повысить эффективность организации вычислительных процессов решения задач СППДР на основе создания единого вычислительного пространства АСДУ.

Разработанная компонентная вычислительная среда параллельного и распределённого моделирования режимов СГ впервые позволяет реализовать решение задач моделирования связанных технологических режимов смежных разнородных технологических комплексов: системы добычи, магистрального транспорта, подземного хранения и распределения газа.

Методология и методы исследования. В работе применены следующие основные методы:

- математического и численного моделирования и решения расчётных задач диспетчерского управления для систем газоснабжения;
- объектно-ориентированного проектирования, программирования гетерогенных параллельных вычислений, создания распределённых систем.

Положения, выносимые на защиту:

- Разработаны архитектурные решения в области построения мультизадачной вычислительной среды, основанной на реализации сервис-ориентированного подхода средствами компонентного программного обеспечения промежуточного уровня и явном управлении вычислительными ресурсами.
- Осуществлена алгоритмическая и программная реализация ядра вычислительной среды – сервисов управления типизированными комплектами вычислительных ресурсов и сервиса диспетчеризации вычислительных ресурсов.
- Разработаны архитектурные решения, осуществлена алгоритмическая и программная реализация вычислительного сервиса моделирования режимов систем газоснабжения (СГ), организующего параллельное

моделирование объектов СГ, в том числе с использованием вычислений общего назначения на графических картах.

– Разработаны архитектурные решения, осуществлена алгоритмическая и программная реализация вычислительного сервиса решения многоуровневых вычислительных итерационных задач ДУ СГ в распределённой среде.

Степень достоверности и апробация результатов. Основные результаты диссертационной работы докладывались на следующих конференциях и семинарах:

1. 9-я Всероссийская конференция молодых учёных, специалистов и студентов по проблемам газовой промышленности «Новые технологии в газовой промышленности» (Москва, РГУ Нефти и газа, 2011 г.).
2. 9-я Всероссийская научно-техническая конференция «Актуальные проблемы развития нефтегазового комплекса России» (Москва, РГУ Нефти и газа, 2012 г.).
3. 66-я Международная молодёжная научная конференция «Нефть и газ 2012» (Москва, РГУ Нефти и газа, 2012 г.).
4. Всероссийский научный семинар с международным участием «Математические модели и методы анализа и оптимального синтеза развивающихся трубопроводных и гидравлических систем» (Вышний Волочек, УПЦ Залучье, 2012 г.).
5. 5-я Международная конференция «Компьютерные технологии поддержки принятия решений в диспетчерском управлении газотранспортными и газодобывающими системами Диском 2012» (Москва, ВНИИГаз, 2012 г.).

Публикации. Материалы диссертации опубликованы в 8 печатных работах [**15**], [**16**], [**17**], [**18**], [**19**], [**20**], [**21**], [**40**] из них 3 статьи в рецензируемых журналах [**15**], [**16**], [**20**] и 5 тезисов докладов.

Личный вклад автора. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в опубликованные

работы. Подготовка к публикации полученных результатов проводилась совместно с соавторами, причем вклад диссертанта был определяющим. Все представленные в диссертации результаты получены лично автором.

Структура и объем работы. Диссертация состоит из введения, трех глав, выводов, литературы из 160 наименований и 1 приложения. Работа изложена на 123 страницах основного текста и 19 страницах приложения. Текст работы содержит 38 рисунков и 11 таблиц.

Глава 1. Анализ проблем в области разработки и развития компьютерных программно-вычислительных комплексов моделирования систем газоснабжения

1.1. Общая характеристика автоматизированной системы диспетчерского управления единой системой газоснабжения России

Единая система газоснабжения (ЕСГ) России представляет собой сложный, единый иерархически-управляемый производственно-технологический комплекс, включающий в себя системы добычи, транспорта, подземного хранения газа, трубопроводные распределительные системы. В состав ЕСГ входят 160,4 тыс. км магистральных газопроводов и отводов, 215 линейных компрессорных станций с общей мощностью газоперекачивающих агрегатов в 42 тыс. МВт, 6 комплексов по переработке газа и газового конденсата, 25 объектов подземного хранения газа.

Основные подсистемы ЕСГ и их взаимосвязи представлены на рисунке 1.1.

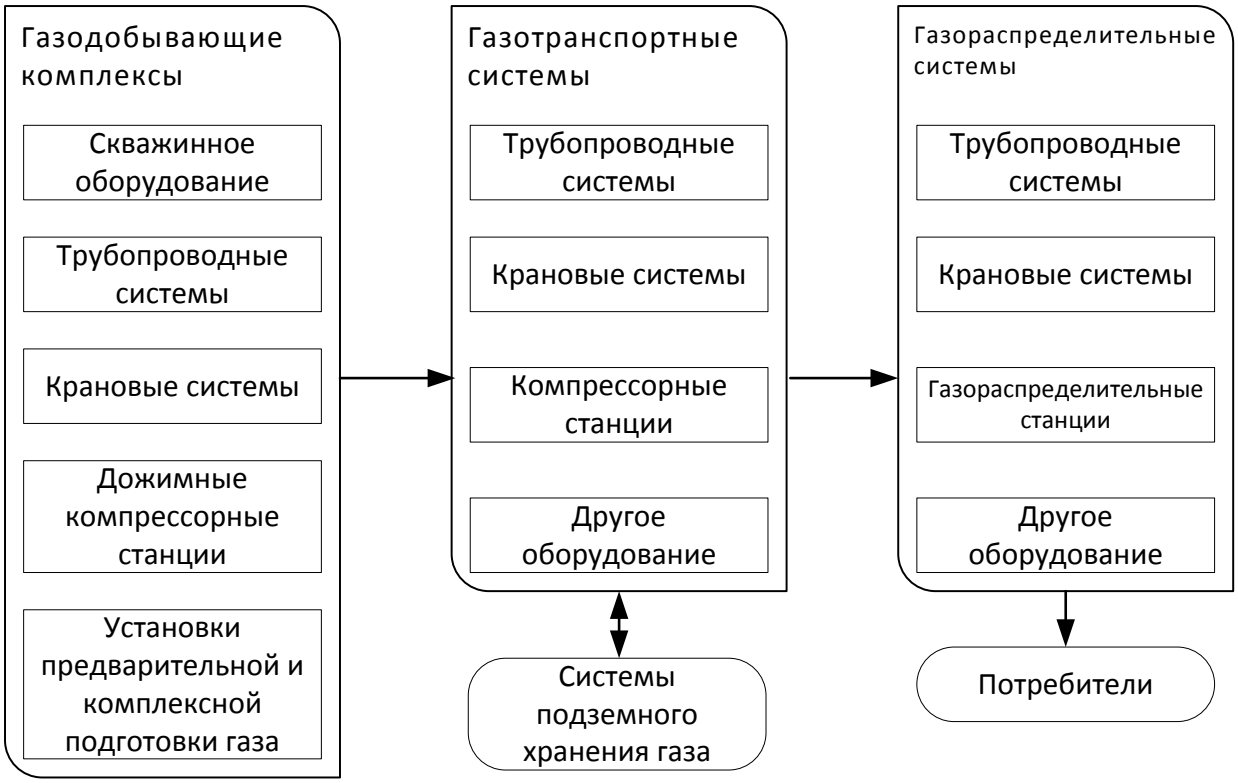

Рисунок 1.1 – Основные подсистемы ЕСГ и их взаимосвязи

Особенностью ЕСГ как объекта управления является необходимость одновременно и централизованного, и децентрализованного управления.

Высокая концентрация мощностей и технологическая взаимозависимость подсистем ЕСГ требует централизации диспетчерского управления потоками газа с выполнением на верхнем уровне не только балансовых, но и режимно-технологических функций, обеспечивающих системную надёжность газоснабжения.

Структура диспетчерского управления ЕСГ подразделяется на четыре основных уровня.

1. ЦПДД – Центральный производственно-диспетчерский департамент.
2. ПДС ЭО – производственно-диспетчерская служба эксплуатирующей организации.
3. ДП ЛПУ МГ – диспетчерский пункт линейного производственного управления магистральными газопроводами.
4. Комплексы телеизмерений и телеуправлений (SCADA-системы). (SCADA – Supervisory Control and Data Acquisition)

Реализация диспетчерского управления ЕСГ России осуществляется посредством Автоматизированной системы диспетчерского управления (АСДУ), укрупнённая схема которой представлена на рисунке 1.2.

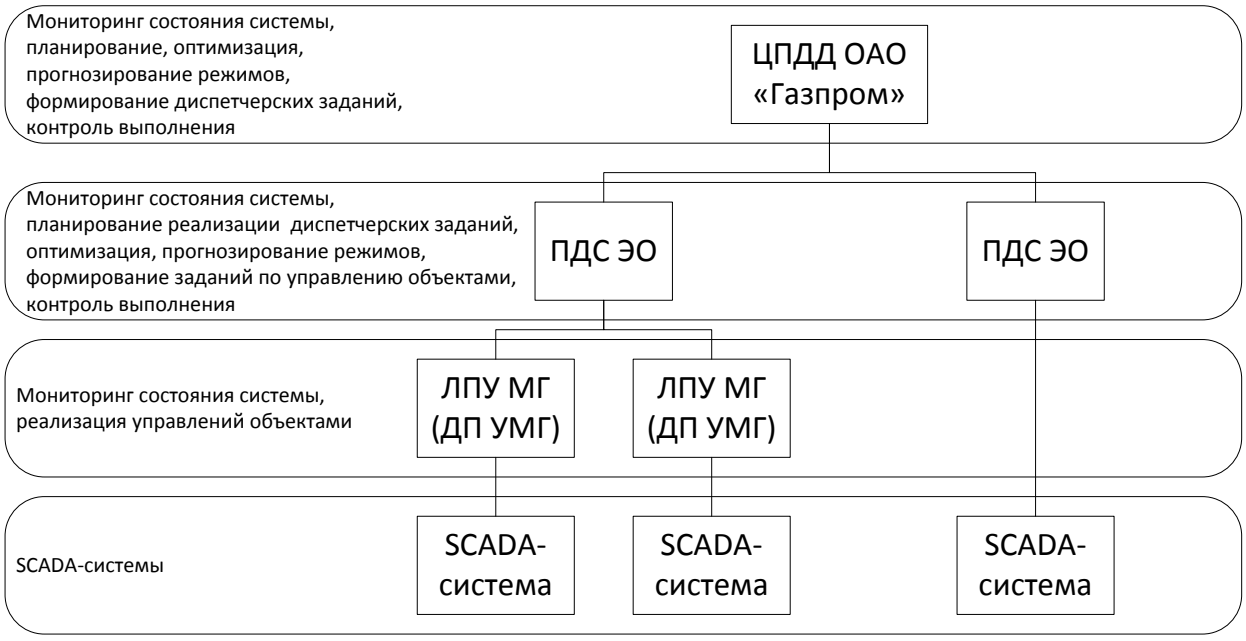

Рисунок 1.2 – Схема диспетчерского управления ЕСГ

Основное назначение АСДУ – информационная, вычислительная, организационно-управленческая поддержка диспетчерского управления системами газоснабжения с целью выполнения контрактных обязательств с максимальной возможной эффективностью.

Каждый уровень ДУ имеет определённые цели, функциональные задачи и располагает средствами их реализации, основанными на принципе вертикальной подчинённости субъектов диспетчерского управления: субъект ДУ каждого уровня достигает реализации поставленных перед ним целей посредством выдачи диспетчерских заданий и оперативных распоряжений субъекту ДУ нижестоящего уровня.

Процессы, цели и средства диспетчерского управления технологическими процессами в ЕСГ зависят от уровня и представлены в таблице 1.1.

Таблица 1.1 – Процессы, цели и средства различных уровней ДУ

Уровень ДУ	Процесс ДУ	Цель	Средство
ЦПДД	Управление запасами газа в системах газоснабжения	Создание наибольшего оптимального запаса газа в СГ	Планирование объёмов добычи, подземного хранения и транспорта потоков газа соответствующими системами газоснабжения
ПДС ГТО	Управление потоками газа в границах ответственности ГДО, ГТО	Распределение потоков по ГТС ГТО согласно параметрам на границах ГТО, установленных ЦПДД	Планирование оптимальных режимов работы технологических объектов СГ, необходимых для выполнения заданий ЦПДД
ДС ЛПУ МГ	Управление режимами работы технологических объектов ГТС в зоне ответственности ЛПУ МГ	Обеспечение заданного ПДС ГТО режима работы участка ГТС	Реализация необходимых режимов работы технологических объектов СГ в зоне ответственности ЛПУ МГ
Оперативный персонал	Непосредственное управление оборудованием конкретного технологического объекта	Создание заданных ДС ЛПУ МГ режимов работы оборудования	Обеспечение работы оборудования в заданных режимах, обеспечение контроля работы оборудования и условий его безопасной эксплуатации

Диспетчерское управление объединяет все технологические процессы функционирования систем газоснабжения и связывает решения производственных и коммерческих задач.

В технологической структуре газового бизнеса ДУ относится к производственному блоку и является связующим звеном между всеми технологическими процессами, обеспечивающими газоснабжение потребителей.

В управленческой структуре эксплуатирующей организации ДУ является связующим звеном между системами прямого управления объектами технологического процесса и корпоративными системами управления, оперирующими финансово-экономическими показателями.

Процесс подготовки и принятия диспетчерских решений

Процесс подготовки диспетчерских решений основан на непрерывном и циклическом выполнении следующих процедур:

- оценка текущего технологического режима системы газоснабжения;
- оценка состояния, в которое требуется перевести систему;
- формирование альтернативных вариантов решения;
- проведение технологических расчетов;
- многокритериальный анализ и сопоставление альтернативных вариантов решения;
- разработка, передача и контроль выполнения диспетчерских заданий по корректировке режимов газоснабжения.

Подготовка диспетчерских решений осуществляется на основе результатов анализа больших объемов режимно-технологической информации и непрерывного расчетного режимно-технологического мониторинга систем газоснабжения. Выполнение указанных функций и процедур требует применения специализированных компьютерных средств поддержки принятия диспетчерских решений.

Схема циклического процесса автоматизированной подготовки и принятия диспетчерских решений представлена на рисунке 1.3.

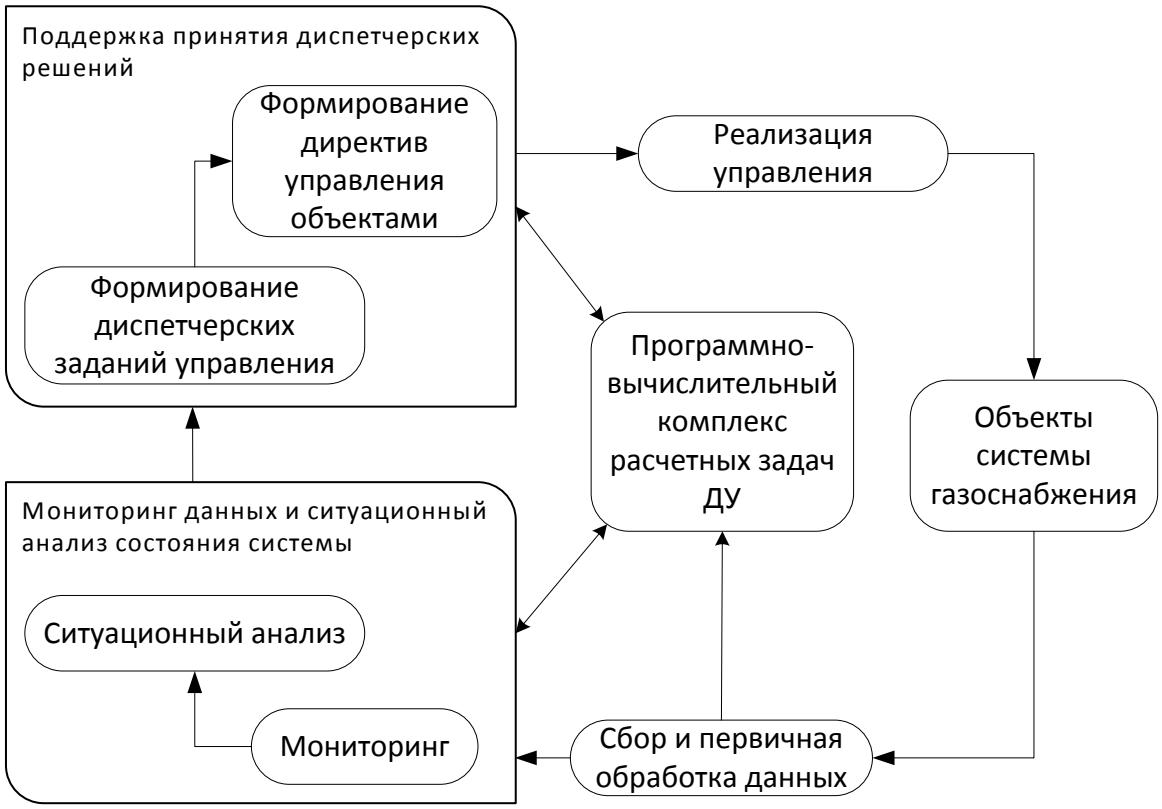

Рисунок 1.3 – Схема автоматизированного процесса подготовки и принятия диспетчерских решений

Центральную роль в процессе принятия диспетчерских решений играет система поддержки принятия диспетчерских решений (СППДР) – совокупность программно-вычислительных средств решения режимно-технологических задач ДУ, обеспечивающая диспетчерские службы информацией, необходимой для принятия решений и формирования диспетчерских заданий в соответствии с их функциями.

СППДР является инструментом для подготовки обоснованных решений по управлению системами газоснабжения на всех уровнях ДУ ЕСГ, предназначена для расчетной и информационно-аналитической поддержки персонала ДС в решении задач ДУ с целью сокращения временных, организационных и ресурсных затрат на формирование решений по управлению системами газоснабжения [**4**], [**49**], [**84**].

Основная функциональная задача СППДР состоит в предоставлении диспетчерскому персоналу средств ситуационного и системного анализа

состояния технологических процессов в системах газоснабжения и оценки последствий возможных управляющих воздействий, в том числе:

- идентификации и диагностики текущего состояния технологического процесса и объектов управления на основе мониторинга и моделирования режимов;
- прогнозирования и оптимального планирования технологических процессов;
- многокритериального сопоставления результатов многовариантных расчетов для выбора эффективных диспетчерских решений;
- обеспечения адаптивного управления системами газоснабжения.

В составе СППДР используются многочисленные программно-вычислительные комплексы различного функционального назначения.

1.2. **Программно-вычислительные комплексы поддержки принятия диспетчерских решений**

Программно-вычислительные комплексы (ПВК), составляющие СППДР подразделяются на ряд категорий:

- общесистемные программные средства;
- информационно-аналитические комплексы;
- расчётные режимно-технологические комплексы.

Общесистемные программные средства СППДР

К общесистемным программным средствам СППДР относятся информационно-технологические компьютерные комплексы, обеспечивающие структурированное хранение данных и шаблонное предоставление диспетчерскому персоналу сеансовой и суточной диспетчерско-технологической информации, необходимой для управления ЕСГ, в частности, следующие системы:

- электронный диспетчерский журнал и журнал по газотранспортному обществу (ГТО) и газодобывающему предприятию (ГДО), адаптированный под программно-технические средства резервного диспетчерского центра ЦПДД (на основе web-технологий);
- графическая база данных (БД) технологических, расчетных и потоковых схем объектов и подсистем ЕСГ (технологические схемы формата Autocad);
- графическая БД технологических схем газотранспортных систем (ГТС) ЕСГ, ближнего и дальнего зарубежья (технологические схемы формата Autocad);
- геоинформационные системы (ГИС);
- система паспортизации объектов и оборудования подсистем ЕСГ;
- программные комплексы планирования, синхронизации, согласования, контроля проведения ремонтных работ;
- программные комплексы формирования, контроля и анализа выполнения диспетчерских заданий.

Общесистемные средства обеспечивают подготовку исходных данных для прикладных программных средств СППДР: информационно-аналитических и расчётных режимно-технологических программных комплексов.

Информационно-аналитические комплексы

Информационно-аналитические комплексы СППДР обеспечивают структурированное хранение данных, направленный поиск и представление в произвольной форме информации для анализа текущих и ретроспективных режимно-энергетических показателей работы ЕСГ.

В СППДР применяются следующие информационно-аналитические комплексы:

- ИУСДУ – информационно-управляющая система диспетчерского управления (отображение информации по производственному технологическому процессу в реальном времени);
- АССПООТИ – автоматизированная система сбора, передачи, обработки и отображения технологической информации (передача «двухчасовых сводок», «суточных закрытий», «месячных отчётов»);
- ССД «Инфотех» – система сбора, обработки и хранения данных о технологических объектах транспорта газа (около 180 отчётов по различным направлениям деятельности);
- системы сбора и передачи диспетчерско-технологической информации по компрессорным станциям в ЦПДД;
- системы сбора и передачи данных реального времени с газовых промыслов в ЦПДД.

Расчётные режимно-технологические программные комплексы

Расчётные ПВК СППДР решают ряд задач моделирования, состав которых определяется типом системы газоснабжения:

1) для газодобывающих систем:
   - моделирование и оптимизация режимов систем: скважины – кусты – промысловый газосборный коллектор – цех сепарации газа – дожимная компрессорная станция (ДКС) – установка комплексной подготовки газа

(УКПГ) – межпромысловый газосборный коллектор – вход в головные компрессорные станции (КС) магистральных газопроводов (МГ);

- моделирование режимов трубопроводных систем, в том числе многофазных потоков скважинной продукции, трубопроводных систем конденсатопроводов;
- моделирование и оптимизация режимов работы ДКС;
- моделирование режимов работы установок предварительной и комплексной подготовки газа к транспорту;
- оптимизация режимов отбора газа с промыслов;

2) для газотранспортных обществ:
   - планирование и контроль балансов и поставок газа;
   - моделирование стационарных и нестационарных режимов транспорта газа по ГТС;
   - оптимизация расчетных технологических режимов работы объектов и подсистем ЕСГ с учетом различных критериев эффективности, технологических и иных ограничений;
   - расчет плановых и нормативных затрат топливно-энергетических ресурсов;
   - расчет показателей энергоэффективности технологических объектов ЕСГ;
   - решение потоковых задач, включая расчет маршрутов и протяженности поставок газа потребителям;
   - расчет технически возможной пропускной способности магистральных газопроводов (МГ) и трубопроводных систем (ТС) при заданных условиях;
3) для подземного хранения газа:
   - составление и контроль в реальном времени баланса ПХГ;
   - оперативный расчет запаса активного газа в ПХГ;
   - оценочный расчет пластового давления в объектах хранения;
   - оценочный расчет водного фактора по скважинам и по ПХГ в целом;
   - расчеты расхода газа на собственные технологические нужды и потери;
   - сравнение плановых и реально достигнутых значений расхода газа на собственные технологические нужды;
   - оценочный расчет потоков газа и пропускной способности наземного технологического комплекса;

- оценка давления газа в точке подключения ПХГ к магистральному газопроводу;
- оперативная оценка максимально возможной производительности ПХГ по закачке и отбору газа;

4) для распределения газа:
   - гидравлический расчет сетей газораспределения;
   - оценка пропускной способности сетей газораспределения;
   - оценка загрузки сетей газораспределения;
   - прогноз газопотребления;
5) отдельные технологические расчёты;
   - свойства газа и газового потока;
   - запас газа в трубопроводной системе;
   - время опорожнения и заполнения трубопровода;
   - условия образования гидратов;
   - скорость газа и время прохождения внутритрубного устройства;
   - расчет режимно-энергетических параметров газоперекачивающего агрегата (ГПА);
   - расчет режимно-тепловых параметров аппарата воздушного охлаждения (АВО).

Не смотря на различия состава расчётных задач в режимно-технологических комплексах СППДР, все они делятся на следующие категории в зависимости от условий эксплуатации:

- автономный режим (off-line);
- асинхронный режим обработки данных SCADA-систем (on-line);
- режим реального времени или следящий режим, синхронизированный со SCADA-системой (off-line);
- режим будущего времени, прогнозный режим (future-time).

Перечисленные режимы применения ПВК отличаются источниками данных и наборами решаемых задач, которые представлены в таблице 1.2.

Таблица 1.2 – Источники данных и категории расчетных задач (off-line, on-line, real-time, future-time)

Режим моделирования	**Источник данных**	**Категории расчетных задач**
Автономный *(Off-line)*	Данные могут быть заданы пользователем, получены из БД. Возможно интерактивное изменение параметров задачи – имитация управления технологическими объектами, аварийных ситуаций, взаимодействия со SCADA-системой	Интерактивная проверка различных управляющих воздействий, аварийных и нештатных ситуаций с целью решения задач планирования стационарных и нестационарных режимов при различных расчётных схемах и минимально достаточных наборах информации
Фактический *(On-line)*	Используются данные из БД телеизмерений, телесигнализация и расчётные показатели нижних уровней автоматизации (SCADA) за любой доступный период предыстории. Так как используются стохастические данные, необходима их статистическая предобработка	Базовая статистическая обработка данных телеизмерений; контроль адекватности и адаптация моделей к реальным режимам; оценка фактического состояния технологических объектов; краткосрочное прогнозирование и планирование режимов
Реальное время *(Real-time)*	Используются данные из БД реального времени за последний доступный период предыстории	Анализ реального состояния и оперативной диагностики режимов работы СГ
Будущее время *(Future-time)*	Используются прогнозные значения краевых параметров газового потока на входах/выходах расчётной СГ, прогноз газопотребления, прогноз состояния оборудования, участвующего в технологическом процессе	Моделирование развития режима системы на будущие периоды времени, решение задач оперативного, текущего и перспективного планирования

Решение большинства расчетных задач ДУ основано на применении базовых вычислительных процедур моделирования режимов систем газоснабжения и отдельных их элементов.

В частности, применительно к задачам ДУ ГТС, базовый расчетный комплекс можно представить в виде иерархической схемы взаимосвязанных процедур моделирования режимов как отдельных объектов технологического процесса перекачки газа: трубопроводы, краны, газо-перекачивающие агрегаты (ГПА), аппараты воздушного охлаждения газа (АВО), так и подсистем магистрального транспорта газа: трубопроводные системы (ТС), компрессорные

станции (КС), магистральные газопроводы (МГ), газотранспортные системы (рисунок 1.4).

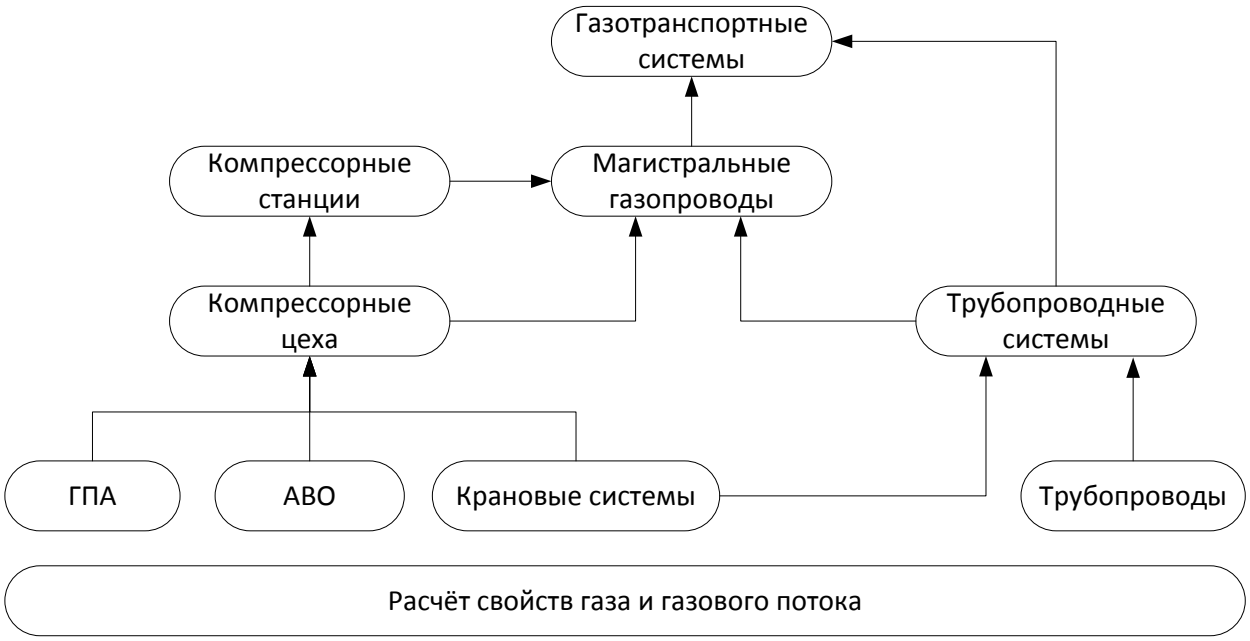

Рисунок 1.4 – Взаимосвязи блоков задач off-line моделирования ГТС

Ниже представлен основной состав расчётных процедур моделирования режимов ГТС для различных моделируемых объектов:

- трубопроводы:
  - расчёт параметров газового потока для стационарного, квазистационарного, нестационарного режима; режима опорожнения трубы; режима при полном или частичном разрыве трубы; режима при гидратообразовании;
- крановые системы:
  - расчёт параметров газового потока через краны: байпасирования потока, редуцирования давления, кран регулятор;
- аппараты воздушного охлаждения (АВО):
  - расчёт параметров газового потока на выходе АВО;
  - расчёт количества секций и вентиляторов, обеспечивающих заданные параметры теплового режима;
- газоперекачивающие агрегаты (ГПА) и их группы при многоступенчатом компримировании:
  - расчёт области допустимых режимов;

- расчёт режимно-энергетических параметров;
- расчёт производительности;
- расчёт оборотов ГПА, обеспечивающих заданные параметры режима;

– компрессорные цеха:
- расчёт режимных и топливно-энергетических параметров;
- расчёт текущей минимальной и максимальной возможной производительности;
- расчёт оптимальной схемы и оборотов ГПА, обеспечивающих заданные параметры режима (для ГПА с газотурбинной установкой);
- расчет режима работы с учетом антипомпажной защиты;
- расчёт степени открытия крана регулятора, обеспечивающего заданные параметры режима (для ГПА с электроприводом);

– компрессорные станции:
- расчёт режимных и топливно-энергетических параметров;
- расчёт оптимальных схем и оборотов ГПА КЦ, обеспечивающих заданные параметры режима;

– трубопроводные системы:
- расчёт параметров газового потока;
- расчет запаса (аккумулированного) газа;
- идентификация эмпирических параметров (коэффициентов гидравлической эффективности и теплообмена с окружающей средой);
- расчет пропускной способности;

– магистральный газопровод:
- расчёт параметров газового потока;
- идентификация эмпирических параметров моделей ТС и КЦ;
- расчёт режимных топливно-энергетических балансовых и экономических показателей;
- расчёт текущей и максимальной возможной пропускной способности;
- расчёт оптимального стационарного режима транспорта газа;
- планирование нестационарного режима;

– газотранспортная система:
- расчёт параметров газового потока;
- идентификация эмпирических параметров моделей ТС и КЦ;

- расчёт режимных топливно-энергетических балансовых и экономических показателей;
- интерактивное моделирование нестационарного режима;
- планирование нестационарных, переходных режимов.

Типовые составы процедур on-line и real-time представлены на рисунках 1.5 и 1.6.

В задачах обработки данных режима on-line:

- для решения используются данные, полученные из базы данных телеизмерений (типа SCADA), то есть данные фактического состояния системы и процесса транспорта газа за любой доступный период предыстории;
- решение проводится асинхронно с системой телеизмерений как в автоматическом режиме, так и по запросу пользователя.

В режиме *on-line*, на основе массивов замеров параметров состояния ГТС и режимно-технологических параметров потоков газа должны, прежде всего, выполняться вычислительные процедуры, обеспечивающие решение следующих задач:

- базовая статистическая обработка данных телеизмерений (фильтрация, сглаживание, расчет точечных и интервальных оценок, построение стохастической модели процесса);
- контроль адекватности и адаптация моделей к реальным режимам транспорта газа;
- оценка фактического состояния технологических объектов ГТС, ГПА, КЦ, ТС;
- краткосрочное прогнозирование и планирование режимов.

Рисунок 1.5 – Типовой состав процедур режима on-line

<u>В задачах обработки данных режима real-time:</u>

– для решения используются данные, полученные из базы системы телеизмерений, однако выборка этих данных обязательно заканчивается последним временным слоем опроса датчиков;
– решение выполняется синхронно с системой телеизмерений, то есть новый цикл обработки данных (решения задач) начинается сразу после поступления в базу очередного массива замеряемых параметров и обязательно заканчивается до следующего сеанса опроса датчиков системой телеизмерений;

Рисунок 1.6 – Типовой состав процедур режима real-time

– решение, как правило, выполняется автоматически в следящем режиме, без вмешательства пользователя;
– поскольку тактовая частота SCADA-системы определяется различными факторами: динамикой изменения состояния объектов системы (в аварийных ситуациях это секунды), динамикой процесса газопередачи (в штатном состоянии это минуты), то состав решаемых задач может также меняться в соответствии с режимом работы SCADA;
– используются высокоскоростные, устойчивые вычислительные алгоритмы, пусть даже с некоторой потерей точности, которую можно восполнить своевременной адаптацией моделей к реальным параметрам режима.

Одним из важнейших блоков обработки данных телеизмерений в режимах *on-line* и *real-time* является:

- расчет параметров газовых потоков (давление, температура, расход газа) для всех объектов газотранспортной системы (ГТС), для которых замеры отсутствуют;
- решение различных режимно-технологических задач оперативного управления, планирования режимов и так далее.

При этом решение большинства из этих задач основано на использовании расчетных процедур Базового комплекса моделирования режимов (Рисунок 1.4).

Основу представленных схем расчетных задач составляет модульная, объектно-ориентированная, распределенная технология, что соответствует требованиям и возможностям ее интеграции в многоуровневые распределенные автоматизированные системы диспетчерского управления, а также возможностям информационной поддержки расчетных задач на каждом уровне управления.

Программно-вычислительные комплексы СППДР, как правило, состоят из следующих подсистем:

- подсистема управления данными;
- подсистема расчетных модулей решения режимно-технологических задач ДУ;
- подсистема управления вычислительным процессом, на основе библиотеки расчётных модулей;
- интерактивный графический интерфейс пользователя;
- подсистема информационного и вычислительного взаимодействия с внешними программными комплексами.

Создание расчётных моделей включает в себя следующие основные этапы:

- разработка (выбор) математических моделей режимов объектов, составляющих технологическую систему;
- алгоритмизация математических моделей режимов объектов и их интеграция в алгоритмическую модель режима технологической системы;
- формализация алгоритмических моделей в виде компьютерной программы;
- выбор множества исходных и множества расчетных параметров компьютерной программы;

– выбор источников исходных данных, программных средств чтения, передачи и размещения данных в ПВК моделирования режимов;
– создание вычислительного процесса выполнения компьютерной программы;
– размещение расчетных параметров, анализ, визуализация,
– оценка адекватности и корректности расчетных режимов фактическим.

Расчетные модули программно-вычислительных комплексов моделирования режимов систем газоснабжения являются наиболее высокотехнологичными и наукоемкими подсистемами ПВК, поскольку являются результатом многолетних научных исследований в области применения математических методов и вычислительных алгоритмов решения сложных режимно-технологических задач. Они обеспечивают остальные подсистемы СППДР, в частности, информационно-аналитические комплексы, вычислительными сервисами и составляют вычислительное ядро СППДР.

Основная база математических методов моделирования, оптимизации прогнозирования режимов систем газоснабжения была разработана многими исследователями, среди которых следует отметить работы Альтшуля А.Д., Бермана Р.Я., Бобровского С.А., Вольского Э.Л., Галиуллина З.Т., Константиновой И.М., Меренкова А.П., Селезнева В.Е., Сухарева М.Г., Ставровского Е.Р., Сарданашвили С.А., Чарного И.А., Хасилева В.Я., Юфина В.А., Яковлева Е.И. [**1**], [**6**], [**7**], [**8**], [**9**], [**10**], [**12**], [**23**], [**35**], [**36**], [**44**], [**45**], [**46**], [**50**], [**59**], [**65**], [**67**], [**68**], [**69**], [**74**], [**75**], [**76**], [**78**], [**89**], [**90**], [**96**].

Активному использованию ЭВМ в научных институтах и диспетчерских службах газотранспортной отрасли способствовали работы, которые заложили теоретический и алгоритмический фундамент для решения задач на основе методов численного моделирования и оптимизации эксплуатационных режимов магистральных газопроводов совместно с компрессорными станциями, расчета максимальной производительности газопроводов, статистической обработки замеряемых данных, адаптации моделей к фактическим режимам и так далее.

С развитием и внедрением в отрасль и практику диспетчерского управления измерительных систем более активно стали появляться исследования в области статистической обработки данных, прогнозирования временных рядов, идентификации параметров и адаптации моделей, параметрической диагностики состояния объектов и систем транспорта газа [**2**], [**14**], [**25**], [**26**], [**32**], [**34**], [**37**], [**60**], [**66**], [**70**], [**71**], [**77**], [**79**], [**80**], [**86**], [**87**].

Значительный вклад в разработку информационных и автоматизированных систем диспетчерского управления, в разработку компьютерных тренажерных комплексов для диспетчерских служб внесли авторы: Григорьев Л.И., Панкратов В.С., Берман Р.Я., Митичкин С.К., Леонов Д.Г., Швечков В.А. [**27**], [**28**], [**29**], [**30**], [**31**], [**38**], [**40**], [**41**], [**42**], [**43**], [**47**], [**48**], [**55**], [**56**], [**57**], [**58**], [**62**], [**63**], [**64**], [**91**], [**92**], [**93**], [**94**].

1.3. **Проблемы и задачи разработки и развития ПВК СППДР**

Действующая АСДУ ЕСГ имеет ряд исторически обусловленных недостатков, связанных с тем, что она складывалась в течение многих лет из различных разнородных компонентов, без единой программной платформы:

- отсутствует единая модель данных и единая НСИ по всему комплексу диспетчерских задач;
- для решения многих задач ДУ недостаточен состав и объём информации, генерируемой и передаваемой в АСДУ ЕСГ;
- слабо используются современные информационные технологии и инструменты обработки информации, управления БД, взаимодействия БД, сбора и представления данных пользователям;
- низок уровень интеграции со смежными существующими и вновь разрабатываемыми информационно-управляющими системами ЭО;
- действующая АСДУ ЕСГ не обеспечивает в необходимом объёме автоматизацию решения функциональных задач диспетчерского управления, не может обеспечить дальнейшего развития системы в соответствии с принятыми Стратегией информатизации и Целевой системной архитектурой АСДУ.

Архитектурная схема действующей АСДУ ЕСГ приведена на рисунке 1.7.

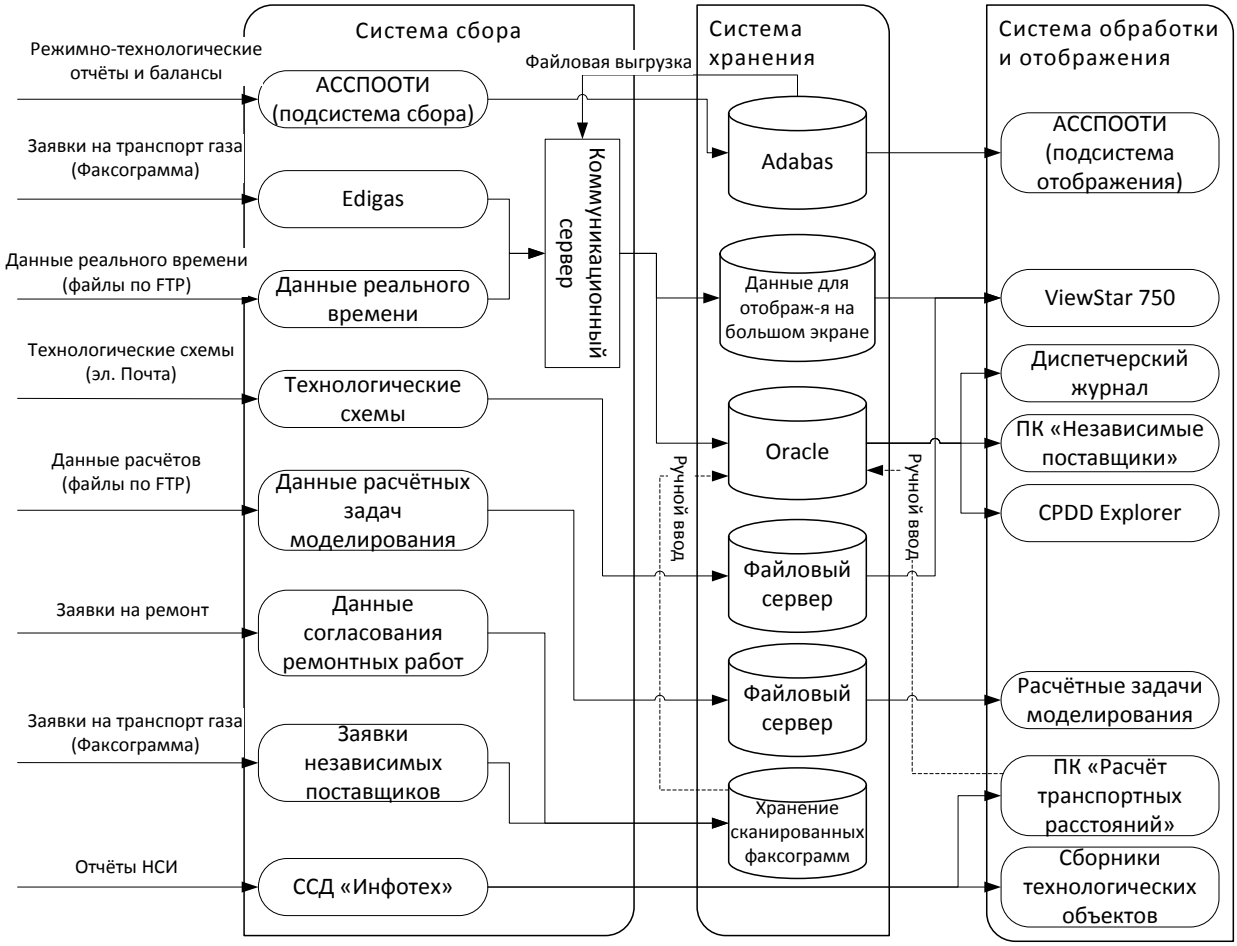

Рисунок 1.7 – Архитектура действующей АСДУ ЕСГ

Для преодоления отмеченных выше недостатков в настоящее время осуществляет проект модернизации АСДУ ЕСГ [**3**], [**72**].

Основными целями проекта модернизации АСДУ ЕСГ являются: повышение надёжности выполнения контрактных обязательств, повышение эффективности и прозрачности диспетчерского управления ЕСГ.

Эти цели предполагается достичь за счёт:

- применения единых стандартов передачи диспетчерской информации;
- расширения информационного пространства и использования программных комплексов;
- применения современных методов автоматизации процесса сбора, обработки, хранения и анализа диспетчерской информации, использования информационных систем, обеспечивающих единство нормативно-справочной информации, баз и хранилищ данных, единых математических моделей и систем отображения, позволяющих обеспечить взаимодействие и

синхронизацию уровней диспетчерского управления и всех систем поддержки принятия диспетчерских решений.

Основная задача проекта модернизации АСДУ ЕСГ состоит в построении принципиально нового ИТ-решения для автоматизации ДУ, основанного на современной промышленной программной платформе за счёт выполнения нижеследующих этапов.

1. Поэтапное замещение функциональностью модернизированной АСДУ ЕСГ систем поддержки принятия диспетчерских решений, существующих и эксплуатируемых в рамках ДУ на данный момент.
2. Создание единого информационного пространства диспетчерского управления, обеспечивающего общие источники данных для принятия управляющих решений на всех иерархических уровнях диспетчерского управления ЕСГ, в том числе:
   - создание единой модели данных с использованием единой нормативно-справочной информации (НСИ), актуализируемой в соответствии с текущим состоянием объектов ЕСГ и используемой на всех уровнях диспетчерского управления;
   - формирование единого хранилища оперативной диспетчерско-технологической, учётной режимной, отчётной балансовой, нормативно-справочной информации.
3. Повышение степени автоматизации процесса сбора, обработки, хранения и представления информации, с учётом поэтапной интеграции с информационно-управляющими системами эксплуатирующих организаций.
4. Реализация функциональной и технологической интеграции функциональных бизнес-процессов «по горизонтали» (на административном уровне) и по «вертикали», и улучшение за счёт этого координации между структурными подразделениями и эксплуатирующими организациями.
5. Создание предпосылок для дальнейшего поэтапного развития и совершенствования АСДУ ЕСГ.

Целевая архитектура модернизированной АСДУ ЕСГ представлена на рисунке 1.8.

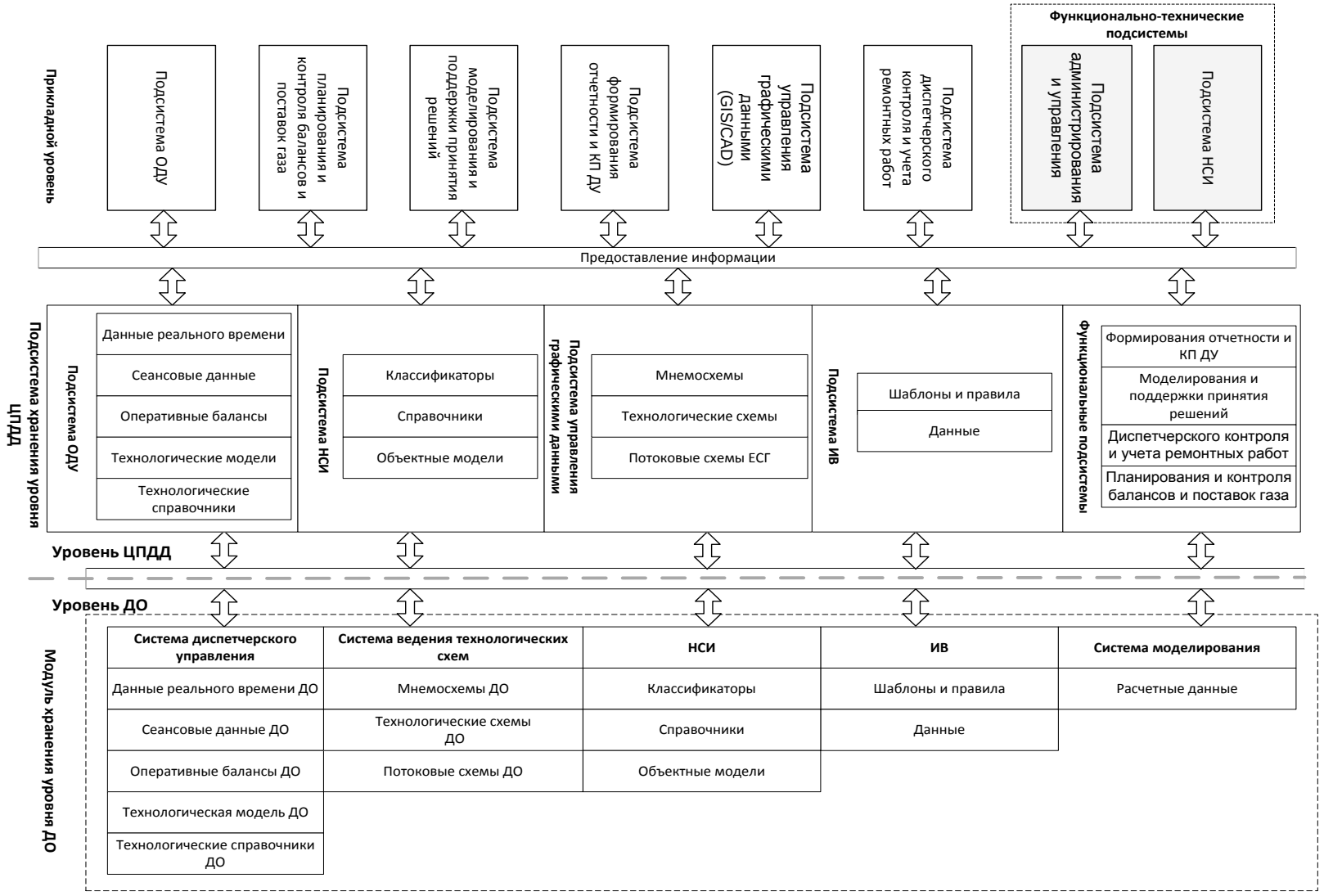

Рисунок 1.8 – Структурная схема информационного обеспечения модернизированной АДСУ ЕСГ

Новые требования к ПВК СППДР и затруднения их выполнения

В связи с модернизацией АСДУ ЕСГ предъявляются новые требования к ПВК СППДР:

- интеграция с единым информационным пространством (ЕИП) и обработка огромных объёмов информации, аккумулируемых в нём;
- функционирование в распределённой среде, в сетевом и распределённом режиме;
- использование стандартных интерфейсов информационных связей между ПВК, унифицированных базовых вычислительных процедур;
- сокращение времени решения вычислительных задач, требующих значительных вычислительных ресурсов, в частности:
  - моделирование систем газоснабжения в режимах on-line и real-time;
  - адаптация моделей СГ к фактическим режимам на различных временных интервалах;
  - расчёт квазиоптимальных режимов работы СГ;
  - расчёт управлений, обеспечивающих переход из одного режима на другой.

Выполнение перечисленных требований затруднено рядом исторически сложившихся проблем.

Применяемые в АСДУ ПВК разрабатываются независимыми коллективами разработчиков, не взаимодействующими между собой. Комплексы автономны, используют собственные базы данных, информационные модели, протоколы информационного обмена. Это приводит к отсутствию единых интерфейсов взаимодействия программных средств как между собой, так и с другими информационными системами.

Во многих ПВК моделируются одни и те же объекты. Из-за отсутствия единых вычислительных сервисов, в каждом ПВК происходит дублирование необходимых моделей, в лучшем случае комплексы взаимодействуют через локальные файлы и базы данных. Дублирование общей функциональности в ПВК затрудняет их разработку, а также согласование результатов их расчётов.

ПВК, используемые в АСДУ, характеризуются следующими основными свойствами:

- ориентированность на локальный, однопользовательский режим работы;
- использование собственных расчётных модулей;
- использование собственных информационных моделей, протоколов и форматов взаимодействия;
- ориентация на последовательную, однопоточную организацию вычислительного процесса;
- длительная разработка на основе процедурного подхода.

Перечисленные свойства привели к тому, что используемые расчётные комплексы стали «монолитными» – все предоставляемые комплексом расчётные задачи – базовый комплекс задач, *off-line*, *on-line* – реализованы в рамках единого исполняемого файла, взаимодействие с которым осуществляется через файловый интерфейс или разделяемую память по закрытому протоколу. Для использования даже самой простой задачи необходимо иметь расчётный комплекс целиком (рисунок 1.9).

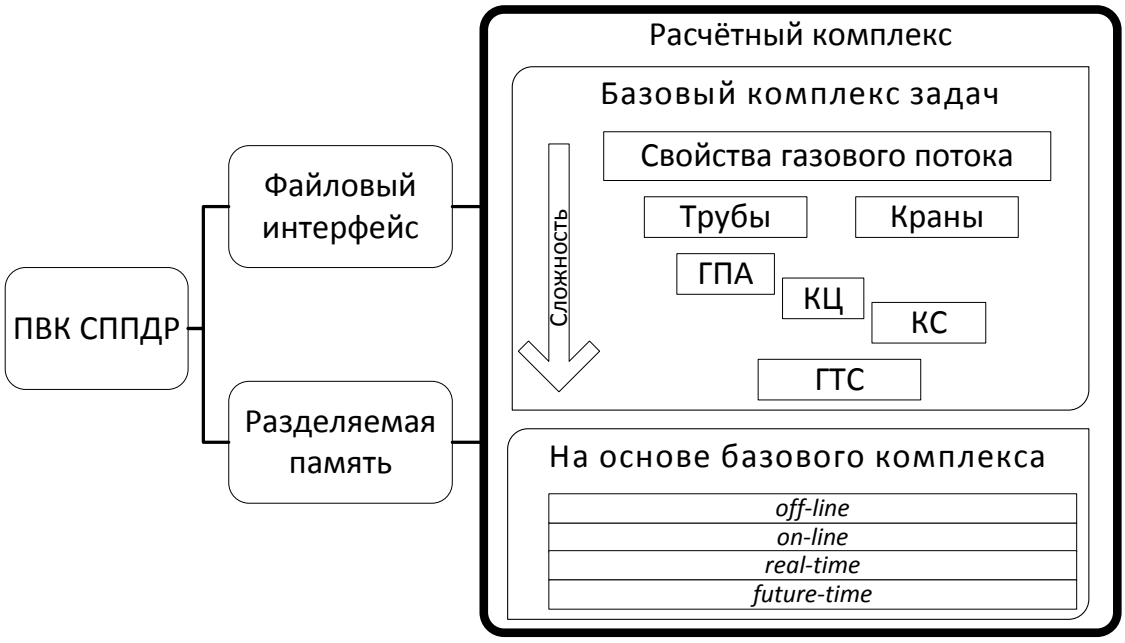

Рисунок 1.9 – Монолитная архитектура ПВК СППДР

Высокая степень взаимопроникновения, взаимозависимости между отдельными процедурами комплекса не позволяет разделить его на отдельные модули и постепенно осуществить модернизацию.

Проблемы, возникающие при модернизации существующих ПВК СППДР, представлены в таблице 1.3.

Таблица 1.3 – Задачи модернизации ПВК СППДР и затрудняющие их выполнение факторы

Задача	**Затрудняющий фактор**
Интеграция с ЕИП	ПВК СППДР используют собственные форматы представления данных и результатов, слабо приспособлены к интеграции с ЕИП и друг с другом
Функционирование в сетевом, распределённом режиме	Расчётные модули ПВК представляют собой монолитные исполнительные файлы, ориентированные на локальный однопользовательский режим работы
Использование единых для всех ПВК СППДР расчётных процедур	Каждый ПВК СППДР использует собственный расчётный комплекс и имеет собственные форматы и протоколы для взаимодействия с ним
Сокращение времени решения сложных вычислительных задач	Для реализации этой задачи требуется использование параллельных и распределённых вычислений, но монолитная архитектура расчётных комплексов не позволяет их внедрить

Преодоление перечисленных затрудняющих факторов для достижения целей модернизации подсистемы ПВК СППДР возможно за счёт разработки единой среды организации параллельного и распределённого вычислительного процесса решения задач СППДР и реализации на её основе Единого вычислительного пространства (ЕВП), обеспечивающего унифицированное обеспечение вычислительных потребностей всех уровней модернизированной АСДУ.

Актуальными нерешёнными проблемами остаются:

– разработка основных проектных и архитектурных решений единой вычислительной среды, отвечающей следующим требованиям:
  - модульность, компонентность, гибкость, развиваемость;

- полная поддержка параллельных вычислений;
- полная поддержка распределённых вычислений;
- открытость: высокая способность к взаимодействию, использованию стандартных протоколов информационного взаимодействия;
- предоставление единых вычислительных сервисов для ПВК СППДР, работающих на всех уровнях АСДУ;

– сравнительный обзор современных информационных технологий реализации архитектурных решений вычислительной среды;
– программная реализация ядра этой среды;
– разработка методики создания вычислительных сервисов в этой среде;
– программная реализация вычислительных сервисов решения конкретных задач СППДР в этой среде;
– разработка методологии интеграции разработанных вычислительных сервисов с ПВК СППДР, осуществление тестовых интеграционных проектов;
– проведение вычислительных экспериментов с разработанными вычислительными сервисами в различных режимах: локальном, сетевом, распределённом.

Следующие главы диссертации посвящены решению указанных проблем.

Выводы

1. Анализ текущего состояния АСДУ ЕСГ и проекта её модернизации показал высокую актуальность разработки моделей, методов, алгоритмов и программной реализации Единого вычислительного пространства ЕВП СППДР, интегрированного с единым информационным пространством модернизированной АСДУ.
2. Монолитная архитектура современных расчётных комплексов ПВК СППДР является основным фактором, затрудняющим достижение целей и решение задач модернизации СППДР на основе современных информационных технологий параллельных и распределенных мультизадачных вычислительных сред.
3. Решение задачи создания ЕВП СППДР выдвигает новые требования к ПВК: возможность работы в сетевом, распределённом, многопользовательском режиме; переход к использованию параллельных и распределённых вычислений; использование различными ПВК согласованных вычислительных процедур.
4. Сформулированы основные задачи исследования, которые необходимо решить в процессе проектирования, разработки и программной реализации единой вычислительной среды, обеспечивающей базу для унифицированного вычислительного обеспечения всех уровней модернизированной АСДУ:
   - проектирование компонентной среды организации параллельных и распределённых вычислительных процессов решения задач моделирования режимов СГ и алгоритмов создания вычислительных сервисов в этой среде;
   - разработка архитектурных решений и алгоритмов для вычислительных сервисов решения задач численного моделирования режимов СГ, адаптации модели режимов СГ к фактическим режимам;
   - программная реализация мультизадачной вычислительной среды и вычислительных сервисов ее поддержки;
   - разработка методологии перехода ПВК СППДР к использованию вычислительных сервисов, созданных на базе мультизадачной вычислительной среды, осуществление интеграции с одним из ПВК СППДР в тестовом режиме.

Глава 2. Разработка компонентной среды организации параллельного и распределённого вычислительного процесса решения расчётных задач СППДР

2.1. Сравнительный анализ и выбор технологий реализации вычислительной среды

Разрабатываемая вычислительная среда должна быть основана на наиболее современных технологиях программирования. Выполнение этого требования возможно на основе проведения анализа и выбора технологий в области параллельного и распределённого программирования.

Анализ и выбор базовых технологий параллельного программирования

Большинство современных вычислительных устройств содержат параллелизм в своей архитектуре, поэтому полное использование их потенциала возможно только на основе программирования параллельных вычислений [**154**], [**155**]. По всей видимости, будущие поколения вычислительных устройств будут использовать параллелизм ещё более интенсивно [**102**], [**143**].

Наиболее распространённые вычислительные устройства параллельной архитектуры – *SMP*-системы (*Symmetric Multi-Processor*), включающие несколько одинаковых процессоров, или процессорных ядер, имеющих доступ к общей памяти.

Всё большую распространённость получают гетерогенные вычислительные системы, в которых одновременно работают несколько различных типов вычислительных устройств, приспособленных для решения специфических задач. В частности, для высокопроизводительных вычислений активно используются высоко-параллельные сопроцессоры [**156**].

Большой вклад в развитие и популяризацию параллельных вычислений внесён В.В. Воеводиным [**22**] и Центром параллельных вычислений [**24**].

Технологии программирования *SMP*-систем

Эта категория представлена многочисленными технологиями: системные средства для работы с процессами и потоками, *Boost Thread, OpenMP, Microsoft PPL (Parallel Primitives Library), Intel TBB (Thread Building Blocks)*, и другие [**22**], [**88**].

Системные средства для работы с процессами, потоками и синхронизацией

Операционные системы, как правило, предоставляют функции для работы с процессами и потоками, необходимые средства меж процессного взаимодействия: файлы, сокеты, очереди сообщений, каналы, разделяемая память, отображаемые на память файлы. Для синхронизации взаимодействия процессов и потоков предоставляются такие механизмы, как семафоры, мьютексы, мониторы Хоара. Перечисленные механизмы доступны для использования через системные вызовы [**61**], [**83**].

Преимуществом использования низкоуровневых системных вызовов является отсутствие накладных расходов, связанных с использованием каких бы то ни было надстроек.

Недостатками является сложность написания программ с использованием только лишь низкоуровневых функций и отсутствие переносимости программ между операционными системами.

Переносимые библиотеки для работы с потоками на примере Boost Thread

Библиотека *Boost Thread* является примером кросс-платформенной библиотеки для работы с потоками и их синхронизацией. За счёт использования в тексте программы предоставляемых библиотекой абстракций, таких как поток, мьютекс, обеспечивается переносимость программы на различные операционные системы [**73**], [**110**], [**111**], [**150**].

Предлагаемые программисту независимые от операционной системы переносимые абстракции реализуются библиотекой при помощи вызова соответствующих низкоуровневых функций операционной системы.

Несмотря на кросс-платформенность такого подхода, абстракция потока остаётся достаточно низкоуровневой и затрудняет создание сложных приложений.

Стандарт параллельного программирования Open MP

OpenMP – это открытый стандарт параллельного программирования многопроцессорных систем с общей памятью [**146**], [**147**]. Существуют реализации этого стандарта для наиболее распространённых операционных систем.

Существенным отличием от программирования в терминах потоков является более высокий уровень абстракций *OpenMP*. Программирование ведётся с использованием высокоуровневых директив компилятора, позволяющих управлять многопоточностью, синхронизацией, классами памяти переменных и так далее.

Синтаксически директивы реализованы таким образом, что компилятор без поддержки *OpenMP* просто проигнорирует их и построит однопоточную программу.

Основными достоинствами *OpenMP* являются широкая распространённость, высокоуровневая модель программирования, совместимость созданных программ с компиляторами без поддержки *OpenMP*.

Microsoft PPL (Parallel Performance Library) и Intel TBB (Thread Building Blocks)

Библиотеки *Microsoft Parallel Performance Library* и *Intel Thread Building Blocks* предоставляют обобщённые многопоточные контейнеры, алгоритмы, позволяют формулировать программу в терминах графа задач, автоматически преобразуемых библиотекой в потоки [**105**], [**148**]. Описание

параллелизма ведётся на логическом уровне: описываются взаимосвязи, зависимости выполняемых задач.

Основные параллельные алгоритмы, предоставляемые библиотеками:

- *parallel_for* – многопоточный цикл for;
- *parallel_foreach* – многопоточный цикл for each;
- *parallel_invoke* – параллельный запуск нескольких задач;
- *parallel_transform* – многопоточный алгоритм трансформации;
- *parallel_reduce* – многопоточный алгоритм редукции;
- *parallel_sort* – многопоточный алгоритм сортировки.

Основные многопоточные контейнеры:

- *concurrent_vector* – многопоточный вектор;
- *concurrent_queue* – многопоточная очередь;
- *concurrent_map* – многопоточный ассоциативный массив;
- *concurrent_set* – многопоточное множество.

API этих библиотек частично совместимы. При этом преимуществами *TBB* по сравнению с *PPL* являются кросс-платформенность и более широкая функциональность.

Выбор технологий параллельного SMP-программирования

Сравнительный анализ описанных технологий представлен в таблице 2.1.

Предпочтительно использование наиболее высокоуровневых и переносимых технологий: *OpenMP*, *Intel TBB*. Они взаимно дополняют друг друга: *OpenMP* предоставляет директивы для быстрого распараллеливания существующего кода, в то время как Intel TBB предоставляет многопоточные контейнеры, алгоритмы и средства распараллеливания задач со сложной логикой.

Целесообразно также использование оптимизированных библиотек численных методов, которые используют технологии *SMP*-программирования в реализации своих вычислительных процедур: *Intel MKL (Math Kernel Library), ACML (AMD Core Math Library)* и других [**108**], [**131**].

Таблица 2.1 – Сравнительный анализ технологий SMP-программирования

Технология программирования	**Механизмы программирования**	**Переносимость**
Системные средства для работы с процессами и потоками	Управление потоками, процессами, синхронизацией с использованием низкоуровневых средств операционной системы	Нет
Boost Thread	Управление потоками, процессами, синхронизацией с использованием средств переносимой библиотеки	Да
OpenMP	Автоматическое распараллеливание программы на основе директив	Да
Microsoft PPL	Обобщённые контейнеры и алгоритмы для параллельной обработки данных	Windows
Intel TBB	Обобщённые контейнеры и алгоритмы для параллельной обработки данных	Да

Технологии программирования высоко-параллельных вычислений

Наиболее широко распространённый подход к программированию высоко-параллельных вычислений – технологии программирования *GPGPU (General Purpose computing on Graphics Processing Units)* – вычислений общего назначения на графических картах.

GPGPU отличают следующие особенности:

- использование модели *SIMT (Single Instruction – Multiple Threads)* – единая программа выполняется множеством нитей [**113**];
- пиковая производительность превосходит пиковую производительность многоядерных центральных процессоров.
- для достижения пиковой производительности *GPGPU*-устройства, как правило, требуется одновременная работа большого количества нитей: сотни и тысячи;
- каждой отдельной нити доступно ограниченное количество регистров.

Наибольшая эффективность использования *GPGPU* достигается в решении задач, характеризующихся высокой размерностью с высокой степенью параллелизма.

Наиболее распространённые технологии *GPGPU*-программирования: *CUDA C (Compute Unified Device Architecture), OpenCL, C++ AMP (Accelerated Massive Parallelism), OpenACC.*

CUDA (Compute Unified Device Architecture)

Технология *CUDA*, разработанная компанией *NVIDIA*, является наиболее развитой и мощной технологией программирования высокопараллельных сопроцессоров [**11**], [**109**], [**149**]:

- *CUDA* – наиболее зрелая технология GPGPU: 1-я версия вышла в 2007 году, на сегодняшний день уже вышла 5-я стабильная версия;
- разработано большое количество библиотек для *CUDA*, как численных методов *(cuFFT – Fast Fourier Transform, cuBLAS, cuLA – LAPACK)*, так и для программирования более высокого уровня с использованием шаблонов, подобных *C++ STL – Thrust* [**130**];
- разработаны и свободно предоставляются отладчики и профилировщики *CUDA*-кода – *Parallel Nsight для Microsoft Visual Studio, Nsight Eclipse edition для Mac OS, Linux*;
- по архитектуре *CUDA* разработаны специальные вычислительные устройства для научных вычислений серии *Tesla*, оптимизированные для большого количества вычислений с двойной точностью;
- существуют предложения для доступа к устройствам архитектуры *CUDA* в облаке – например, *Amazon Web Services Elastic Compute Cloud*;
- за счёт того, что модель программирования позволяет непосредственно использовать функции, заложенные в архитектуру устройств, достигается максимальная производительность;
- сама архитектура *CUDA* 5-й версии предлагает уникальный среди *GPGPU*-технологий набор возможностей, таких как динамический параллелизм, линковка объектного кода для *GPU*, прямой доступ к памяти от устройства к устройству (*RDMA*) и так далее.

Основным недостатком *CUDA* является невозможность запуска *CUDA*-программ на устройствах другой архитектуры, других производителей (не *NVIDIA*).

OpenCL

OpenCL – это открытый стандарт программирования гетерогенных параллельных вычислений, поддерживаемый такими производителями как *NVIDIA, Intel, AMD* [**118**]. В отличие от *CUDA*, *OpenCL* не привязан к какой-либо конкретной архитектуре вычислительных устройств.

Для поддержки независимости от платформы *OpenCL* предлагает абстрактную модель управления памяти и потоком выполнения, позволяющие в дальнейшем скомпилировать программу для выполнения на конкретном устройстве.

Главным преимуществом *OpenCL* по отношению к *CUDA* является независимость от платформы, однако это же качество влечёт за собой и недостаток: программы на *OpenCL* как правило проигрывают в производительности *CUDA*, либо нуждаются в настройке под конкретные вычислительные устройства.

OpenCL имеет более сложную модель программирования: по сравнению с *CUDA* исходные тексты аналогичных программ на *OpenCL* получаются сложнее и объёмнее, требуют большего времени на разработку.

Microsoft C++ AMP (Accelerated Massive Parallelism)

C++ AMP – это новое решение для гетерогенных параллельных вычислений от *Microsoft* – оно впервые введено в среде программирования *Visual Studio* 2012 [**119**].

C++ AMP позволяет отметить функцию C++ ключевым словом *restrict*, которое проинструктирует компилятор проверить, что она использует только подмножество инструкций, доступных большинству сопроцессоров.

Затем такая функция может быть использована в обобщённом параллельном алгоритме, входящим в *Microsoft PPL* (*Parallel Performance*

Library). При наличии совместимого сопроцессора, функция будет выполнена на нём – при отсутствии сопроцессора функция будет выполнена с использованием центрального процессора.

Библиотека так же позволяет управлять памятью сопроцессора с использованием специализированного контейнера.

Существенным недостатком *AMP* является возможность исполнения только в операционной системе *Windows*. Не смотря на то, что *C++ AMP* позиционируется как открытая спецификация, реализация существует только для *Windows* и *Visual Studio 2012*.

OpenACC

OpenACC – это новый (представлен в 2012 году) открытый стандарт гетерогенных параллельных вычислений, разработанный на основе идей, лежащих в основе *OpenMP* [**160**]. Этот стандарт предоставляет набор директив компиляторов *C, C++, Fortran* для выделения циклов, данных, участков кода, которые желательно выполнить на сопроцессоре. Благодаря использованию директив, обеспечивается независимость от операционной системы, центрального процессора и сопроцессоа, простота программирования и лёгкость освоения технологии создания гетерогенных приложений.

Предполагается, что *OpenACC*, в качестве расширения, может быть включен в последующие версии *OpenMP*.

Выбор технологий программирования высоко-параллельных вычислений

Сравнение описанных технологий, представленное в таблице 2.2, показывает, что технология *CUDA* – наиболее зрелая, развитая технология программирования *GPGPU* вычислений, основным недостатком которой является возможность выполнения созданных с её использованием программ только на устройствах архитектуры *CUDA*.

Тем не менее, целесообразно ориентироваться на использование этой технологии, поскольку она предоставляет все средства для быстрой разработки наиболее эффективных программ.

Целесообразно также использование оптимизированных численных библиотек, использующих *CUDA*-устройства в реализации вычислительных процедур – их производительность в ряде случаев многократно опережает аналогичные библиотеки для многоядерных процессоров.

Таблица 2.2 – Технологии программирования *GPGPU*-устройств

Технология программирования	**Механизмы программирования**	**Переносимость**	**Зрелость**
NVIDIA CUDA	Расширения языка программирования C для программирования вычислительных устройств архитектуры *CUDA* от *NVIDIA*	ОС: *Windows*, *Linux*, *Mac*; Только устройства архитектуры *CUDA*.	Наибольшая, v1 – 2007 г., v5 – 2012 г.
OpenCL	Программирование гетерогенных параллельных вычислений посредством платформо-независимых конструкций	Различные ОС, устройства.	Средняя, v1 – 2009, v1.2 - 2012
C++ AMP	Программирование гетерогенных параллельных вычислений посредством платформо-независимых конструкций	Только *Windows*, различные устройства	Низкая, v0.99 – 2012
OpenACC	Автоматическое распараллеливание программ на основе директив подобно *OpenMP*	Различные ОС, устройства	Низкая, v1 – 2011

Анализ и выбор технологий распределённого программирования

Под термином «распределённые вычисления» понимается совместное решение общей проблемы несколькими независимыми вычислительными системами.

Параллельные и распределённые вычисления имеют ряд общих черт и областей взаимного проникновения:

- и в параллельных, и в распределённых вычислениях несколько операций выполняются одновременно;
- существуют модели распределённого программирования, позволяющие абстрагироваться от распределённого характера вычислительной системы и программировать в терминах меж процессного взаимодействия, характерного для программирования параллельных вычислений;
- на основе модели передачи сообщений, характерной для программирования распределённых систем, можно программировать и локальные параллельные вычисления.

Главным признаком, отличающим распределённые вычисления от параллельных вычислений, является отсутствие общей разделяемой памяти у взаимодействующих вычислительных систем, в связи с чем базовым механизмом их взаимодействия является передача сообщений.

Фундаментальные теоретические основы распределённых вычислений рассмотрены в работа К. Хьюитта, Й. Шохама, Ч. Хоара, Э. Таненбаума, В. Олифера, Д. Бэкона, [**33**], [**51**], [**52**], [**81**], [**82**], [**126**], [**127**], [**128**], [**129**], [**152**].

Отметим, что технологии и языки программирования, реализующие такие высокоуровневые модели как агентно-ориентированное программирование, модель акторов, слабо приспособлены для реализации высокопроизводительных вычислений [**97**], [**98**], [**116**], [**120**], [**152**].

Сокеты на основе протоколов TCP, UDP

Распространённым низкоуровневым механизмом передачи сообщений по сети является использование интерфейса сокетов, предоставляемого операционной системой, в совокупности с одним из протоколов сетевого взаимодействия, например, *UDP, TCP* [**51**]. На основе этого фундамента строятся более высокоуровневые модели распределённого программирования.

Программирование с использованием сокетов возможно как на основе системных вызовов, так и с использованием кросс-платформенных библиотек. В любом случае этот подход является низкоуровневым, что затрудняет создание сложных систем на его основе.

Single System Image – образ единой системы

SSI-система – это вычислительный кластер, логически работающий как единая система [**103**].

Существуют различные варианты построения *SSI*-систем, например распределённая операционная система или кластер с планировщиком задач.

Распределённая операционная система позволяет представить кластер в виде единой операционной системой и таким образом свести распределённое программирование к программированию в терминах меж процессного взаимодействия.

Кластер с планировщиком задач позволяет пользователям ставить задачи с указанием требуемых для их выполнения вычислительных ресурсов в очередь. Получаемые задачи распределяются планировщиком в соответствии с алгоритмом планирования для максимизации загрузки кластера.

Преимуществом подхода *SSI* являются простота программирования.

Недостатками являются большие накладные расходы при функционировании системы, сложность настройки, поддержки, отладки.

Message Passing Interface (MPI) – интерфейс передачи сообщений

MPI – это стандарт высокоуровневого распределённого программирования на основе передачи сообщений, разработанный для достижения кроссплатформенности, высокой производительности и масштабируемости [**114**], [**134**]. Реализации этого стандарта существуют для основных операционных систем, а так же для высокопроизводительных вычислительных кластеров.

Реализация *MPI* представляет собой библиотеку, обеспечивающую следующие функции:

– отправка сообщений процессам с заданными идентификаторами, широковещательная рассылка, централизованный сбор информации со всех процессов;
– возможность передачи типизированных данных; используются как встроенные типы, например *MPI_INT, MPI_DOUBLE, MPI_CHAR*, так и определяемые пользователем типы;
– возможность синхронизации процессов, осуществления как синхронных, так и асинхронных вызовов, и так далее.

Технология *MPI* используется для программирования как распределённых систем, так и многоядерных процессоров: в процессы *MPI* отображаются на вычислительные ядра центрального процессора.

Технология *MPI* активно используется совместно с технологиями параллельного программирования систем с общей памятью, например, *OpenMP, CUDA*. В этом случае *MPI* организует взаимодействие процессов в распределённой сети, а *OpenMP* и *CUDA* применяются для параллельного программирования отдельных вычислительных узлов.

Компонентные технологии создания распределённых систем (middleware)

Компонентные технологии предоставляют средства для создания систем, состоящих из компонентов, взаимодействующих в распределённой среде по единой схеме, представленной на рисунке 2.1 [**137**], [**157**].

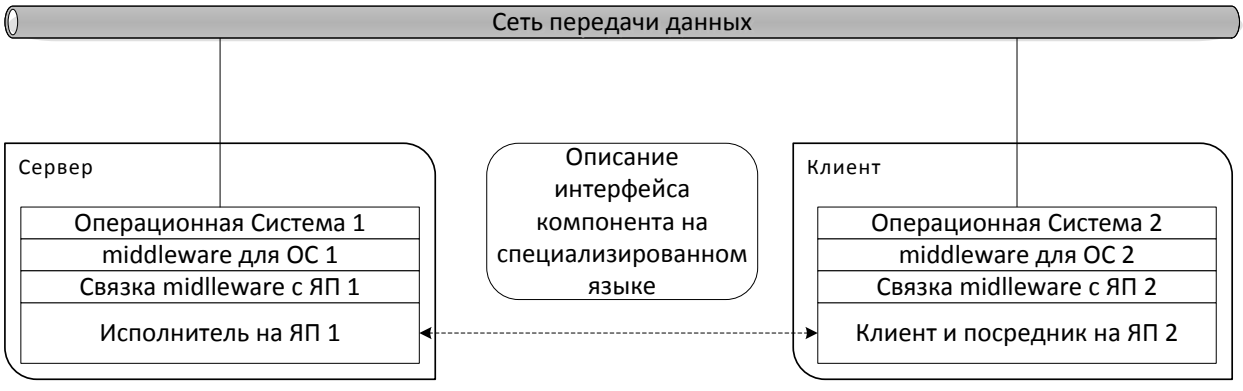

Рисунок 2.1 – Схема работы *middleware*

Каждый компонент имеет интерфейс, описанный на специализированном языке, определяющий перечень предоставляемых им методов.

Реализация компонента и клиента для него осуществляется на одном из языков программирования (ЯП), для которых есть связка с используемым *middleware*. При этом допускается различие языков программирования и операционных систем, используемых для работы взаимодействующих компонентов. Это обеспечивается за счёт того, что компонентное *middleware* играет роль адаптера, преобразуя получаемые и отправляемые данные и связывая их с реализациями компонентов на конкретных языках программирования.

Для использования удалённого компонента – сервера, клиент, как правило, генерирует объект-посредник по интерфейсу сервиса. Объект посредник является локальным для клиента объектом, предоставляющим набор методов, указанный в интерфейсе сервера. При этом вызов метода объекта-посредника проходит по цепочке клиент – посредник – *middleware* клиента – операционная система клиента – сеть передачи данных – операционная система сервера – *middleware* сервера – реализация компонента на сервере – и обратно.

Сравнение технологий распределённого программирования представлено в таблице 2.3.

Таблица 2.3 – Технологии распределённого программирования

Технология	**Механизмы программирования**
Сокеты на основе протоколов *UDP, TCP*	Обмен сообщениям с использованием низкоуровневых механизмов
Single System Image	Представление вычислительного кластера в виде единой системы
Message Passing Interface	Программирование взаимодействующих процессов на основе передачи сообщений
Компонентные технологии	Средства для создания компонентных систем, функционирующих в распределённой среде

Наиболее предпочтительным способом создания модульных систем, работающих в распределённой среде, является использование компонентного программного обеспечения промежуточного уровня.

Технология *MPI* является стандартом де-факто программирования высокопроизводительных кластеров и также может найти применение в решении наиболее сложных вычислительных задач моделирования систем газоснабжения.

Более подробный анализ и выбор технологии компонентного программирования приводится ниже.

Выбор компонентной технологии построения распределённых систем

К распространённым компонентным технологиям построения распределённых систем относятся следующие:

- *Microsoft DCOM (Distributed COM);*
- *Microsoft Remoting;*
- *Microsoft WCF (Windows Communication Foundation);*
- *CORBA (Common Object Request Broker);*
- *Java EE (Enterprise Edition);*
- *ZeroC ICE (Internet Communication Engine).*

Компонентные технологии Microsoft: DCOM (Distributed Component Object Model), Remoting, WCF (Windows Communication Foundation)

DCOM (Distributed COM) – распределённая компонентная технология, разработанная *Microsoft* в качестве расширения существующей не распределённой компонентной модели *COM* (*Component Object Model*) [**158**].

DCOM сохранял интерфейс *COM* для обеспечения обратной совместимости. Так как интерфейс *COM* предполагал использование механизма подсчёта ссылок, то и *DCOM* пошёл по этому же пути. Однако механизма подсчёта ссылок оказался не жизнеспособен в распределённой среде.

Со временем идея подсчёта ссылок в распределённой среде, как и сам *DCOM*, была оставлена и вместе с первой версией платформы *.NET* был выпущен механизм *Remoting* [**141**]. На смену ему впоследствии пришёл механизм *Windows Communication Foundation* (*WCF*), являющийся наиболее современной платформой распределённых вычислений от *Microsoft* [**107**], [**145**].

WCF обладает существенным недостатком: слабой приспособленностью к работе в операционных системах, отличных от Windows.

CORBA (Common Object Request Broker Architecture)

Стандарт *CORBA* был одним из первых стандартов компонентных middleware [**122**]. Однако по разным причинам *CORBA* имеет ряд существенных недостатков [**125**]:

- слишком объёмный стандарт, многое из которого никогда не было реализовано;
- не достаточно широкая поддержка языков программирования: основными поддерживаемыми языками программирования являются *Java* и *C++*, при этом привязка к *C++* весьма многословная и сложная;
- слабая поддержка безопасности;
- отсутствие поддержки для работы с версиями;
- не эффективный протокол удалённого взаимодействия и так далее.

Java EE (Enterprise Edition)

Java EE (*Enterprise Edition*) – это платформа построения распределённых приложений на языке *Java* [**104**]. Платформа *Java EE* проще *CORBA*, поддерживает технологии работы с *Internet*, *web*-сервисы, безопасность, нашла широкое применение в построении масштабируемых распределённых корпоративных приложений.

Существенным недостатком *Java EE* является поддержка только одного языка программирования: *Java*.

Web-сервисы

Web-сервисы – это механизм коммуникации компонентов в *WWW (World Wide Web)*, использующий стандартные для *WWW* протоколы и языки: *HTTP (Hyper Text Transfer Protocol), XML (eXtended Markup Language), WSDL (Web-Service Definition Language), SOAP (Simple Object Access Protocol)* и так далее [**99**], [**106**], [**144**]. Благодаря этому *web*-сервисы обладают максимальной способностью к взаимодействию: если клиент и сервер подключены к интернету, они могут взаимодействовать.

Web-сервисы характеризуются наличием следующих недостатков:

- низкое быстродействие, связанное с использованием текстовых протоколов;
- сложно читаемые и создаваемые определения интерфейсов сервисов на языке *WSDL*;
- отсутствие поддержки многих функций, присутствующих в *WCF, CORBA, Java EE, ICE*.

ZeroC ICE (Internet Communication Engine)

ICE – это современная платформа разработки распределённых компонентных систем [**121**], [**123**], [**124**], [**159**], характеризующаяся следующим:

- поддержка наиболее распространённых языков программирования: *C++, Java, C#* (и другие *.NET* языки, такие как *Visual Basic*), *Python, Ruby, PHP, ActionScript, Objective-C*;
- поддержка наиболее распространённых операционных систем: *Windows, Linux, Mac, iOS, Android*;
- высокое быстродействие бинарного протокола удалённого взаимодействия компонентов системы;
- лаконичный и выразительный язык описания интерфейсов;
- поддержка безопасных протоколов удалённого взаимодействия;
- широкий набор предоставляемых дополнительных инструментов построения распределённых систем:
  - сервис обхода сетевого экрана;

- сервис автоматического распространения обновлений в распределённой системе;
- сервер приложений, обеспечивающий возможность динамического запуска и останова компонент с использованием шаблона проектирования *SuperServer* [**132**];
- сервис хранения объектов в базе данных с автоматической записью и чтением;
- сервис сообщений с возможностью широковещательной рассылки;
- сервис управления грид-системами.

Выбор технологий создания компонентных систем

Основные характеристики перечисленных компонентных технологий представлены в сравнительной таблице 2.4.

Таблица 2.4 – Технологии создания компонентных систем

Технология программирования	**Поддержка языков программирования**	**Поддержка операционных систем**	**Особенности**
Microsoft DCOM	Множество	Windows	Устарела
Microsoft Remoting	Платформа *.NET*	Windows	Устарела
Microsoft WCF	Платформа *.NET*	Windows	Только для *Windows* и платформы *.NET*
CORBA	*Java, C++*	Различные	Слишком сложный стандарт, из которого многое не реализовано; сложная привязка к *C++*
Java EE	*Java*	Различные	Только *Java*
Web-сервисы	Множество	Различные	Использование стандартных технологий интернета: *HTTP, XML*
ZeroC ICE	Множество	Различные	Лаконичный и богатый язык описания интерфейсов; быстрый бинарный протокол; набор дополнительных инструментов

В большинстве перечисленных аспектов лидирует компонентное программное обеспечение промежуточного уровня *(Middleware) ZeroC ICE*. Его целесообразно использовать в качестве базового механизма построения компонентной распределённой системы.

Для тех случаев, когда требуется более универсальный протокол, нежели протокол *ICE*, целесообразно применение web-сервисов.

Выбранные технологии для разработки вычислительной среды

На основе приведённого анализа осуществлён выбор технологий, представленный в таблице 2.5.

Выбранные технологии в большей степени, нежели их конкуренты, отвечают следующим требованиям: кросс-платформенность; высокоуровневость модели программирования; производительность.

Таблица 2.5 – Технологии реализации вычислительной среды

Категория	**Технология**	**Назначение**
Программирование систем с общей памятью	*OpenMP*, *Intel TBB*, библиотеки численных методов	Программирование сложных, разветвлённых алгоритмов
Программирование высоко-параллельных вычислений	*CUDA*, библиотеки численных методов	Программирование простых алгоритмов с высокой степенью параллелизма
Программирование вычислительных кластеров	*MPI*	Программирование наиболее вычислительно сложных задач
Компонентное *middleware*	*ZeroC ICE*	Использование в качестве базы построения распределённой системы

Перечисленные технологии предпочтительны при разработке вычислительной среды и вычислительных сервисов на её основе.

2.2. **Архитектура компонентной среды организации параллельного и распределённого вычислительного процесса решения расчётных задач СППДР**

На основании проведённого в первой главе анализа предметной области сформулированы основные требования к разрабатываемой компонентной, мультизадачной вычислительной среде:

- модульность, гибкость, расширяемость и развиваемость;
- организация в распределённой среде вычислительных процессов произвольной конфигурации, в том числе соответствующей распределённой иерархии АСДУ ЕСГ;
- минимизация накладных расходов удалённого взаимодействия в распределённой среде;
- высокая производительность решения вычислительно-сложных задач моделирования СГ за счёт полной поддержки технологий параллельного и распределённого программирования, в том числе вычислений общего назначения на графических картах;
- унифицированное вычислительное обеспечение СППДР АСДУ за счёт предоставления вычислительных сервисов на основе стандартных протоколов;
- наличие развитых механизмов информационного взаимодействия для интеграции с ЕИП АСДУ.

Данные требования явились основой для последующей разработки проектных, алгоритмических и программных решений реализации вычислительной среды.

Архитектура построения вычислительной среды

Для реализации перечисленных выше требований при разработке архитектуры вычислительной среды были приняты три ключевых принципа.

1. Сервис-ориентированный подход на основе использования компонентного программного обеспечения промежуточного уровня (*middleware*).
2. Обеспечение явного управления вычислительными ресурсами.

3. Разработка вычислительных сервисов для среды по унифицированному алгоритму, позволяющему максимально использовать её преимущества.

Сервис-ориентированный подход средствами компонентного *middleware*

Сервис-ориентированный подход заключается в разработке программного обеспечения на основе использования слабо связанных компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам в распределённой среде – сервисов.

Для реализации сервис-ориентированного подхода применяются различные технологии, наиболее распространённая из них – использование *SOAP (Simple Object Access Protocol) Web*-сервисов. Однако, на основе проведённого анализа технологий распределённого программирования, принято решение использовать в диссертации компонентное обеспечение промежуточного уровня (*middleware*), характеризующееся более высокой эффективностью.

Сервис-ориентированный подход, реализованный на базе *middleware*, обеспечивает:

- высокую степень модульности системы – она строится из набора слабо-связанных компонентов с чётко определёнными интерфейсами, что создаёт предпосылки для гибкости системы;
- независимость от физического расположения компонентов – обеспечивается единообразное взаимодействие компонентов как в рамках одного процесса, так и в распределённой среде;
- инвариантность к платформе – поддерживается взаимодействие компонентов, работающих под управлением различных операционных систем и аппаратных платформ;
- поддержку различных языков программирования – обеспечивается взаимодействие компонентов, созданных с использованием различных языков и технологий программирования;

– простоту программирования – *middleware* предоставляет поддержку исполнения синхронных и асинхронных удалённых вызовов, развитую объектную модель, дополнительные инструментальные средства построения распределённых компонентных систем;
– высокую производительность – *middleware*, в отличие от *Web*-сервисов использует эффективные бинарные протоколы, многократно снижающие накладные расходы взаимодействия в распределённой среде.

Сервисы в распределённой системе выполняются компьютерами, связанными сетями передачи данных, – узлами, каждый из которых:

– содержит набор вычислительных ресурсов: центральных процессоров (ЦП); *GPGPU*-видеокарт (*General Purpose Graphics Processing Unit*); других устройств, способных осуществлять вычисления;
– работает под управлением той или иной операционной системы;
– выполняет ряд приложений, написанных с использованием тех или иных языков программирования и технологий параллельного программирования;
– содержит *middleware runtime* – среду времени выполнения компонентного программного обеспечения промежуточного уровня, выполняющую функции адаптера между прикладными сервисами и низкоуровневыми механизмами удалённого взаимодействия.

Схема взаимодействия сервисов в распределённой среде представлена на рисунке 2.2.

Каждый сервис в системе строится из двух частей: интерфейса и исполнителя. Интерфейс сервиса описывает множество предоставляемых сервисом методов на специализированном языке описания интерфейсов. Исполнитель сервиса реализует методы, объявленные в интерфейсе, функционирует в рамках вычислительных ресурсов узла распределённой системы, принимая и обрабатывая удалённые вызовы методов от клиентов.

Сервисы способны как предоставлять некоторую функциональность для клиентов, так и использовать другие сервисы, то есть одновременно находиться в роли и клиента, и сервера.

Рисунок 2.2 – Схема взаимодействия сервисов в распределённой среде

Для использования методов, предоставляемых сервисом, клиент генерирует объект-посредник – локальный для клиента объект, реализующий интерфейс сервиса – и использует его. Посредник устанавливает связь с исполнителем сервиса и переадресует локальные вызовы удалённому серверу прозрачно для клиента.

Традиционные подходы к управлению вычислительными ресурсами

Наиболее распространены следующие подходы к управлению вычислительными ресурсами в распределённых системах.

Прозрачное управление вычислительными ресурсами балансировщиком нагрузки. Балансировщик принимает удалённые вызовы методов сервисов от клиентов и переадресует их узлам распределённой системы, обеспечивая равномерное распределение нагрузки. Выделяется два типа сервисов.

1. Без сохранения состояния – вся необходимая информация для исполнения сервиса передаётся при каждом вызове. В этом случае распределение нагрузки организуется наиболее просто – сервер приложений переадресует вызов клиента наименее загруженному узлу.
2. С сохранением состояния – клиент выполняет последовательность операций, меняющих состояние удалённого объекта и зависящих друг от друга. В этом случае распределение нагрузки осуществляется более сложно. Применяются различные стратегии хранения состояний объектов в оперативной памяти, на жёстком диске, в базе данных.

Управление ресурсами вычислительного кластера штатной системой очередей. Для совместной работы множества пользователей с вычислительными кластерами используется следующая система: для запуска расчёта необходимо оформить вычислительное задание, состоящее из запускаемой программы и спецификации необходимых для её выполнения вычислительных ресурсов, и поставить его в очередь, используя штатную систему управления заданиями [**133**]. Такая система планирует очерёдность выполнения поставленных в очередь заданий, чтобы максимизировать общую загрузку кластера.

Явное управление вычислительными ресурсами

Для решения вычислительно-сложных многоуровневых итерационных задач моделирования СГ характерны требования, выходящие за рамки традиционных подходов.

Монопольный доступ клиента к вычислительному ресурсу. В каждый конкретный момент времени с вычислительным ресурсом может работать только один клиент, что целесообразно по ряду причин:

- большинство расчётных задач моделирования СГ является вычислительно сложными и могут полностью использовать потенциал ресурса – в такой ситуации разделение вычислительного ресурса между несколькими расчётными процессами на основе многозадачности приводит к снижению производительности;
- для некоторых вычислительных устройств, в частности CUDA-видеоадаптеров, использование в режиме многозадачности невозможно из-за недостаточно развитых механизмов прерываний, защиты памяти – в таком случае организация монопольного доступа необходима.

Выделение вычислительным сервисам, использующим технологии параллельных и распределённых вычислений, соответствующих вычислительных ресурсов. Для функционирования высокопроизводительных вычислительных сервисов требуются специализированные вычислительные ресурсы. Например, для сервиса, использующего вычисления общего назначения на графических картах, требуется соответствующий видеоадаптер; для сервиса, использующего технологию программирования *MPI (Message Passing Interface)*, требуется несколько узлов с настроенной средой выполнения *MPI.*

Прямой доступ клиента к серверу. В течение монопольной работы клиента с вычислительным ресурсом целесообразно организовать между ними прямой, без посредников, доступ.

Поддержка операций с сохранением состояния удалённого вычислительного ресурса без накладных расходов организуется автоматически за счёт монопольной работы клиента с вычислительным ресурсом. В том числе реализуется сохранение состояния памяти сопроцессоров удалённого вычислительного ресурса.

Управление конфигурацией вычислительного процесса в распределённой среде с использованием наиболее подходящих типов вычислительных ресурсов: от персональных компьютеров до ЦОД.

Перечисленные требования реализованы следующим образом.

Все сервисы в системе подразделены на две категории: служебные и вычислительные.

- Вычислительные сервисы решают расчётные задачи.
- Служебные сервисы организуют взаимодействие клиентов, сервисов и вычислительных ресурсов в распределённой среде.

Введено понятие *комплекта вычислительных ресурсов* как некоторой совокупности вычислительных ресурсов: от подмножества вычислительных ресурсов одного узла до объединения нескольких узлов.

Каждому комплекту вычислительных ресурсов присваивается *тип комплекта вычислительных ресурсов*. На основе типа сопоставляются вычислительные сервисы и подходящие для их выполнения комплекты вычислительных ресурсов.

В рамках каждого комплекта вычислительных ресурсов запускается *служебный сервис управления типизированным комплектом вычислительных ресурсов*, предоставляющий методы для запуска вычислительных сервисов соответствующего типа в рамках управляемого комплекта.

Для явного управления вычислительными ресурсами на одном из узлов вычислительной среды запускается *служебный сервис диспетчеризации ресурсов*. Его интерфейс предоставляет клиенту возможность установления прямого монопольного соединения с сервисом управления комплектом вычислительных ресурсов заданного типа – для этой операции введён термин «захват вычислительного ресурса».

Схема функционирования явного управления типизированными комплектами вычислительных ресурсов приведена на рисунке 2.3.

Рисунок 2.3 – Схема явного управления комплектами вычислительных ресурсов

Первоначально выполняется этап формирования вычислительной среды.

На узлах распределённой системы формируются типизированные комплекты вычислительных ресурсов и запускаются соответствующие служебные сервисы управления комплектами ресурсов. На одном из узлов запускается сервис диспетчеризации комплектов ресурсов.

Сервисы управления комплектами вычислительных ресурсов регистрируются в сервисе диспетчеризации, передавая ему свои объекты-посредники. Полученные посредники собираются сервисом диспетчеризации в типизированные пулы (для каждого типа объектов создаётся свой пул). Тем самым формируется множество комплектов вычислительных ресурсов, управляемых сервисом диспетчеризации.

Для осуществления своих вычислительных потребностей клиент инициирует следующую последовательность действий.

1. Клиент определяет, какой тип комплекта вычислительных ресурсов поддерживает выполнение необходимой ему операции и запрашивает у сервиса диспетчеризации объект-посредник сервиса управления комплектом вычислительных ресурсов необходимого типа.
2. Сервис диспетчеризации обеспечивает установление прямого монопольного соединения между клиентом и сервисом управления комплектом ресурсов запрошенного типа, передавая клиенту объект-посредник, изъятый из соответствующего пула.
3. Клиент использует методы сервиса управления комплектом вычислительных ресурсов для запуска необходимых вычислительных сервисов на захваченном комплекте вычислительных ресурсов.
4. Клиент выполняет удалённые вызовы методов запущенных вычислительных сервисов для выполнения необходимой работы.
5. Клиент освобождает захваченный вычислительный ресурс, и сервис диспетчеризации возвращает посредник его сервиса управления в соответствующий пул.

Контроль над организацией вычислительного процесса в распределённой среде достигается за счёт выбора клиентом сервиса диспетчеризации, у которого он запрашивает необходимый комплект вычислительных ресурсов – тем самым клиент определяет пул, из которого ему будет выделен комплект вычислительных ресурсов.

Различные комбинации расположения клиента, сервиса диспетчеризации, множества управляемых им вычислительных ресурсов формируют следующие конфигурации и режимы работы вычислительных процессов: однопользовательский режим; клиент-серверный режим; режим с использованием вычислительных ресурсов клиентов в локальной сети; распределённый режим, в частности, соответствующий распределённой иерархии АСДУ; комбинированный режим.

Локальный режим

Диспетчер и управляемые им комплекты вычислительных ресурсов находятся на компьютере клиента; набор доступных вычислительных сервисов ограничивается типом комплекта вычислительных ресурсов клиента. Схема локального режима работы представлена на рисунке 2.4.

Рисунок 2.4 – Локальный режим работы

Клиент-серверный режим

Комплекты вычислительных ресурсов сосредоточены в рамках сервера моделирования – множества узлов, обладающих вычислительными ресурсами, достаточными для одновременного обслуживания множества клиентов – в том числе в работе рассмотрено использование эластичных облачных вычислительных ресурсов. Вычислительные ресурсы сервера управляются одним сервисом диспетчеризации, удовлетворяющим запросы вычислительных ресурсов от клиентов. Клиенты освобождены от вычислительной нагрузки.

Схема клиент-серверного режима работы изображена на рисунке 2.5.

В качестве вычислительного сервера также могут быть использованы «эластичные» «облачные» ресурсы. Эластичность означает, что узлы облачного кластера запускаются и останавливаются при необходимости – например, дополнительные вычислительные узлы могут быть запущены для единовременного решения вычислительно сложной задачи или в периоды наиболее активной работы пользователей с системой.

Рисунок 2.5 – Клиент-серверный режим

Режим использования вычислительных ресурсов локальной сети клиента

При работе системы в этом режиме на компьютерах клиентов формируются комплекты вычислительных ресурсов; на одном из компьютеров запускается сервис диспетчеризации, управляющий всеми ресурсами локальной сети. Каждый клиент получает возможность расширить набор доступных ему вычислительных ресурсов за счёт свободных мощностей соседних компьютеров. Схема изображена на рисунке 2.6.

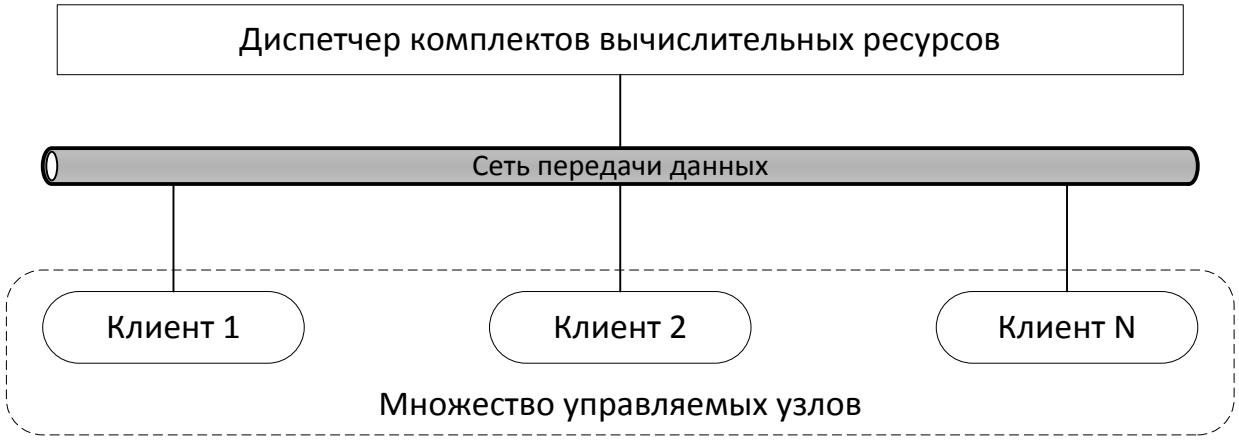

Рисунок 2.6 – Режим работы с использованием вычислительных ресурсов клиентов

Распределённый режим

Организация вычислительного процесса в распределённой системе, соответствующей иерархии диспетчерского управления, осуществляется следующим образом. Комплекты вычислительных ресурсов, расположенные на серверах диспетчерских служб различных уровней – ЦПДД, ПДС ЭО, ДП ЛПУ – управляются соответствующими сервисами диспетчеризации (рисунок 2.7).

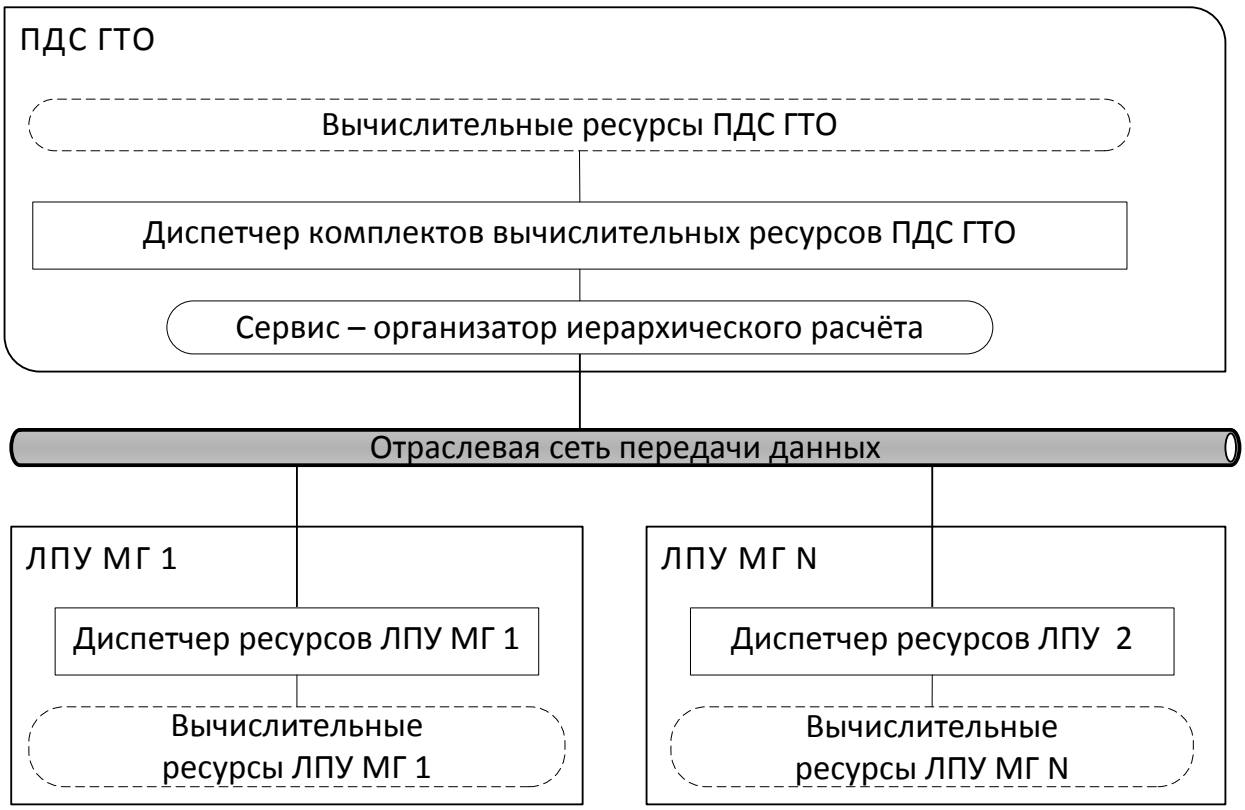

Рисунок 2.7 – Распределённый режим работы

Варианты организации вычислительных процессов:

– ЦПДД – ПДС ЭО: ЦПДД требуется организовать итерационный многоуровневый вычислительный процесс, отдельные части которого выполняются на уровне ПДС ЭО, а общее управление ведётся на уровне ЦПДД. Для этого запускается вычислительный сервис, работающий на вычислительных ресурсах ЦПДД, который, в свою очередь, запрашивает дополнительные комплекты вычислительных ресурсов у сервисов диспетчеризации уровня ПДС ЭО.

– ПДС ЭО – ЦПДД: ПДС ЭО требуется быстрое решение вычислительно-сложной задачи. ПДС ЭО запрашивает мощный вычислительный ресурс у сервиса диспетчеризации уровня ЦПДД (или одного из ЦОД АСДУ) и выполняет с его использованием решение необходимых задач.

Комбинированный режим

Перечисленные выше режимы комбинируются, например, при организации иерархического процесса взаимодействия ЦПДД – ПДС ЭО на уровне ПДС ЭО для решения расчётных задач целесообразно использовать вычислительные ресурсы локальной сети.

Алгоритм разработки вычислительных сервисов в предложенной вычислительной среде

Построение вычислительной среды на основе сервис-ориентированного подхода, наличие развитых средств для создания высокопроизводительных сервисов создают предпосылки для высокой степени модульности, гибкости, производительности системы, однако, не могут их гарантировать.

Унифицированный алгоритм разработки вычислительных сервисов в предложенной вычислительной среде введён для достижения следующих целей:

– обеспечение соответствия вновь разрабатываемых сервисов архитектуре вычислительной среды и максимальное использование её преимуществ;
– сохранение лёгкости и простоты внесения изменений в систему по мере разработки новых вычислительных сервисов;
– обеспечение высокой производительности разрабатываемых сервисов.

Разработанный алгоритм использует достижения в областях объектно-ориентированного анализа, проектирования и программирования, разработки программного обеспечения таких учёных как Г. Буч, Э. Гамма, Б. Лисков, Р. Мартин, К. Бек, Б. Страуструп, М. Фаулер и др. [**13**], [**73**], [**100**], [**115**], [**117**], [**138**], [**139**]. Схема алгоритма представлена на рисунке 2.8.

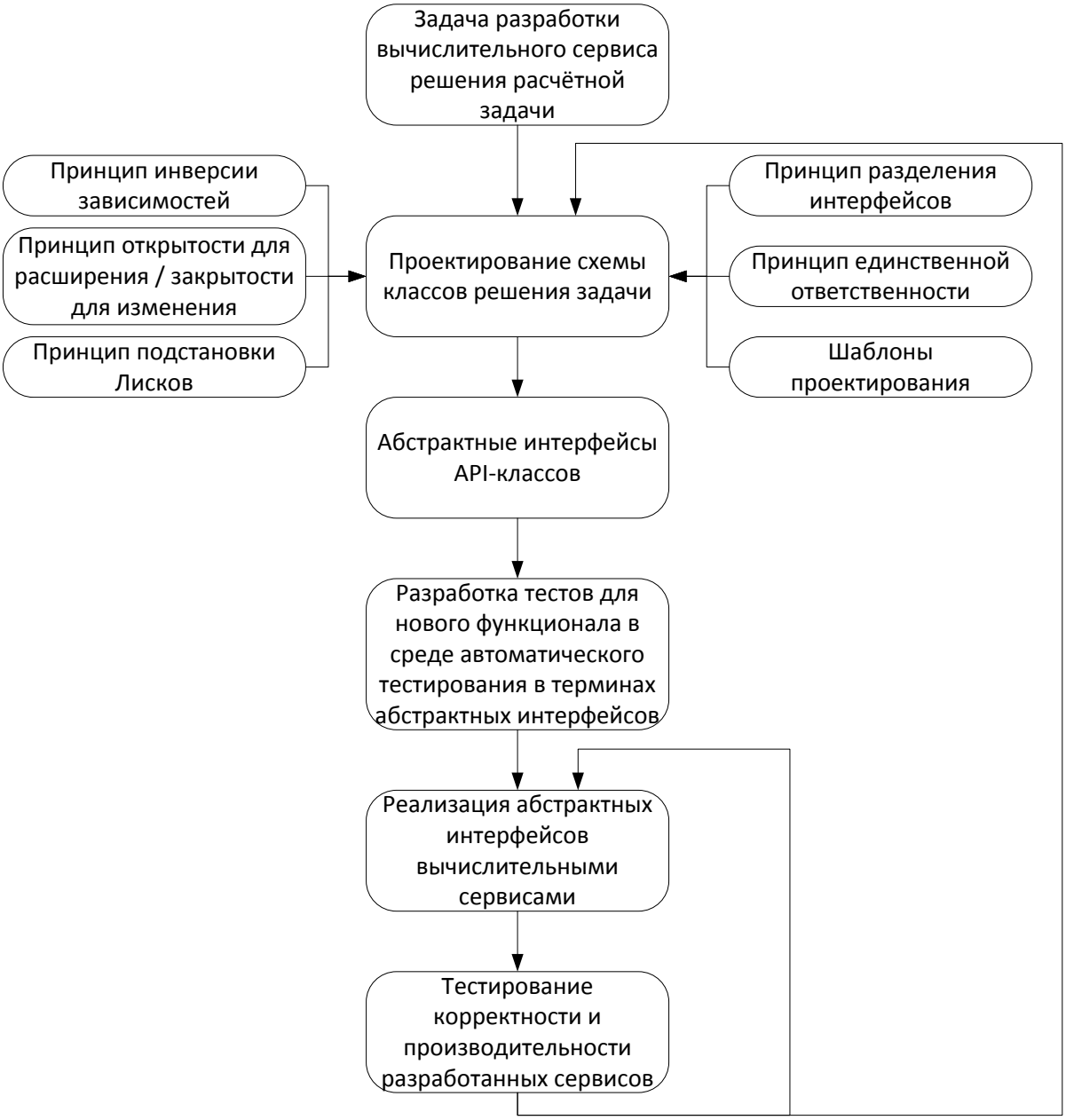

Рисунок 2.8 – Унифицированный алгоритм разработки вычислительных сервисов

Алгоритм состоит из следующих этапов.

1. Проектирование схемы классов вычислительного сервиса.
2. Разработка тестов для автоматизированной системы тестирования в терминах абстрактных интерфейсов.
3. Реализация абстрактных интерфейсов вычислительным сервисом.
4. Автоматизированное тестирование корректности и производительности разработанного сервиса.

5. Внесение изменений в систему в процессе эксплуатации.

Проектирование схемы классов решения задачи

Проектирование схемы классов сервиса выполняется с использованием языка моделирования *UML (Unified Modeling Language)* [**13**], [**95**], [**153**], принципов и шаблонов объектно-ориентированного программирования на трёх взаимосвязанных уровнях: концепций, интерфейсов и реализаций [**151**].

– На уровне концепций выявляются основные взаимодействующие сущности, их ответственности и взаимодействие для решения поставленной задачи. На основе анализа вариаций основных концепций выделяются части программы, подверженные изменениям – они инкапсулируются в виде семейств взаимозаменяемых классов.
– На уровне интерфейсов разрабатываются абстрактные интерфейсы, описывающие взаимодействие выявленных концепций для решения задачи. Результатом проектирования на этом уровне становится набор абстрактных интерфейсных классов.
– На уровне реализаций уточняется адекватность сформулированных интерфейсов, при необходимости выполняется их коррекция.

Схема этапа проектирования представлена на рисунке 2.9.

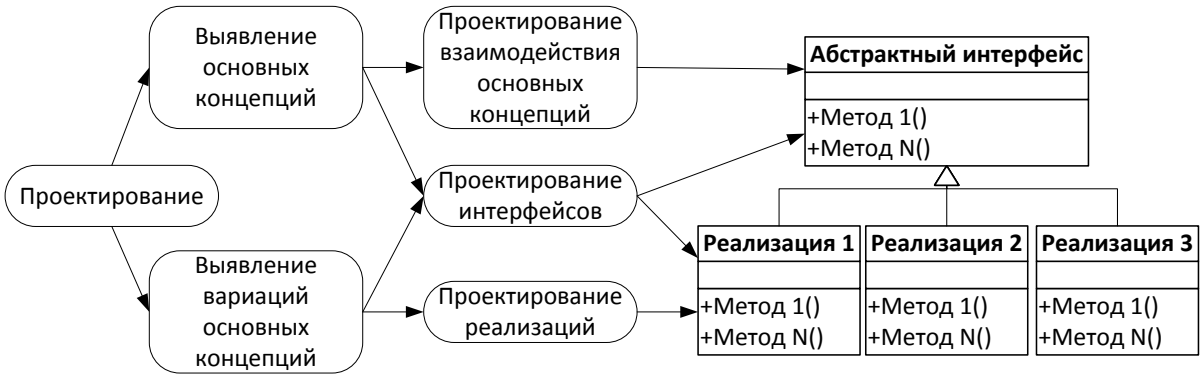

Рисунок 2.9 – Схема этапа проектирования

Выявление основных концепций позволяет определить основные сущности, их концептуальные ответственности и взаимодействия для решения задачи.

Анализ возможных вариаций основных концепций позволяет определить, какие части программы наиболее вероятно будут подвержены изменениям, инкапсулировать их и разработать все необходимые реализации в виде взаимозаменяемого семейства классов-реализаций.

Анализ вариаций способствует также уточнению основных концепций. Например, схема системы газоснабжения может быть загружена из источников различных типов: из файлов, баз данных различных форматов. В таком случае концепция – класс-загрузчик схем, ответственностью которого является осуществление загрузки схемы из внешнего источника; его вариации – различные реализации классов-загрузчиков для работы со схемами разных конкретных форматов.

После выявления основных концепций, представления решения задачи за счёт их взаимодействия, производится проектирование на уровне интерфейсов. Результатом этого этапа становится схема абстрактных интерфейсов основных концепций, являющаяся каркасом, смысловым ядром программы.

Абстрактные интерфейсы не содержат в себе никаких подробностей реализации, благодаря этому могут быть разработаны различные взаимозаменяемые вариации абстрактного интерфейса.

Использование основных принципов и шаблонов объектно-ориентированного программирования позволяет достичь высокой степени модульности системы, низкой степени взаимопроникновения отдельных классов друг в друга, сохранять гибкость системы по мере её развития.

При проектировании классов системы применяются принципы инверсии зависимостей; открытости для расширения при закрытости для изменения; принцип подстановки Лисков; принцип разделения интерфейсов; принцип единой ответственности [**138**], [**139**], [**140**]. При возможности используются шаблоны объектно-ориентированного программирования [**117**], [**135**], практики эффективного программирования [**142**].

Разработка тестов в терминах абстрактных интерфейсов

Разработка тестов для выполнения в системе автоматического тестирования осуществляется на основании разработанных на первом этапе интерфейсов, но до реализации, что позволяет тестировать различные реализации интерфейсов единым образом [**100**], [**112**]. При этом тесты формулируются в терминах методов абстрактных интерфейсов.

Реализация абстрактных интерфейсов

После разработки наборов тестов в терминах абстрактных интерфейсов создаются реализации этих интерфейсов, способные их пройти.

Реализации абстрактных интерфейсов осуществляются следующим образом.

1. Разрабатывается интерфейс вычислительного сервиса, совместимый с разработанным абстрактным интерфейсом.
2. Разрабатывается соответствующая реализация вычислительного сервиса.
3. Определяется тип комплекта вычислительного ресурса, необходимый для эффективного выполнения созданного сервиса; в интерфейс сервиса управления этим типом комплекта вычислительных ресурсов вносится метод для запуска и получения доступа к созданному вычислительному сервису.
4. Разрабатывается класс-посредник, реализующий абстрактный интерфейс, содержащий указатель на интерфейс вычислительного сервиса. Этот класс переадресует реализацию всех своих методов вычислительному сервису.
5. В абстрактный интерфейсный класс включается статический метод – «виртуальный конструктор» _ в зависимости от параметров запускающий необходимую реализацию сервиса и записывающий указатель на неё в класс-посредник.

В результате для получения доступа к разработанному сервису необходимо запросить у «виртуального конструктора» интерфейсного класса создание посредниеа, передав параметры, регулирующие необходимую

реализацию сервиса и диспетчер вычислительных ресурсов, у которого будет запрошен комплект вычислительных ресурсов для запуска сервиса.

Внесение изменений в систему

В процессе эксплуатации, как правило, требуется внесение изменений в систему. Такие изменения разделяются на две группы [**115**].

- Без изменения функциональности – улучшение архитектуры классов, скорости и эффективности выполнения задач. Целостность системы контролируется работой автоматических тестов: ни один тест не должен перестать показывать корректность работы.
- С изменением функциональности: новая функциональность, модификация существующей, устранение ошибок. В этом случае происходит возвращение на этап проектирования.

В результате применения предложенного алгоритма решение отдельной вычислительной задачи представляется в виде совокупности сервисов, которые динамически запускаются, останавливаются и взаимодействуют в распределённой среде.

2.3. **Проектирование, алгоритмическая и программная реализация вычислительных сервисов**

В структуре задач моделирования режимов СГ выделяется базовый комплекс расчётных задач, включающий в себя следующие основные процедуры:

- расчёт свойств газа и газового потока;
- процедуры моделирования режимов отдельных объектов СГ: трубопроводы, крановые системы, аппараты воздушного охлаждения, газоперекачивающие агрегаты (ГПА);
- процедуры моделирования подсистем СГ: трубопроводные системы, компрессорные цеха (КЦ), магистральные газопроводы, газотранспортные системы.

На основе процедур базового комплекса расчётных задач решаются задачи более высокого уровня – в частности, задачи оптимизации и прогнозирования режимов СГ, мониторинга фактического состояния СГ, ситуационного анализа, идентификации эмпирических параметров модели.

Многие высокоуровневые расчётные задачи, основанные на использовании базового комплекса, представляют собой многоуровневые иерархические вычислительные процессы, в которых моделирование режима отдельной СГ является основным вычислительным элементом.

Эффективное решение таких задач требует организации одновременного моделирования нескольких режимов СГ в распределённой среде с использованием параллельных вычислений для моделирования каждого из режимов.

В качестве апробации возможностей спроектированной вычислительной среды в данной работе реализовано два вычислительных сервиса:

- сервис моделирования режимов СГ;
- сервис идентификации эмпирических параметров модели режимов СГ.

Общая схема моделирования режимов систем газоснабжения

Для решения задачи моделирования режимов СГ топология системы задаётся условно-ориентированным «расчётным графом» (дуги имеют направление потоков газа, но оно может изменяться в процессе расчётов) [**5**], [**39**], [**53**], [**85**]

$$G = (V, E), |V| = m, |E| = n. \quad (1)$$

V – множество узлов графа, E – множество дуг.

Дугам графа соответствуют моделируемые технологические объекты, участвующие в перекачке газа (трубопроводы, крановые системы, ГПА, КЦ и так далее), – узлы графа соединяют входы/выходы дуг – соответствующих моделируемых объектов.

В краевых узлах графа задаются параметры газового потока, необходимые для расчёта режима.

Для каждого моделируемого объекта формируется расчётная модель, которая строится для каждой категории моделируемых объектов на основе математического представления физических законов моделируемого процесса и которую можно условно представить в виде функционала:

$$\{T_{\text{вых}}, q\} = f(P_{\text{вх}}, P_{\text{вых}}, T_{\text{вх}}, Gas, Environment, Object), \quad (2)$$

где q – расход газа по дуге, $P_{\text{вх}}, P_{\text{вых}}$ – давления на входе и выходе дуги, $T_{\text{вх}}, T_{\text{вых}}$ – температуры газа на входе и выходе дуги, Gas – компонентный состав газового потока, $Environment$ – параметры внешней среды, $Object$ – параметры объекта: нормативно-справочные (паспортные) данные; параметры фактического состояния; режимно-технологические ограничения.

Для сокращения выкладок используется упрощённый вид функционала (2):

$$q(P_{\text{вх}}, P_{\text{вых}}, \overline{A}) \quad (3)$$

где $\overline{A}$ – множество других заданных параметров объекта – дуги.

В общем случае функционал (3) представляет собой итерационную вычислительную процедуру.

Решение задачи состоит в определении поля давлений и температур во всех узлах графа, обеспечивающего баланс потоков по дугам во всех узлах графа, и соблюдение всех режимно-технологических ограничений расчётных объектов.

Эта задача решается итерационным (l – номер текущей итерации) *методом узловых давлений*, состоящим из последующих этапов [**67**].

1. Задание начального поля давлений и температур газа $\{P_k^0, T_k^0\}$ в узлах графа.
2. Расчёт расхода по каждой дуге на основе функционала (3).
3. Проверка наличия существенных дисбалансов в узлах графа.

$$|\delta q_k^{l+1}| < \varepsilon_q, \quad (4)$$

где k – номер узла, ε_q – точность балансирования потоков. Если дисбалансов нет – алгоритм завершён, иначе – переход к следующему шагу.

4. Формирование системы уравнений баланса потоков газа в каждом i-м узле графа в соответствии с 1-м законом Кирхгофа:

$$\delta q_k^{l+1} = \sum_{i=1}^{N_k} q_{i,k}^{l+1}\left(P_{i,\text{вх}}^{l+1},\, P_{i,\text{вых}}^{l+1}\right) - Q_k = 0, \quad (5)$$

где Q_k – заданные притоки или отборы газа в k-м узле, N_k – количество инцидентных k −му узлу дуг.

Функционал (3) на $l+1$ –й итерации представляется линейной составляющей разложения в ряд Тейлора:

$$q_i^{l+1}\left(P_{i,\text{вх}}^{l}, P_{i,\text{вых}}^{l}\right) \simeq q_i^l + \frac{\partial q_i^l}{\partial P_{i,\text{вх}}^{l+1}}\Delta P_{i,\text{вх}}^{l+1} + \frac{\partial q_i^l}{\partial P_{i,\text{вых}}^{l+1}}\Delta P_{i,\text{вых}}^{l+1} \quad (6)$$

и подставляется в соответствующие уравнения (5). В результате формируется система линейных алгебраических уравнений относительно расчетных компонент ΔP^{l+1}.

5. Решение системы линейных уравнений (5), расчёт нового поля давлений газа в узлах графа:

$$P_i^{l+1} = P_i^l + g \cdot \Delta P_i^{l+1}, \quad (7)$$

где g – регуляризационный параметр, одинаковый для всех выражений (7) узлов графа, начальное значение g равно единице.

6. Расчёт по каждой дуге расхода и температуры потока газа при новом поле давлений по функционалу (3).

   Если для каких-то объектов происходит нарушение режимно-технологических ограничений, осуществляется коррекция значения (7) посредством уменьшения параметра g.

7. Вычисление расчётного суммарного дисбаланса потоков газа в узлах графа

$$F^{l+1} = \sum_{k=1}^{m} \delta\, q_k^{k+1} \qquad (8)$$

Определение параметра g для выполнения условия $F^{l+1} < F^{l}$.

Переход к проверке условия завершения алгоритма (шаг 3).

Это общая алгоритмическая схема расчета режима газовых потоков в произвольной системе газоснабжения.

Проектирование архитектуры сервиса моделирования режимов систем газоснабжения

В соответствии с унифицированным алгоритмом разработки вычислительных сервисов в предложенной среде, на первом этапе осуществляется проектирование схемы классов задачи. Полученная диаграмма классов представлена на рисунке 2.10.

Архитектурная схема вычислительного сервиса состоит из следующих абстрактных классов.

Класс *Вычислитель режима СГ* – управляет всем процессом расчёта режима СГ. Он предоставляет единственный открытый метод *Рассчитать СГ*.

Класс *Загрузчик схемы* считывает схему из некоторого источника данных и создаёт объект *Расчётная схема.*

Класс *Расчётная схема* предоставляет методы заполнения расчётного графа дугами, обхода расчётного графа с помощью итераторов, расчёта интегрального дисбаланса в расчётном графе, смешения потоков в узлах схемы, а также моделирования всех дуг схемы. Дуги и узлы схемы представлены соответственно классами *Дуга схемы* и *Узел схемы.*

Класс *Дуга схемы* предоставляет методы задания рабочих параметров, получения расчётных параметров, доступа к начальному и конечному узлам дуги.

Класс *Узел схемы* предоставляет метод расчёта дисбаланса в узле и возможности доступа к входящим и исходящим дугам на основе итераторов.

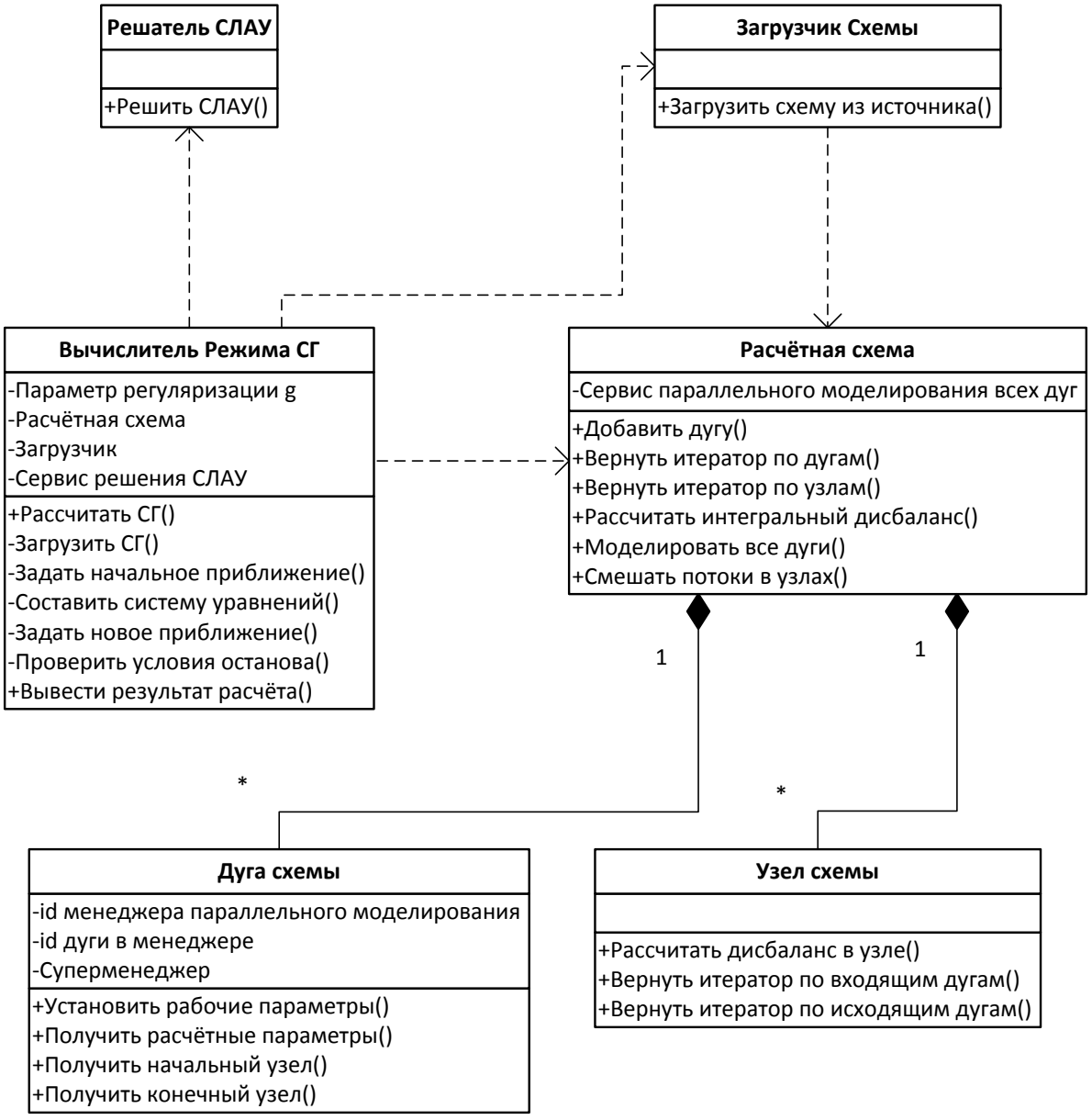

Рисунок 2.10 – Диаграмма классов сервиса моделирования СГ

Взаимодействие представленных классов для решения задачи расчёта режима СГ проиллюстрировано диаграммой взаимодействия на рисунке 2.11.

Выполняется следующая последовательность действий.

1. *Вычислитель режима СГ* создаёт *Загрузчик схемы*.
2. *Вычислитель режима СГ* вызывает метод «*Загрузить схему*» *Загрузчика схемы*.
3. *Загрузчик схемы* создаёт *расчётную схему*.
4. *Загрузчик схемы* считывает дуги схемы из БД и заполняет *Расчётную схему* соответствующими дугами.

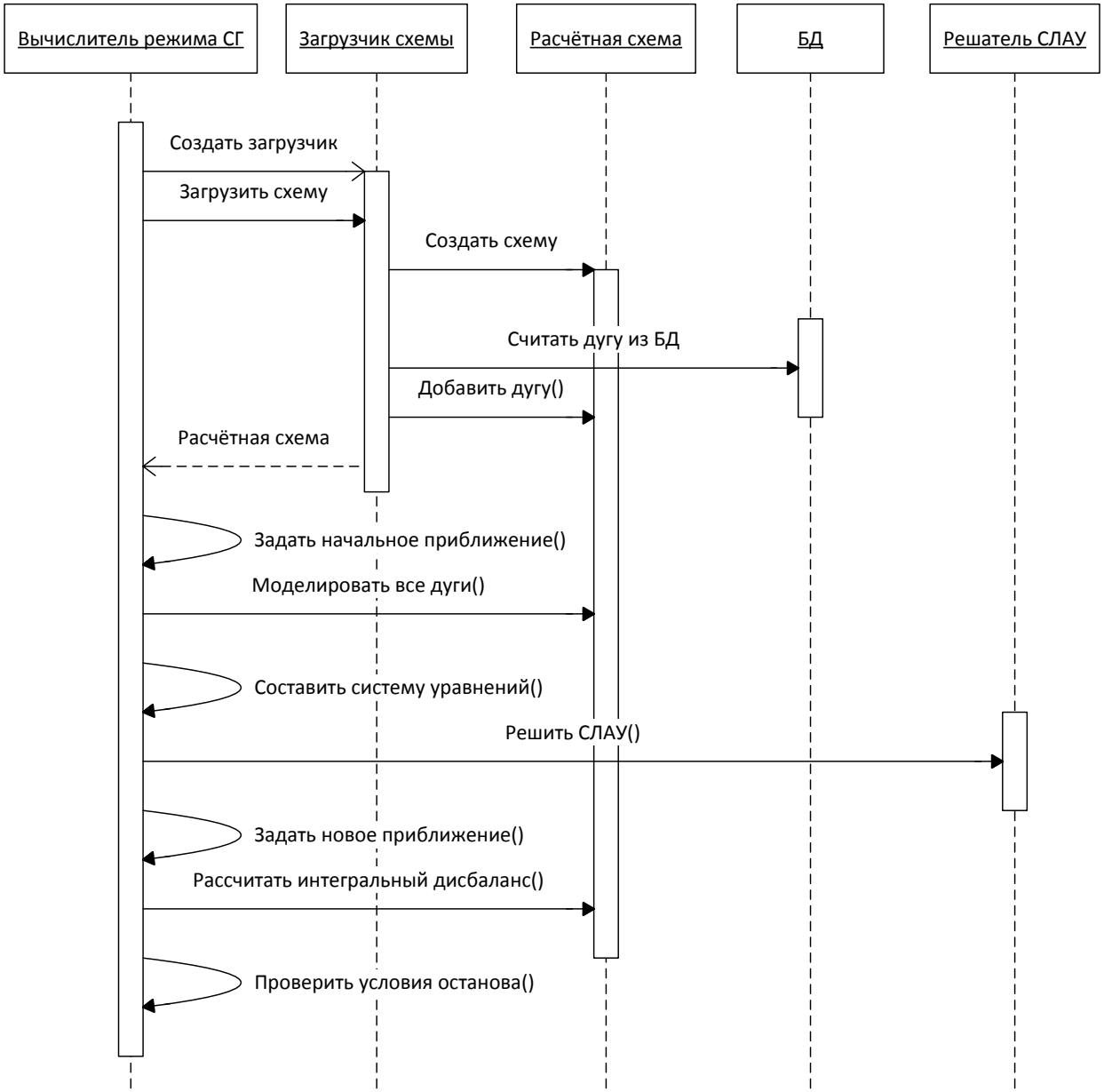

Рисунок 2.11 – Диаграмма последовательности действий моделирования СГ

5. *Загрузчик схемы* возвращает построенную *Расчётную схему Вычислителю режима СГ*.
6. *Вычислитель режима* задаёт начальное приближение в *Расчётной схеме*, выполняя обход дуг и узлов с использованием предоставленных итераторов.
7. *Вычислитель режима* вызывает метод *Расчётной схемы* «*Моделировать все дуги*» – осуществляется моделирование всех дуг расчётной схемы: расчёт расходов и производных.

8. *Вычислитель режима СГ* составляет систему уравнений баланса потоков газа в узлах схемы, выполняя обход *Расчётной схемы* и используя рассчитанные значения расходов и производных.
9. *Вычислитель режима СГ* вызывает метод решения системы линейных алгебраических уравнений (СЛАУ) *Решателя СЛАУ* и получает решение – вектор dP.
10. *Вычислитель режима СГ* задаёт новое приближение поля давлений в *Расчётной схеме*.
11. *Вычислитель режима СГ* вызывает метод расчёта интегрального дисбаланса в *Расчётной схеме*.
12. *Вычислитель режима СГ* проверяет выполнение условий останова: если условия выполнены, то алгоритм завершён; если не выполнены – происходит возврат на шаг 7 данного алгоритма.

Проектирование этапа параллельного моделирования дуг схемы

Моделирование дуг графа в методе узловых давлений на итерационных шагах (2) и (6) целесообразно производить параллельно, поскольку расчёт режима каждой дуги является независимой процедурой, связывающей изолированные для каждой дуги множества заданных и расчетных параметров.

Для эффективного решения этой задачи разработано специализированное решение.

Все дуги расчётного графа разбиваются на группы по типам объектов – формируются векторы однотипных расчётных объектов.

Для каждого типа расчётных объектов разработан свой менеджер параллельного моделирования, осуществляющий эффективное размещение в памяти и параллельное моделирование управляемых объектов. Для координации совместной работы всех менеджеров параллельного расчёта разработан сервис суперменеджер.

Схема классов разработанного решения представлена на рисунке 2.12.

Рисунок 2.12 – Диаграмма классов менеджеров параллельного моделирования дуг расчётной схемы

Менеджер параллельного расчёта для каждого типа моделируемых объектов – это класс, отвечающий за организацию хранения данных и параллельного моделирования всех моделируемых объектов данного типа.

Для моделирования всех дуг расчётной схемы необходимо организовать выполнение параллельного моделирования всех векторов объектов их менеджерами. Для этого тоже целесообразно ввести класс – суперменеджер.

Класс *Дуга схемы* имеет метод «*Установить рабочие параметры по id менеджера и id дуги*». В реализации этого метода выполняется изменение рабочих параметров, хранящихся в суперменеджере.

В реализации метода «*Получить рассчитанные параметры по id менеджера и id дуги*» значение получается из множества рассчитанных параметров, хранящихся в суперменеджере.

При вызове метода «*Моделировать все дуги всех менеджеров*» *Суперменеджер* отправляет изменившиеся множества рабочих параметров *Менеджерам параллельного моделирования:* они выполняют параллельное

моделирование и возвращают *Суперменеджеру* множество рассчитанных значений.

Таким образом, суперменеджер является для расчётной схемы единой точкой для работы с рабочими и расчётными параметрами дуг любых типов.

Процессы загрузки расчётной схемы с дальнейшим параллельным моделированием всех дуг расчётной схемы представлены на диаграмме взаимодействия на рисунке 2.13.

Выполняется следующая последовательность действий:

1. *Загрузчик* создаёт *Расчётную схему*.
2. *Загрузчик* создаёт *Суперменеджер*.
3. *Загрузчик* создаёт *Менеджеры параллельного моделирования* для всех типов дуг, входящих в считываемую схему.
4. *Загрузчик* передаёт созданные менеджеры под управление *Суперменеджеру*.
5. *Загрузчик* считывает из источника очередную дугу графа, определяет её тип и передаёт под управление *Менеджеру параллельного моделирования* соответствующего типа.
6. *Менеджер параллельного моделирования* возвращает свой id и id принятого под управление объекта *Загрузчику*.
7. *Загрузчик* добавляет в *Суперменеджер* дугу с полученными id.
8. *Загрузчик* добавляет в *Расчётную схему* дугу с полученными id.
9. После завершения загрузки *Расчётная схема* вызывает метод *Моделировать все дуги всех менеджеров Суперменеджера*.
10. *Суперменеджер* вызывает методы *Установить множество рабочих параметров* для всех *Менеджеров параллельного расчёта*.
11. *Суперменеджер* вызывает методы *Моделировать все управляемые объект*» всех *Менеджеров параллельного моделирования* с учётом вычислительных ресурсов, на которых они выполняются. Менеджеры, использующие различные вычислительные ресурсы, работают параллельно. Менеджеры, использующие один и тот же вычислительный ресурс, ставятся в очередь.
12. *Менеджеры параллельного моделирования* осуществляют параллельное моделирование управляемых объектов и возвращают *Суперменеджеру*

множества рассчитанных параметров. *Суперменеджер* актуализирует состояние своего множества расчётных параметров.

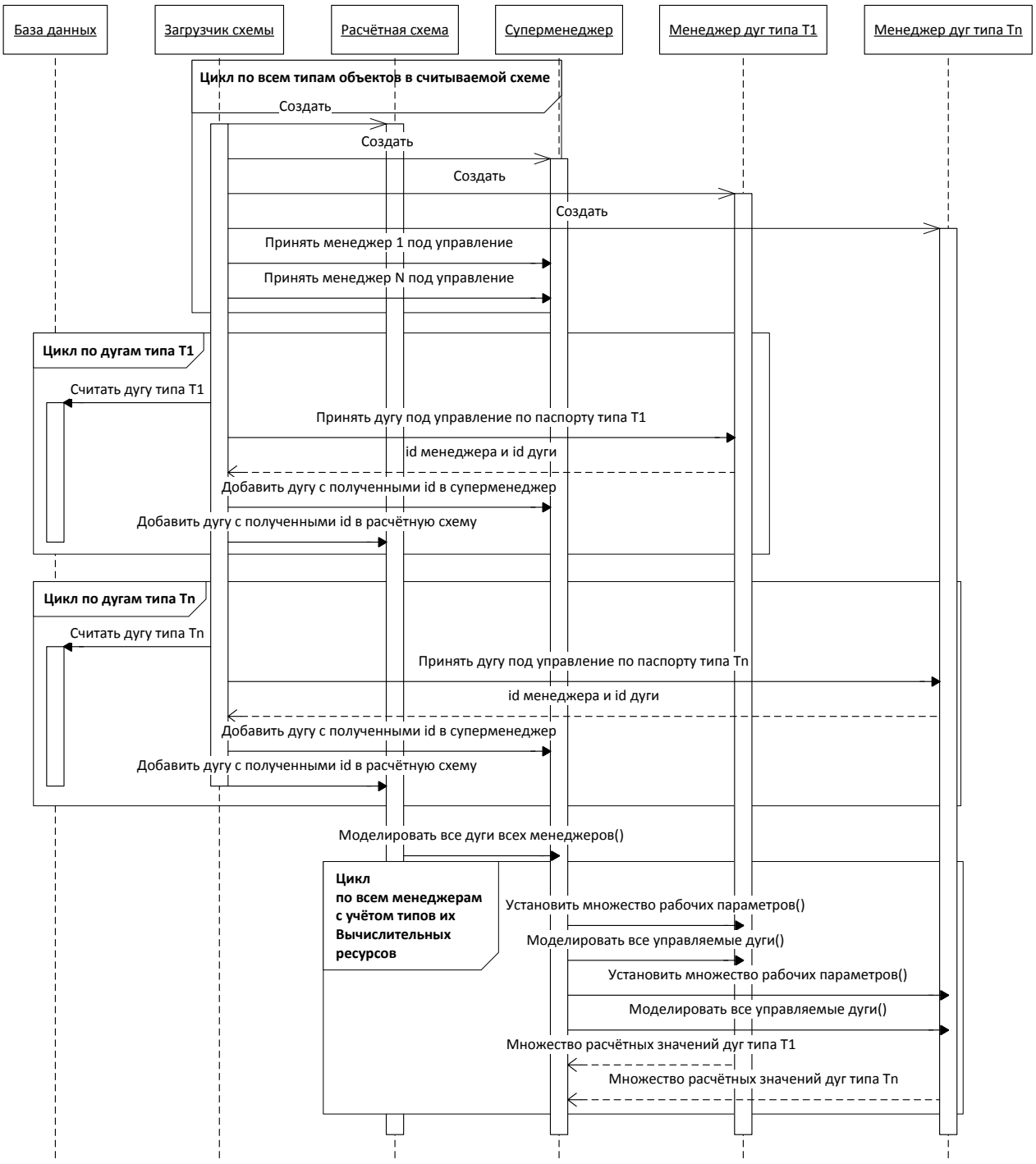

Рисунок 2.13 – Последовательность действий при параллельном моделировании объектов расчётной схемы

Проектирование этапа параллельного решения системы уравнений

Решение систем линейных уравнений целесообразно производить с использованием оптимизированных численных библиотек: такие библиотеки существуют как для многоядерных процессоров, так и для GPGPU-устройств, так и для вычислительных кластеров.

Различные вариации класса *Решатель СЛАУ* сделаны взаимозаменяемыми с использованием унифицированного алгоритма разработки вычислительных сервисов и шаблона проектирования *Strategy* [**54**], [**117**]. Создание конкретной реализации класса *Решатель СЛАУ* выполняется классом *Вычислитель режима СГ* в зависимости от наличия доступных вычислительных ресурсов и размерности задачи.

Системы уравнений балансов потоков в узлах расчётной схемы, формируемые в методе узловых давлений, как правило, являются разреженными. В связи с этим в абстрактном интерфейсе класса *Решатель СЛАУ* метод «Рассчитать СЛАУ» принимает матрицу системы уравнений в формате представления разреженных матриц.

Выбор комплекта вычислительных ресурсов для работы сервиса

Для работы вычислительных сервисов, используемых сервисом моделирования режимов СГ, требуются следующие вычислительные ресурсы:

- Многоядерный центральный процессор для организации вычислительного процесса моделирования; параллельного моделирования расчётных объектов, обладающих сложными вычислительными моделями, соответствующими менеджерами параллельного моделирования; работы сервиса суперменеджера;
- *GPGPU*-видеокарта *CUDA* для параллельного моделирования режимов многочисленных объектов, обладающих простыми вычислительными моделями, таких как трубопроводы; решения систем линейных алгебраических уравнений с использованием оптимизированной библиотеки численных методов.

Таким образом, для эффективного выполнения вычислительного сервиса моделирования режимов СГ необходим комплект вычислительных ресурсов, содержащий центральный процессор и *CUDA*-видеокарту. Тип такого комплекта вычислительных ресурсов назван «ЦП + *CUDA*».

Схема взаимодействия созданных сервисов в рамках комплекта вычислительных ресурсов «ЦП + *CUDA*» для решения задачи моделирования СГ представлена на рисунке 2.14.

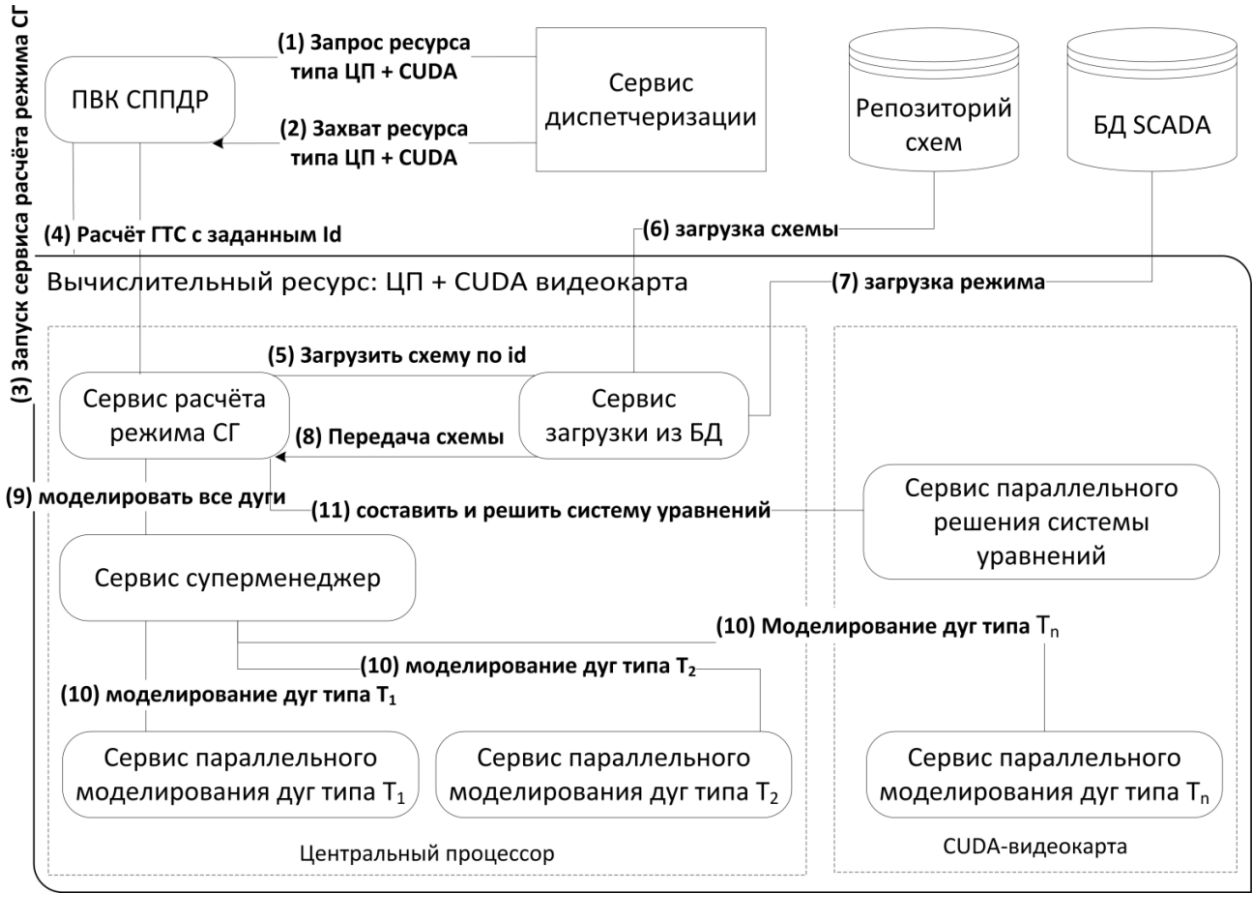

Рисунок 2.14 – Сервис расчёта режима системы газоснабжения

ПВК СППДР использует разработанный сервис моделирования по следующей схеме.

1. ПВК СППДР вызывает метод программного интерфейса, предоставляемого средой моделирования для Расчёта режима СГ.
2. Происходит захват вычислительного ресурса, состоящего из центрального процессора и *CUDA*-видеокарты.
3. На захваченном ресурсе запускается сервис расчёта режима СГ.

4. Осуществляется вызов метода расчёта СГ запущенного сервиса, в качестве параметра ему передаётся *id* схемы в БД.
5. Сервис расчёта режима СГ создаёт реализацию загрузчика из БД и запрашивает его о загрузке расчётной схемы.
6. Загрузчик осуществляет загрузку схемы из БД.
7. Загрузчик считывает фактические параметры режима из БД *SCADA*-системы (только для режима работы on-line).
8. Загрузчик формирует расчётную схему, создаёт Суперменеджер, необходимые менеджеры параллельного расчёта. Сервисы параллельного расчёта моделируемых сложных объектов запускаются на ЦП, а сервисы моделирования многочисленных и простых объектов на *CUDA GPU*.
9. Сервис расчёта режима СГ вызывает метод Суперменеджера «Моделировать все дуги всех менеджеров».
10. Менеджеры параллельного моделирования осуществляют параллельное моделирование управляемых ими объектов.
11. Сервис расчёта СГ запускает сервис решения СЛАУ на *CUDA GPU,* формирует и решает систему уравнений балансов потоков газа в узлах расчётной схемы.
12. Далее выполняются итеративные расчёты параметров режимов СГ, начиная с шага 7, за исключением того, что все необходимые объекты уже созданы, необходимые сервисы запущены.

Моделирование системы газоснабжения в распределённой среде

Разработанные вычислительные сервисы применимы для организации распределённого иерархического вычислительного процесса расчёта режима СГ на основании того, что в методе узловых давлений в качестве дуг могут быть использованы любые моделируемые объекты, в том числе – подсистемы.

Подсистема СГ представляет собой изолированный граф потоков газа, сопряженный с другими частями системы по краевым узлам. При этом для расчета технологического режима подсистемы достаточно задать параметры потока в этих краевых узлах.

Таким образом, если сложную распределенную систему газоснабжения можно декомпозировать на множество смежных подсистем меньшей размерности, то задача расчета технологического режима по всей системе может быть сведена к двум итерационным процессам:

- балансирование потоков между подсистемами посредством расчета давлений газа в краевых узлах смежных подсистем;
- балансирование потоков внутри подсистем, когда давления потока газа в краевых узлах подсистемы принимаются рассчитанными на верхнем итерационном уровне.

Таким образом, распараллеливание может быть выполнено как в процедуре балансирования потоков внутри каждой подсистемы, так и при балансировании потоков между подсистемами.

Такой подход позволяет решать задачи моделирования единых технологических режимов систем большой размерности (десятки и сотни тысяч расчетных элементов), примерами которых являются региональные газотранспортные системы или системы газоснабжения, объединяющие такие разнородные системы как системы добычи, транспорта, подземного хранения и распределения газа.

Задача параллельного моделирования множества «подсистем» выполняется соответствующим менеджером параллельного моделирования.

Схема использования предложенной архитектуры для организации распределённого иерархического вычислительного процесса представлена на рисунке 2.15.

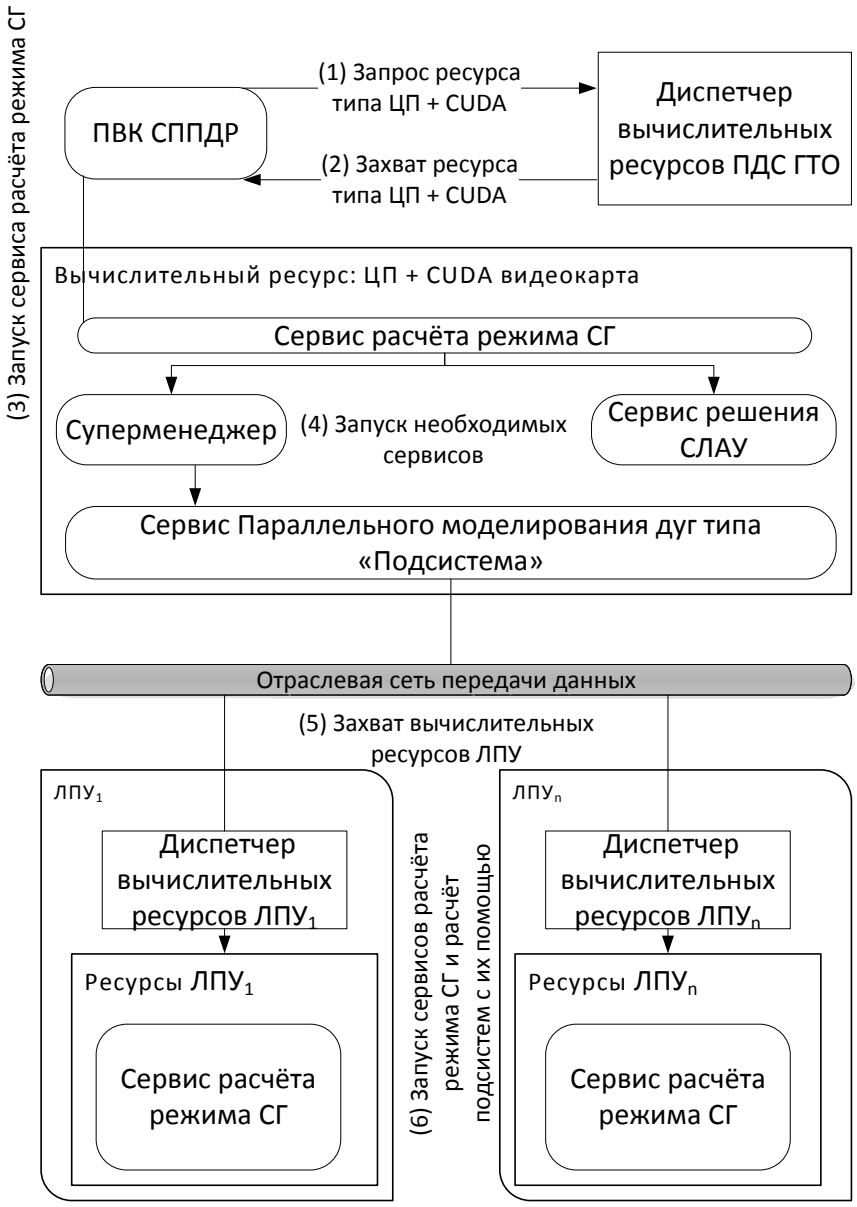

Рисунок 2.15 – Организация вычислительного процесса моделирования СГ в распределённой среде

Основной особенностью предложенного вычислительного процесса является то, что Менеджер параллельного моделирования «дуг» типа «Подсистема» захватывает дополнительные удалённые вычислительные ресурсы типа «ЦП + *CUDA*» и организует параллельное моделирование управляемых подсистем в распределённой среде.

Характеристика предложенной архитектуры сервиса моделирования систем газоснабжения

Эффективное использование параллельных вычислений. Для каждого типа моделируемых объектов разрабатывается собственный менеджер параллельного моделирования, обеспечивающий использование наиболее подходящей технологии параллельного программирования и наиболее подходящей стратегии размещения данных в памяти именно для этого типа объектов.

Расширяемость и гибкость. Для включения в систему нового типа моделируемого объекта достаточно разработать менеджер параллельного расчёта для объекта этого типа и переработать классы-загрузчики таким образом, чтобы они передавали считываемые объекты соответствующего типа под управление новым менеджерам.

Проектирование архитектуры сервиса идентификации эмпирических параметров модели режимов систем газоснабжения

Идентификация параметров модели режима СГ производится для повышения адекватности расчётных результатов модели, то есть их приближения к замеряемым значениям параметров газового потока в контрольных точках системы. Решение этой задачи требует большого объёма вычислений и имеет потенциал для многократного ускорения с использованием параллельных и распределённых вычислений в разработанной в данной диссертации вычислительной среде.

Исходные данные задачи те же, что и для расчёта режима СГ, а также дополнительная, избыточная информация – множество значений замеров параметров газового потока в некоторых контрольных узлах СГ. В качестве замеров часто используются:

– P_i^*, T_i^* – замеры давления и температуры газа на входах и выходах КЦ, ГПА;

– Q_i^* – расходы газового потока, замеряемые на газоизмерительных станциях (ГИС).

Состав эмпирических параметров модели СГ определяется типами технологических объектов системы.

Для трубопроводных систем такими параметрами являются: $K_{\text{то}}, K_{\text{эф}}$ – коэффициенты теплообмена и гидравлической эффективности трубопроводов, а для компрессорных цехов: K_η, K_N – интегральные по КЦ поправочные коэффициенты на газодинамические характеристики центробежных нагнетателей (ЦБН) однотипных ГПА, а также на коэффициенты технического состояния ЦБН ГПА.

Следует, однако, отметить, что для идентификации эмпирических параметров модели каждого объекта системы перекачки газа потребовались бы замеры давления и температуры на входах и выходах объекта, а также замер расхода газа через объект, что в реальных технологических системах невозможно.

Поэтому объекты одного типа, например трубы или ГПА объединяются в трубопроводные подсистемы или компрессорные цеха, а идентифицируемыми параметрами выбираются не сами эмпирические параметры модели, а поправки к ним в целом для всей соответствующей подсистемы однотипных объектов перекачки.

При этом в соответствии с условием корректности задачи количество идентифицируемых поправок к эмпирическим параметрам моделей подсистем не должно превышать количества избыточных контрольных параметров-замеров газового потока в рассматриваемой системе. В противном случае количество линейно-независимых уравнений по расчетным идентифицируемым параметрам может оказаться меньше числа самих параметров и решение задачи будет многовариантным. Более того, расчетные параметры газового потока в контрольных узлах должны быть чувствительны по идентифицируемым параметрам модели, то есть производные не должны быть равны нулю. Иначе множество

идентифицируемых параметров модели будет содержать неидентифицируемое подмножество и вычислительная работа, затрачиваемая на их расчет, будет бесполезной.

Для математического описания рассматриваемой задачи введем следующие обозначения: $\vec{a} \in \mathbb{R}^{n_a}$, вектор идентифицируемых параметров расчётной модели, $\vec{p}^{\,*} \in \mathbb{R}^{n_p}$ и $\vec{p} \in \mathbb{R}^{n_p}$ – векторы замеряемых и расчётных параметров режима соответственно.

Множество расчётных параметров $\vec{p}$ модели процесса является функцией идентифицируемых параметров $\vec{a}$:

$$\vec{p} = f(\vec{a}) \tag{9}$$

Задача идентификации эмпирических параметров модели в общем виде сводится к минимизации некоторого критерия рассогласования $\vec{p}^{\,*}$ и $\vec{p}$, в качестве которого можно использовать различные критерии математической статистики, в частности сумму нормированных квадратов разностей разноразмерных параметров

$$F = \min_{\vec{a}} \sum_{r=1,\dots n_p} \left(\frac{p_r^* - p_r(\vec{a})}{p_r^*} \right)^2 \tag{10}$$

Для решения задачи (10) применяются различные численные методы – в частности, *метод чувствительности*, основанный на итерационном поиске компонент искомого вектора эмпирических параметров:

$$\vec{a}^{\,l+1} = \vec{a}^{\,l} + g \cdot \Delta\vec{a}^{\,l+1}, \tag{11}$$

где g – регуляризационный параметр.

Условием минимума (10) является система уравнений

$$\frac{\partial F^{l+1}}{\partial \Delta a_k^{l+1}} = 0, \text{ для } k = 1, \dots n_a \tag{12}$$

Функционал (9) на *l+1* – ой итерации представляется в виде линейной составляющей разложения в ряд Тейлора:

$$p_r(\vec{a}^{l+1}) = p_r(\vec{a}^{l}) + \sum_{i=1}^{n_a}\left(\frac{\partial p_r(a^l)}{\partial a_i^l}\Delta a_i^{l+1}\right) \quad (13)$$

Подставляя (13) в (12) и выполняя ряд преобразований, получаем систему линейных уравнений относительно Δa_i^{l+1}:

$$\sum_{i=1}^{n_a}\sum_{r=1}^{n_p}\left(\frac{1}{(p_r^*)^2}\frac{\partial p_r^l}{\partial a_i^l}\frac{\partial p_r^l}{\partial a_k^l}\right)\Delta a_i^{l+1} = \sum_{r=1}^{n_p}\frac{1}{(p_r^*)^2}(p_r^* - p_r^l)\left(\frac{\partial p_r^l}{\partial a_k^l}\right), \quad (14)$$
$$k = 1 \ldots n_a$$

Процедура метода чувствительности строится на основе итерационного формирования и решения системы линейных уравнений (14), вычисления поправок Δa_i^{l+1} к идентифицируемым параметрам модели режима СГ.

Для численной аппроксимации производных в (14) требуется вычисление $n_a + 1$ режимов СГ при следующих значениях вектора эмпирических параметров модели:

$$\vec{a} = (a_1, a_2, \ldots, a_k+, \ldots, a_{n_a});\ \vec{a}_k^+ = (a_1, a_2, \ldots, a_k + \delta, \ldots, a_{n_a}), \quad (15)$$
$$k = 1..n_a$$

Размерность задачи многократно возрастает, когда имеется временной ряд замеров, и функция цели задачи (10) содержит сумму рассогласований контрольных параметров по всем временным слоям $t_j \in [t_0, t_m]$.

На каждой внешней итерации минимизации функции цели (10) при соответствующих состояниях вектора идентифицируемых параметров $\vec{a}$ расчеты режимов СГ могут выполняться независимо, поэтому имеется возможность их распараллеливания. Расчёт даже одного режима СГ с помощью разработанного в данной диссертации вычислительного сервиса полностью использует вычислительные ресурсы отдельного узла среды моделирования. Поэтому необходимо организовать одновременный расчёт множества режимов СГ на нескольких узлах распределённой системы – многоуровневый итерационный вычислительный процесс в распределённой среде.

Схема работы вычислительного сервиса, на примере задачи идентификации эмпирических параметров модели режима, в распределённой вычислительной среде представлена на рисунке 2.16.

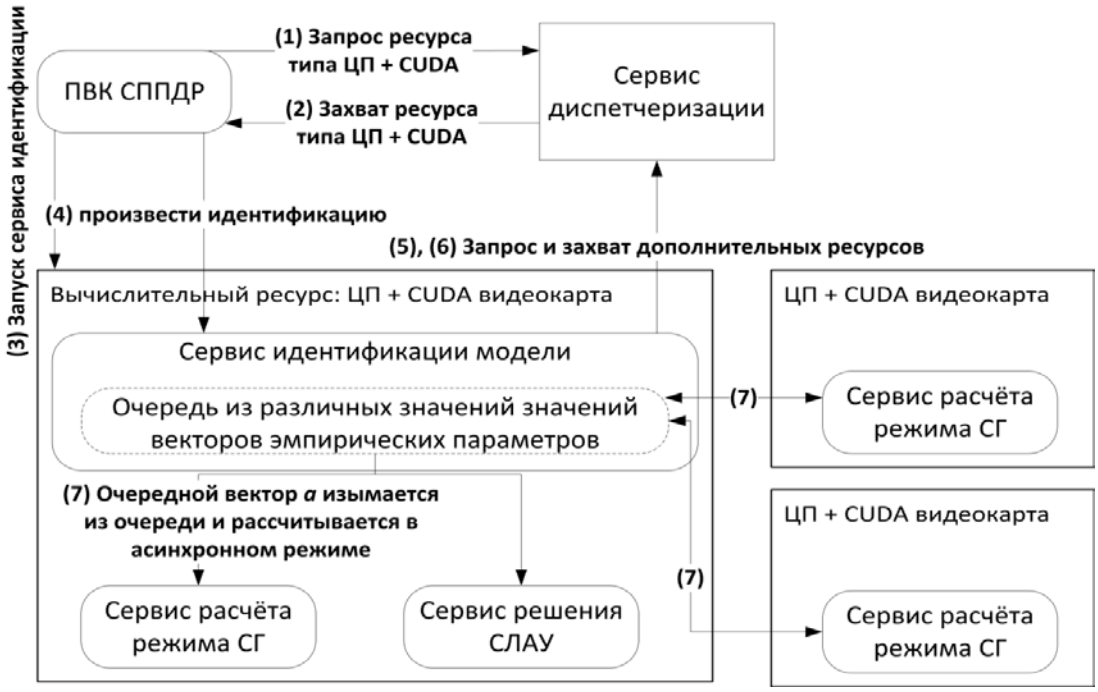

Рисунок 2.16 – Схема работы вычислительного сервиса идентификации эмпирических параметров модели СГ в распределённой среде

На этапах (1) – (6) осуществляется формирование вычислительной среды: запуск сервиса идентификации, захват им дополнительных вычислительных ресурсов типа «ЦП + CUDA» для моделирования режимов СГ при различных значениях вектора эмпирических параметров $\vec{a}$.

На этапе (7) реализуется итерационный многоуровневый процесс идентификации параметров модели СГ в распределённой среде.

Сервис идентификации запускает на каждом из захваченных ресурсов сервис расчёта режима СГ и вызывает метод загрузки адаптируемой СГ на каждом из них.

На каждой итерации метода чувствительности сервис идентификации формирует очередь из всех значений вектора идентифицируемых параметров

модели (15), которые необходимо рассчитать для вычисления частных производных $\frac{\partial p_r(a^l)}{\partial a_i^l}$ в (14).

Сервис идентификации выполняет асинхронный запуск расчёта режима СГ на каждом из доступных сервисов моделирования режима СГ. Как только один из вычислительных ресурсов заканчивает свой расчёт СГ, сервис идентификации извлекает из очереди очередную компоненту вектора $\vec{a}$ и снова загружает его работой.

После исчерпания очереди сервис идентификации формирует систему уравнений (12), решает её с помощью сервиса решения систем уравнений, формирует новое состояние вектора идентифицируемых параметров модели $\vec{a}$, проверяет условие завершения алгоритма и переходит к следующей итерации либо завершает работу.

Решающее значение в эффективности описанного алгоритма имеет использование операций с сохранением состояния. На каждом из комплектов вычислительных ресурсов моделируемая СГ загружается только один раз. Для каждого расчёта режима передаётся только новое состояние вектора эмпирических параметров $\vec{a}$. Без возможности сохранения состояния удалённого вычислительного ресурса было бы необходимо для каждого расчёта режима загружать СГ целиком, что привело бы к чрезмерным накладным расходам.

Выводы

1. Выполнен сравнительный анализ современных технологий параллельного, распределённого программирования, создания компонентных систем – на его основе выбраны технологии построения единой вычислительной среды.
2. Разработаны основные проектные решения вычислительной среды, позволяющие реализовать предъявляемые к ней требования. Архитектурные решения основаны на реализации сервис-ориентированного подхода средствами *middleware*, обеспечении явного управления вычислительными ресурсам и применении унифицированного алгоритма разработки вычислительных сервисов.
3. Для апробации предложенных архитектурных решений спроектированы вычислительные сервисы моделирования систем газоснабжения и идентификации эмпирических параметров модели системы газоснабжения на основе фактических режимов.
4. Для разработанных вычислительных сервисов предложены проектные решения, позволяющие организовывать вычислительные процессы решения многоуровневых итерационных задач, сочетающие гетерогенные параллельные вычисления, в том числе вычисления общего назначения на графических картах, с распределёнными вычислениями.

Глава 3. Программная реализация сервисов моделирования систем газоснабжения на основе разработанной вычислительной среды

3.1. Программная реализация вычислительных сервисов

Механизм создания сервисов на основе выбранных технологий

Базовый метод построения предложенной среды моделирования – создание сервисов.

Сервис состоит из трёх компонентов:

- описания интерфейса на специализированном языке описания интерфейсов *ICE* («*Slice*»);
- класса-исполнителя, написанного на одном из поддерживаемых языков программирования и реализующего все методы, заявленные в интерфейсе сервиса;
- класса-посредника, написанного на языке программирования клиента и предоставляющего возможности синхронного и асинхронного вызова методов сервиса, удалённо выполняемых исполнителем на сервере.

Процесс разработки сервиса, его класса-исполнителя, класса-заместителя представлен на рисунке 3.1.

В состав *ICE* входят компиляторы интерфейсов для различных языков программирования. Компилятор интерфейсов по файлу *Slice* генерирует программный код класса объекта-посредника и программный код заготовки класса-исполнителя (на соответствующем языке программирования). Заготовка класса-исполнителя содержит перечень методов, объявленных в интерфейсе сервиса – их необходимо реализовать разработчику.

Класс-исполнитель и класс-посредник компилируются для соответствующих языков программирования вместе с библиотекой *ICE Runtime*, выполняющей функции промежуточного уровня.

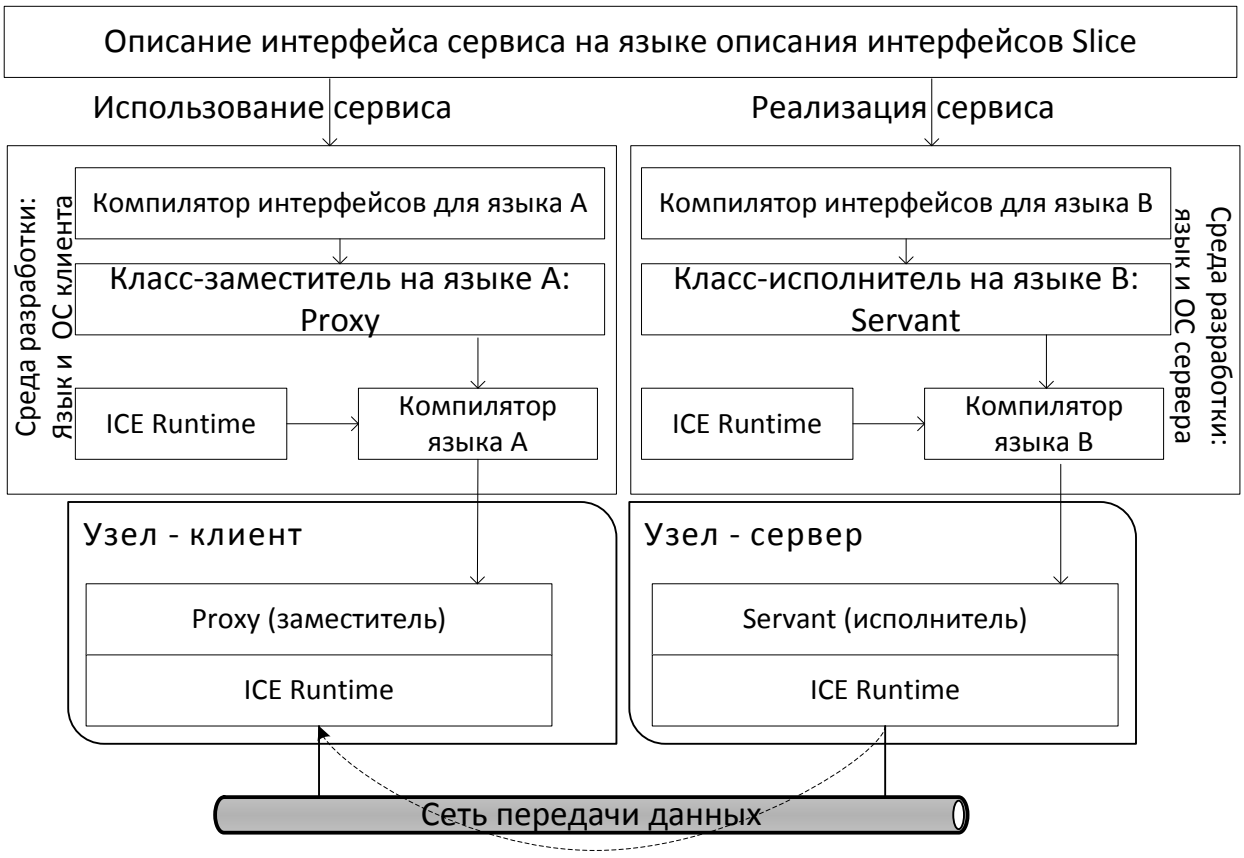

Рисунок 3.1 – Схема разработки сервиса в *ICE*

Скомпилированный класс-исполнитель запускается на сервере. Клиент использует класс-посредник для синхронного или асинхронного вызова методов удалённого исполнителя, работая с ним как с обычным локальным объектом; при этом вызовы методов посредника через *ICE Runtime* передаются на сервер исполнителю, выполняются в нём, а результаты (также через *ICE Runtime*) возвращаются клиенту.

Менеджеры параллельного моделирования расчётных объектов

Абстрактный интерфейс менеджера параллельного моделирования

Общими для всех менеджеров параллельного моделирования расчётных объектов являются следующие методы:

– «*Установить для управляемых дуг множество расчётных параметров*»;

– «*Моделировать все управляемые дуги*»;

– «*Получить множество расчётных параметров управляемых объектов*».

Перечисленные методы отражены в абстрактном интерфейсе, который реализуют все конкретные менеджеры параллельного моделирования.

```
/** Абстрактный интерфейс менеджера параллельного моделирования объектов.*/
#include <vector>             // Вектор используется для хранения объектов
#include "work_params.h"      // Определение типа рабочих параметров
#include "calculated_params.h"// Определение типа расчётных параметров
class ParallelManagerI {
public:
  // Установить рабочие параметры для вектора управляемых объектов
  virtual void SetWorkParams(std::vector<WorkParams> const &work_params) = 0;
  virtual void CalculateAll() = 0; // Моделировать все управляемые объекты
  virtual void GetCalculatedParams(// Вернуть вектор расчётных значений
      std::vector<CalculatedParams> *calculated_params) = 0;
};
```

Каждый менеджер параллельного моделирования объектов типа T_n так же реализует метод «*Принять под управление вектор моделируемых объектов типа* T_n». Этот метод не выносится в общий интерфейс, так как для каждого типа объектов набор паспортных параметров индивидуален.

Диаграмма классов для менеджеров параллельного моделирования представлена на рисунке 3.2.

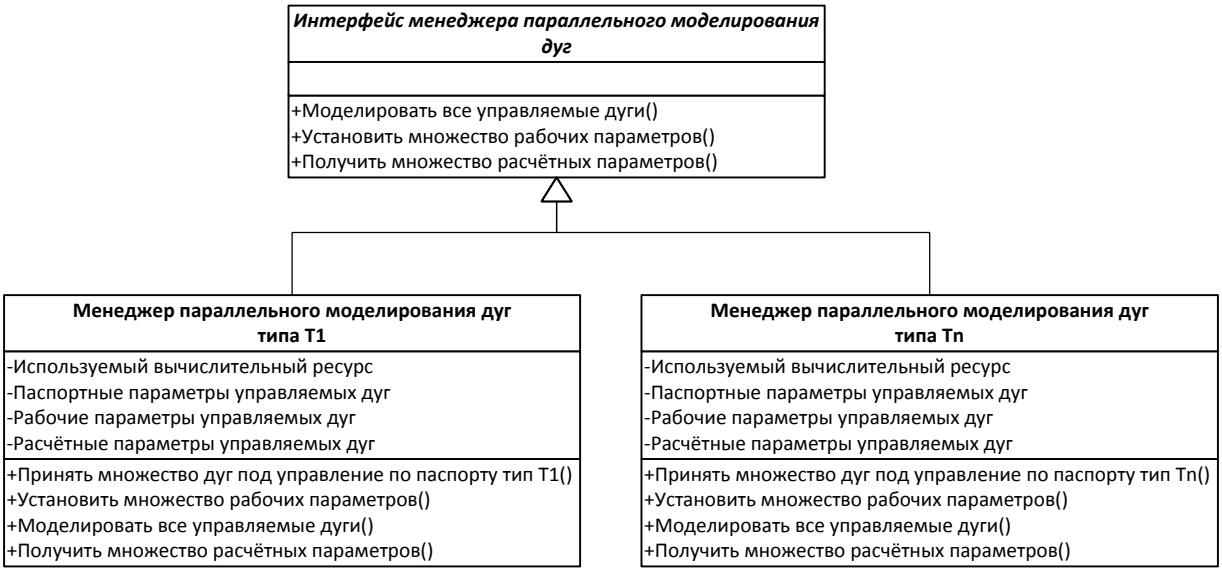

Рисунок 3.2 Общий интерфейс сервисов параллельного моделирования расчётных объектов

Базовый менеджер параллельного моделирования для трубопроводов

Для моделируемого объекта типа «Трубопровод» разработан класс, позволяющий:

– создавать трубопровод по паспорту;
– задавать рабочие параметры;
– производить моделирование трубопровода;
– получать расчётные параметры.

Соответствующая *UML*-диаграмма приведена на рисунке 3.3.

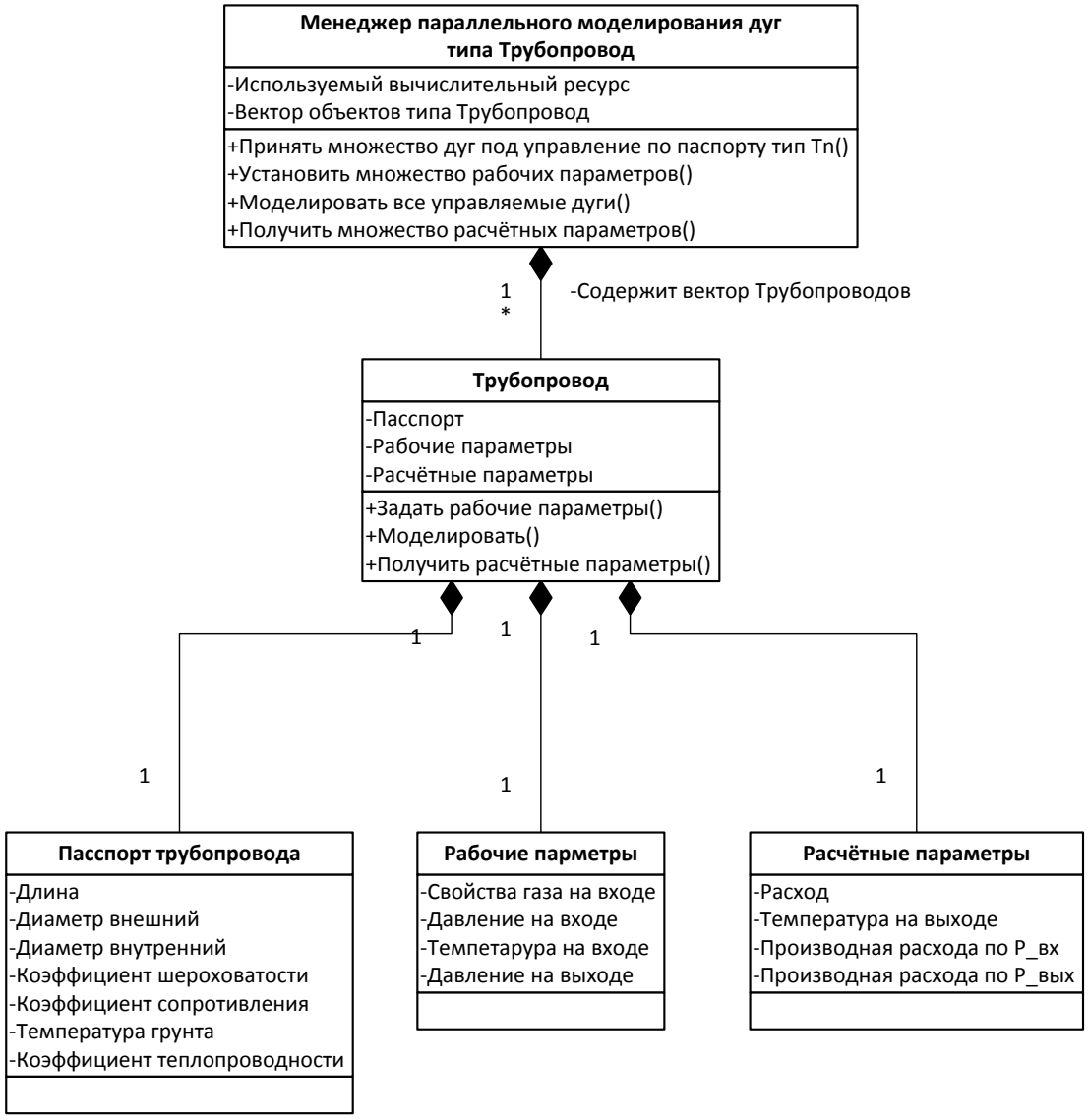

Рисунок 3.3 Базовая реализация менеджера параллельного моделирования для трубопроводов

На базе такого класса менеджер параллельного моделирования объектов типа «Трубопровод» может реализовать необходимые методы следующим образом.

1. При вызове метода «*Принять под управление множество трубопроводов по множеству паспортов*» менеджер параллельного моделирования создаёт внутри себя вектор объектов типа «Трубопровод» и заполняет его объектами, созданными на основании полученных паспортов.
2. При вызове метода «*Задать расчётные параметры*» менеджер вызывает метод «*Задать расчётные параметры*» для каждого объекта в векторе трубопроводов.
3. При вызове метода «*Моделировать все управляемые объекты*» менеджер в цикле вызывает метод «*Моделировать*» для каждого объекта в векторе трубопроводов.
4. При вызове метода «*Получить расчётные значения*» менеджер формирует вектор расчётных параметров на основании вызова соответствующего метода каждого объекта в векторе трубопроводов и возвращает сформированный вектор.

Ниже представлен фрагмент исходного кода реализации базового менеджера параллельного моделирования трубопроводов.

```
/** Реализация базового менеджера параллельного моделирования труб.*/
#include "parallel_manager_pipe_singlecore.h" // заголовочный файл
#include "model_pipe_sequential.h"            // модель трубы
#include <vector>                             // вектор для хранения моделей

// Принять вектор объектов под управление
void ParallelManagerPipeSingleCore::
    TakeUnderControl(std::vector<PassportPipe> const &passports) {
  for(auto p = passports.begin(); p != passports.end(); ++p) {
    PassportPipe passport = *p;
    ModelPipeSequential model(&passport);
    models_.push_back(model);
  }
}

// Установить вектор рабочих параметров
void ParallelManagerPipeSingleCore::
    SetWorkParams(std::vector<WorkParams> const &work_params) {
  auto wp = work_params.begin();
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    Gas gas_in;
```

```
    gas_in.composition.density_std_cond = wp->den_sc_in();
    gas_in.composition.co2 = wp->co2_in();
    gas_in.composition.n2 = wp->n2_in();
    gas_in.work_parameters.p = wp->p_in();
    gas_in.work_parameters.t = wp->t_in();
    m->set_gas_in(&gas_in);

    Gas gas_out;
    gas_out.work_parameters.p = wp->p_out();
    m->set_gas_out(&gas_out);
    ++wp;
  }
}

// Выполнить моделирование всех управляемых объектов
void ParallelManagerPipeSingleCore::CalculateAll() {
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    m->Count();
  }
}

// Вернуть вектор расчётных параметров
void ParallelManagerPipeSingleCore::
    GetCalculatedParams(std::vector<CalculatedParams> *calculated_params){
  calculated_params->erase(calculated_params->begin(), calculated_params-
>end());
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    CalculatedParams cp;
    cp.set_q         ( m->q() );
    cp.set_dq_dp_in  ( m->dq_dp_in() );
    cp.set_dq_dp_out ( m->dq_dp_out() );
    cp.set_t_out     ( m->gas_out().work_parameters.t );
    calculated_params->push_back(cp);
  }
}
```

Реализация менеджера параллельного моделирования трубопроводов на многоядерном процессоре

Методы моделирования всех управляемых объектов базового менеджера параллельного моделирования трубопроводов целесообразно распараллелить, так как управляемые трубопроводы являются независимыми объектами.

В соответствии с выбором средств параллельного программирования такое распараллеливание осуществлено с использованием технологии *OpenMP*. Все соответствующие циклы помечены директивой *#pragma omp parallel for*.

Ниже представлен распараллеленный вариант метода моделирования всех управляемых объектов.

```
void ParallelManagerPipeOpenMP::CalculateAll() {
#pragma omp parallel for
  for(int i = 0; i < models_.size(); ++i) {
    models_[i].Count();
  }
}
```

Обобщённая реализация менеджера параллельного моделирования произвольных расчётных объектов на многоядерном центральном процессоре

Различные моделируемые объекты реализуют единый абстрактный интерфейс, но различаются типом паспортных параметров, как показано на рисунке 3.4.

Различие паспортных параметров объектов затрудняет создание единого полиморфного менеджера параллельного расчёта моделируемых объектов, реализующих интерфейс «Моделируемый объект».

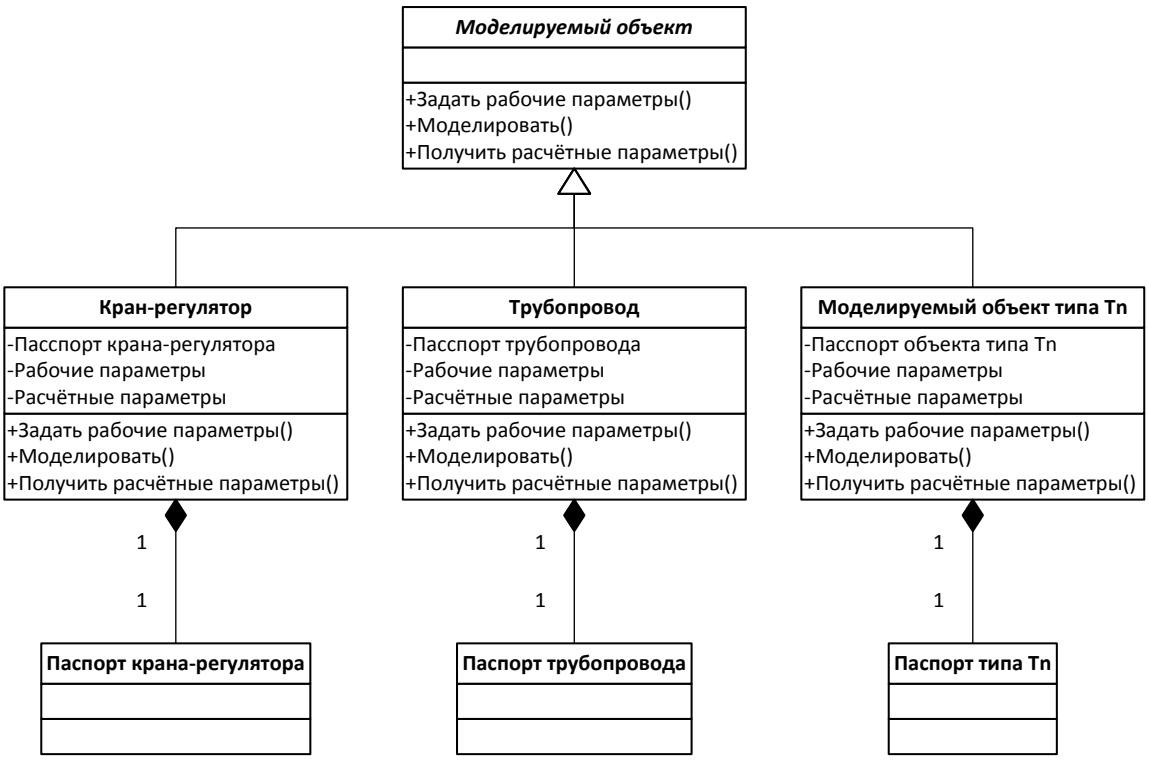

Рисунок 3.4 Единый интерфейс моделируемых объектов

Более целесообразна реализация шаблона класса моделируемого объекта, параметризованного следующими типами:

– тип моделируемого объекта;
– тип паспорта моделируемого объекта.

Такой шаблон класса разработан, с его использованием возможно создание менеджера параллельного моделирования любого объекта, реализующего интерфейс «Моделируемый объект» и допускающего создание на основании паспорта.

Ниже представлен фрагмент исходного кода, иллюстрирующий основные особенности разработанного обобщённого менеджера моделирования.

```
/** Обобщённый менеджер параллельного моделирования*/
#include <vector>                    // Хранение вектора моделей
#include "work_params.h"             // Определение типа рабочих параметров
#include "calculated_params.h"       // Определение типа расчётных параметров
Template <class Model, class Passport>
class GeneralParallelManager : public ParallelManagerPipeI{
public:
 // Принять под управление вектор объектов, допускающих создание по паспорту
 virtual void TakeUnderControl(std::vector<Passport> const &passports) {
   for(auto p = passports.begin(); p != passports.end(); ++p) {
      Passport passport = *p;
      Model model(&passport);
      models_.push_back(model);
    }
 }
 virtual void SetWorkParams(std::vector<WorkParams> const &work_params) {
   // Реализация аналогична приведённой выше
 };
 // Моделировать вектор объектов, поддерживающих метод  Count
 virtual void CalculateAll() {
   #pragma omp parallel for
   for(int i = 0; i < models_.size(); ++i) {
     models_[i].Count();
   }
};
 virtual void GetCalculatedParams(std::vector<CalculatedParams>
*calculated_params) {
   // Реализация аналогична приведённой выше
 };
private:
  std::vector<Model> models_; // Хранение вектора объектов
};
```

Моделируемые объекты в менеджере параллельного моделирования хранятся в векторе, данные управляемых объектов располагаются в памяти в виде массива структур, как представлено на рисунке 3.5.

Расположение данных в памяти в виде массива структур хорошо подходит для расчёта на многоядерном процессоре, так как позволяет процессору эффективно использовать кэш-память.

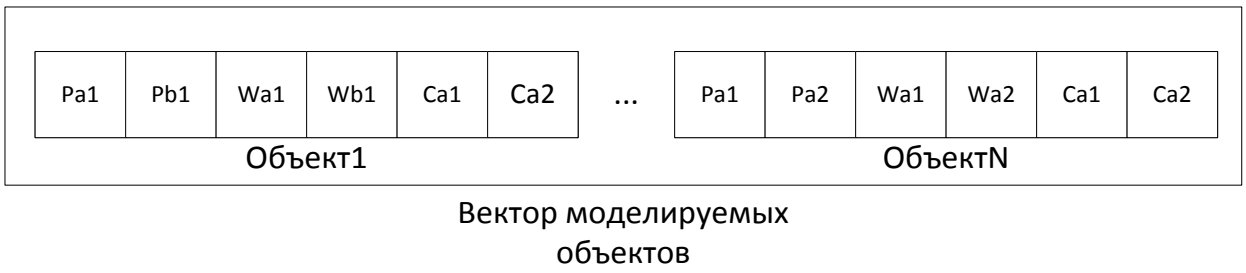

Рисунок 3.5 Расположение данных в виде массива структур; *P* – паспортные параметры, *W* – рабочие параметры, *C* – расчётные параметры

Предложенный шаблон класса менеджера параллельного моделирования предоставляет универсальное решение для организации параллельного моделирования расчётных объектов на центральном процессоре. Однако такое решение не всегда будет наиболее эффективным: в зависимости от типа объекта предпочтительными могут быть другие подходы.

Моделирование векторов объектов на многоядерном центральном процессоре эффективно для объектов, обладающих вычислительно-сложными моделями с большим количеством условных переходов. Для объектов, обладающих простыми вычислительными моделями и высоким количеством можно добиться более производительного параллельного моделирования с использованием высоко-параллельных GPGPU-расчётов.

Обобщённая реализация менеджера параллельного моделирования произвольных расчётных объектов на *CUDA*-устройствах

Обобщённый менеджер параллельного моделирования для *CUDA*-устройств реализован подобно описанному выше менеджеру для центральных процессоров на основе библиотеки обобщённого программирования для *CUDA «thrust»*. Вместо стандартного контейнера вектора *std::vector* используются *thrust::host_vector*, *thrust::device_vector*; вместо стандартного алгоритма *std::for_each – thrust::for_each.*

Для того чтобы моделируемый объект мог быть рассчитан на *GPU,* он должен дополнительно предоставить метод «*Рассчитать на GPU*», помеченный директивой __device__. Параллельное моделирование на GPU осуществляется алгоритмом thrust::for_each, в котором функтор, применяемый ко всем объектам вектора, вызывает метод «*Рассчитать на GPU*».

```
/** Обобщённый менеджер параллельного моделирования на CUDA */
#include <thrust>                    // Вектор thrust
#include <thrust/host_vector>
#include <thrust/device_vector>
#include <thrus/functional>
#include "work_params.h"             // Определение типа рабочих параметров
#include "calculated_params.h"       // Определение типа расчётных параметров

template <class Model>
CountFunctor { // Шаблонный функтор для моделирования объекта
public:
  void operator ()(Model _m) {_m.Count();} };
}

template <class Model, class Passport>
class GeneralParallelManagerThrust : public ParallelManagerPipeI{
public:
 // Моделировать вектор объектов, поддерживающих метод  Count
 virtual void CalculateAll() {
   models_device_ = models_host_; // Отправка данных в память видеокарты
   // Параллельное моделирование объектов на CUDA-устройстве
   thrust::for_each(models_device_.begin(), models_device.end(),
CountFunctor() );
   models_host = models_device_; // Получение результатов из памяти
видеокарты
};
// Реализация остальных функций аналогична приведённой выше
private:
```

```
  thrust::host_vector<Model> models_;          // Хранение вектора управляемых
объектов
  thrust::device_vector<Model> models_device_;// Вектор объектов в памяти
CUDA
};
```

В этой реализации данные так же хранятся в векторе и располагаются в виде массива структур, что слабо подходит для работы с *CUDA*-устройствами: эффективная работа с памятью для *CUDA*-устройств требует размещения данных в виде структуры массивов, позволяя использовать механизм объединения запросов к памяти устройства, многократно ускоряя операции обмена с памятью [**136**]. Такое расположение данных представлено на рисунке 3.6.

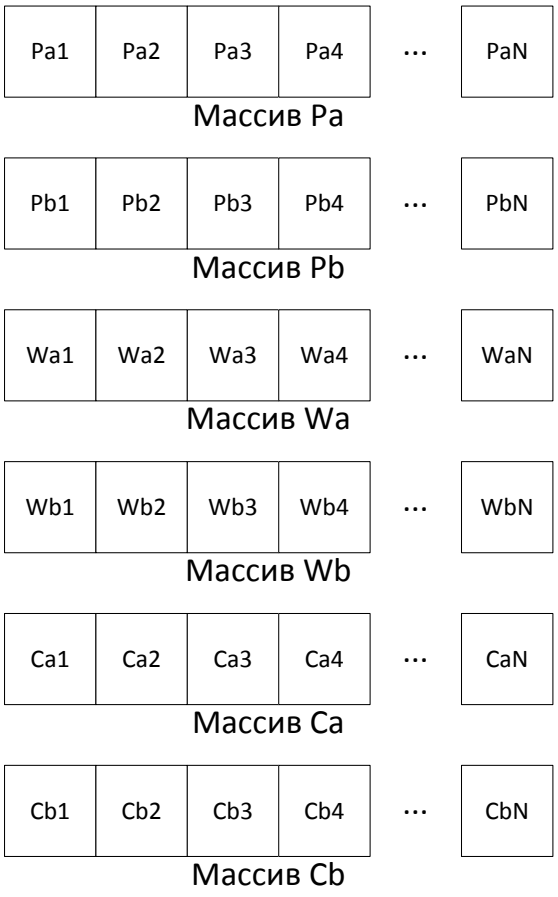

Рисунок 3.6 Расположение данных в виде структуры массивов; P – паспортные параметры, W – рабочие, C – расчётные

Оптимизированная реализация менеджера параллельного моделирования трубопроводов на *CUDA*-устройствах

Разработан оптимизированный менеджер параллельного *GPGPU*-моделирования трубопроводов, располагающий данные в виде структуры массивов.

Менеджер реализован с использованием *CUDA C*. Схема его работы представлена на рисунке 3.7.

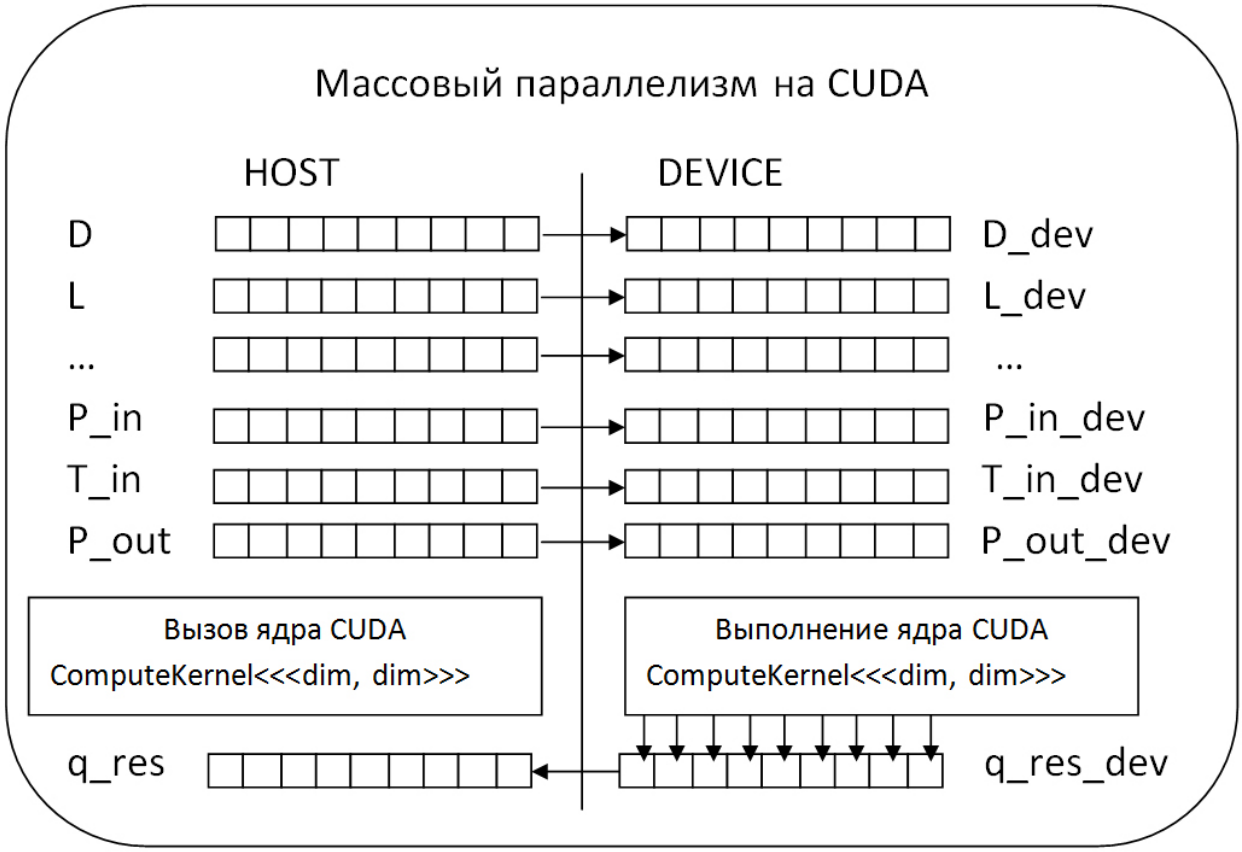

Рисунок 3.7 Схема работы оптимизированного менеджера параллельного моделирования трубопроводов; *host* – компьютер, *device* – *CUDA*-устройство

1. Для выполнения метода «Принять под управление» менеджер создаёт в памяти компьютера (*host*) и *GPU*-устройства (*device*) структуры массивов для хранения паспортных, рабочих и расчётных параметров, отправляет в память *device* паспортные параметры.
2. Для выполнения метода «Задать расчётные параметры» менеджер реорганизует полученный вектор рабочих параметров в соответствующий массив структур и отправляет их на *device*.

3. Для выполнения метода «Моделировать все объекты» менеджер вызывает вычислительное *CUDA*-ядро «*ComputeKernel*», передавая ему в качестве параметров указатели на паспортные, рабочие и расчётные параметры в памяти *device*. Вычислительное ядро осуществляет вычисления и записывает результаты в массив расчётных параметров в памяти *device*.
4. Для выполнения метода «*Получить расчётные параметры*» менеджер копирует расчётные параметры из памяти устройства, реорганизует их в вектор расчётных параметров и возвращает его в качестве результата метода.

Программный код оптимизированного менеджера представлен в приложении А.

Сервисы решения систем линейных алгебраических уравнений

Системы уравнений, возникающие при решении задач моделирования газотранспортных систем, как правило, являются разреженными. По этой причине общий абстрактный интерфейс класса «Решатель системы уравнений» использует передачу параметров в разреженном формате.

Схема классов представлена на рисунке 3.8.

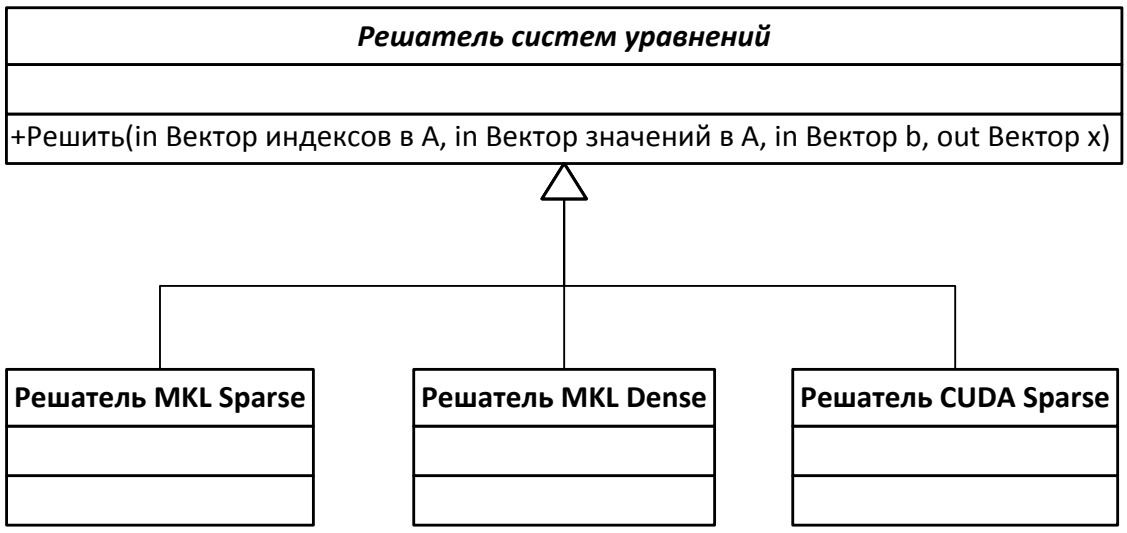

Рисунок 3.8 Интерфейс решателя систем линейных уравнений

Метод «*Решить*» имеет следующие параметры:

1. Вектор индексов A – вектор индексов ненулевых элементов матрицы A (его размерность равна количеству ненулевых элементов в A).
2. Вектор значений A – вектор значений ненулевых элементов в A в порядке, соответствующем порядку, в котором перечислены элементы в векторе индексов.
3. Вектор b – вектор правой части – его размерность равна количеству строк в матрице A.
4. Вектор x – через этот параметр возвращается решение системы уравнений.

Ниже представлено описание абстрактного интерфейса решателя СЛАУ.

```
/** Абстрактный интерфейс решателя СЛАУ SlaeSolverI. */
#pragma once
#include <vector>
class SlaeSolverI {
 public:
   virtual void Solve( // Решить систему уравнений
     std::vector<int> const &A_indexes,  // Вектор индексов значимых коэф-в
     std::vector<double> const &A_values,// Вектор значений значимых коэф-в
     std::vector<double> const &B,       // Вектор правой части
     std::vector<double> *X              // Решение – out-параметр
  ) = 0;
};
```

Разработан ряд сервисов, реализующих этот интерфейс:

– на основе библиотеки решения плотных матриц на центральном процессоре из *Intel MKL (Math Kernel Library)*;
– на основе библиотеки решения разреженных матриц на центральном процессоре из *Intel MKL*;
– на основе библиотеки решения разреженных матриц на *CUDA*-устройствах - *Cusp* (от *CUDA Sparse*) [**101**].

Каждый из разработанных сервисов осуществляет преобразование полученных параметров к формату, необходимому для вызова библиотечной функции, производит решение с использованием библиотечной функции, и возвращает полученный результат. Их программные коды приведены в приложении А.

3.2. **Проведение и анализ вычислительных экспериментов**

Механизм тестирования производительности

Для проведения тестов производительности автором разработан специализированный механизм, схема которого представлена на рисунке 3.9.

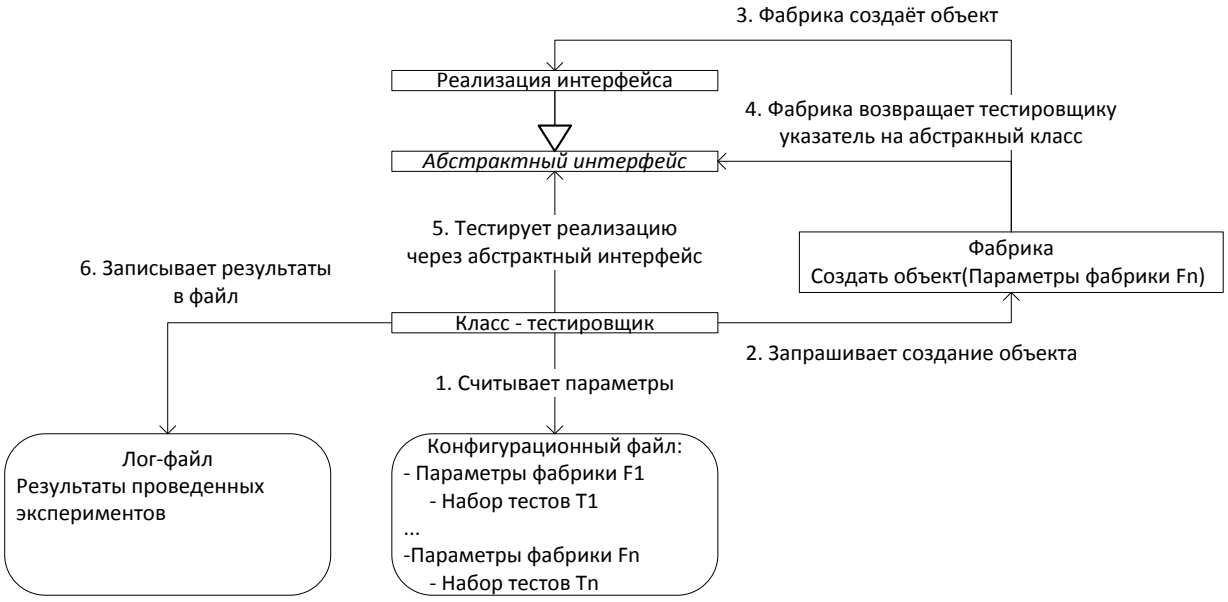

Рисунок 3.9 Схема механизма тестирования

Тесты разрабатываются для абстрактных интерфейсных классов, таким образом, все классы, реализующие общий интерфейс, проходят один и тот же тест.

Все разработанные тесты осуществляют многократное выполнение тестируемых методов с последующим усреднением полученных замеров времени работы. После выполнения тестируемого метода проверяется корректность результата его работы.

Для каждого теста абстрактного интерфейса конкретный набор тестируемых сервисов, параметры их работы определяются конфигурационным файлом, имеющим иерархическую структуру:

- уровень абстрактного класса: тестировать класс, или нет; имя лог-файла;
- уровень конкретного класса: параметры, передаваемые фабрике объектов для получения конкретного класса;

– уровень отдельных тестов: количество повторов, размерность тестовых примеров.

Результаты выполнения тестов записываются в лог-файлы.

Тестирование осуществляется по следующему алгоритму.

1. Класс тестировщик считывает из конфигурационного файла параметры фабрики F_1.
2. Тестировщик запрашивает у фабрики объектов создание объекта, передавая фабричному методу считанные параметры F_1.
3. Фабрика создаёт конкретный объект на основании полученного параметра.
4. Фабрика возвращает тестировщику указатель на абстрактный интерфейс, указывающий на созданный конкретный объект.
5. Тестировщик считывает из конфигурационного файла набор тестов T_1. и проводит тестовые испытания реального объекта через абстрактный интерфейс.
6. Тестировщик записывает результаты тестирования в лог-файл для последующего анализа.

Наборы тестовых примеров

В качестве базового тестового примера использовалась схема трубопроводной системы Саратов – Горький, которая включает в себя 155 моделируемых объектов, 48 краевых узлов (поставщики/потребители), 18 идентифицируемых параметров модели режима $K_{\text{то}}, K_{\text{эф}}$. На основе базовой схемы построены тестовые примеры размерности, превосходящей исходную до двухсот раз.

Общая схема трубопроводной системы «Саратов – Горький», загруженная в интерфейсе ПВК «Веста», представлена на рисунке 3.10.

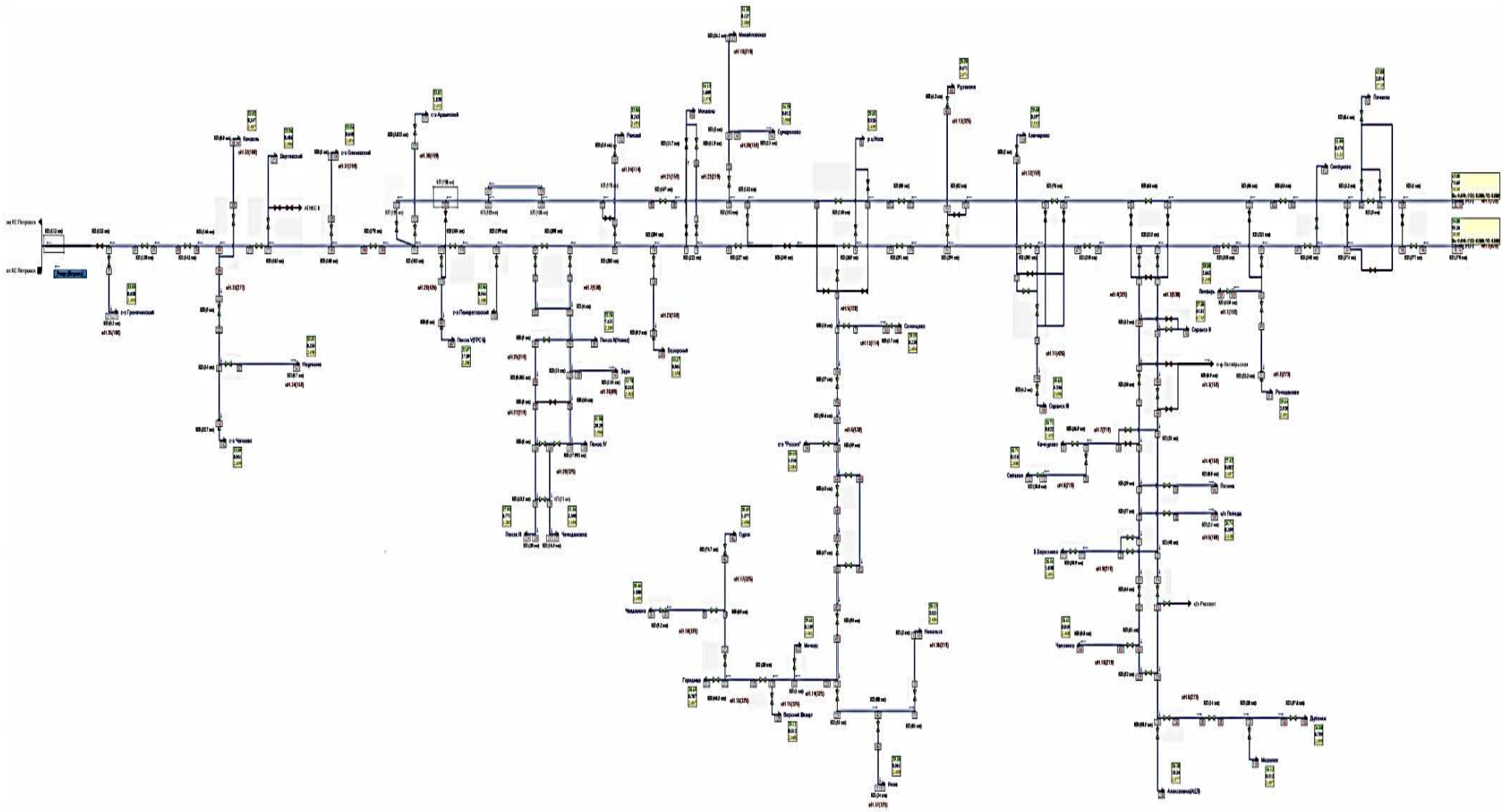

Рисунок 3.10 Схема Саратов-Горький

Формируемые тестовые примеры обрабатывались следующими алгоритмами.

– Параллельное моделирование трубопроводов:
  - базовый менеджер моделирования;
  - менеджер параллельного моделирования, использующий *OpenMP*;
  - менеджер параллельного моделирования, использующий оптимизированную реализацию на *CUDA*.
– Решение систем линейных уравнений:
  - решатель систем на базе библиотеки *MKL* для плотных матриц;
  - решатель систем на базе библиотеки *MKL* для разреженных матриц;
  - решатель систем на базе библиотеки *Cusp* (на *CUDA*) для разреженных матриц.
– Расчёт режима перекачки газа по ГТС:
  - с использованием менеджера параллельного моделирования и решателя систем уравнений на центральных процессорах;
  - с использованием менеджера параллельного моделирования и решателя систем уравнений на *CUDA*-устройствах.
– Расчёт множества режимов ГТС при различных значениях вектора параметров адаптации.

Эксперименты проводились в вычислительных конфигурациях, представленных в таблице 3.1.

Таблица 3.1 – Вычислительные конфигурации при проведении экспериментов

Форм-фактор	**Процессор**	**CUDA-устройство**
Ноутбук	*Intel core i5, 4 ядра*	*GeForce GTX 460M*
Персональный компьютер	*Intel core i7, 8 ядер*	*2xGeForce GTX 460*
Виртуальная машина в облаке *Amazon Web Services Elastic Compute Cloud*	*2xIntel Xeon X5570, 4 ядра*	*2xTesla M2050*
Группа из четырёх виртуальных машин в облаке	*2xIntel Xeon X5570, 4 ядра*	*2xTesla M2050*

Вычислительные эксперименты производились в следующих режимах:

- с использованием локальных классов в качестве реализаций алгоритмов;
- с использованием сервисов, расположенных на сервере тестирующего компьютера;
- с использованием сервисов, расположенных на сервере в локальной сети (тестировщик – *Notebook*, сервер – *Desktop*);
- с использованием сервисов, расположенных на сервере в облаке (тестировщик – *Notebook*, сервер – *AWS*);

Вычислительные эксперименты были проведены для всех возможных комбинаций перечисленных параметров.

Анализ результатов вычислительных экспериментов

Параллельные вычисления для моделирования вектора расчётных объектов

Чтобы оценить эффект от использования параллельных вычислений в решении задачи параллельного моделирования вектора расчётных объектов рассмотрены результаты экспериментов, проведённые локально в конфигурации *Desktop* (8-ядерный процессор, 2 *CUDA GPU*). Результаты представлены в таблице 3.2.

Таблица 3.2 – Параллельные вычисления для моделирования расчётных объектов; время приведено в миллисекундах

Задача	**Исходное время**	***OpenMP***		***CUDA*-устройство**		**Два *CUDA*-устройства**	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
Саратов	14	18	0,77	17	0,82	18	0,77
10xСаратов	120	27	4,44	21	5,71	21	5,71
50xСаратов	588	117	5,02	40	14,7	19	30,94
100xСаратов	1177	230	5,11	76	15,48	37	31,81
200xСаратов	2345	456	5,14	145	16,172	76	30,85

Для небольших задач ускорение практически не достигается, возможно замедление. Но для задач высокой размерности ускорение становится многократным при использовании многоядерного процессора, и ещё более значительным при использовании *CUDA GPU.*

Переход от использования одного *GPU* к двум осуществлён следующим образом: вектор моделируемых параметров разделён на равные части, каждая из которых передана одному из *GPU*. Видно, что производительность при этом растёт практически линейно.

Таким образом, в рамках одного и того же компьютера достигается многократное ускорение моделирования векторов расчётных объектов с использованием гетерогенных параллельных вычислений, в частности, вычислений общего назначения на графических картах.

Расчёт множества режимов ГТС при различных значениях идентифицируемых параметров в распределённой среде

Основным вычислительным элементом задачи идентификации эмпирических параметров модели СГ является расчёт режимов системы газоснабжения при различных значениях вектора идентифицируемых параметров.

В качестве вычислительного эксперимента был проведён параллельный расчёт 200 режимов схемы Саратов-Горький при различных значениях вектора эмпирических параметров модели.

Полученные результаты приведены в таблице 3.3.

Результаты проведённого вычислительного эксперимента показывают:

– эффективность использования удалённых вычислительных ресурсов высокой производительности для решения вычислительно сложных задач: по сравнению с локальным расчётом на *Notebook* решение задачи проведено в 3,31 раза быстрее на более мощной рабочей станции *Desktop* и в 2.6 раза быстрее на удалённом вычислительном ресурсе *AWS;*

Таблица 3.3 – Расчёт схемы Саратов-Горький при различных значениях вектора эмпирических параметров модели; время в секундах

Выполнение расчётов	**Время на клиенте**	**Время на сервере**	**Задержка**	**Доля накладных расходов**	**Ускорение**
Локально на *Notebook*	155,53	155,531	0	0%	1
В локальной сети на *Desktop*	46,88	46,11	0,77	1,64%	3,31
В облаке *AWS* на одной виртуальной машине	59,6	58,67	0,93	1,56%	2,60
В облаке *AWS* на группе из 4-х виртуальных машин	14,9	15,05	1,23	7,43%	9,56

– масштабируемость решения вычислительно-сложных задач, основанных на многократном моделировании режимов СГ вычислительным сервисом моделирования СГ в распределённой среде: решение задачи сократилось с 58,67 секунд до 15,05 секунд при переходе от использования одной виртуальной машины *AWS* к кластеру из четырёх виртуальных машин;
– целесообразность использования облачных вычислительных ресурсов: не смотря на существенную долю накладных расходов удалённого взаимодействия (7,43 %) расчёт на кластере из четырёх облачных виртуальных машин в целом осуществляется в 9,56 раз быстрее, чем локальный расчёт на *Notebook.*

Полученные результаты показывают эффективность предложенных архитектурных решений и их программной реализации.

3.3. **Интеграция разработанных вычислительных сервисов с различными программно-вычислительными комплексами**

Элементы методологии структурной, программной и алгоритмической модернизации ПВК СППДР для перехода к использованию вычислительных сервисов

Изначально ПВК СППДР использует «монолитный» расчётный модуль через специально созданный для обеспечения этого взаимодействия интерфейс: текстовые файлы определённого формата, локальные базы данных и разделяемую память. Такое состояние системы изображено на рисунке 3.11.

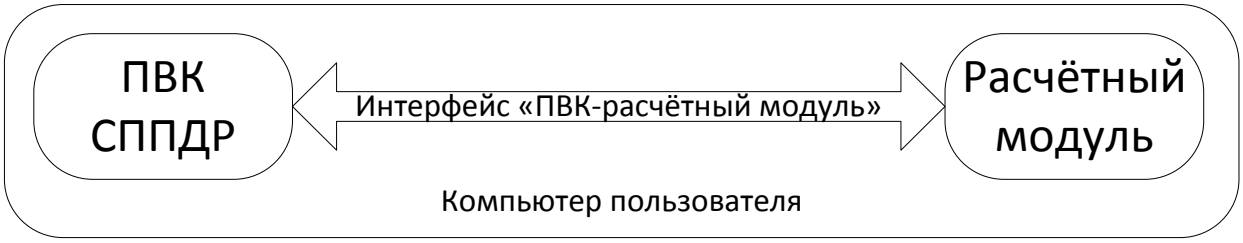

Рисунок 3.11 Использование ПВК СППДР монолитного расчётного модуля

Методология перехода ПВК СППДР к использованию новых вычислительных сервисов состоит из двух этапов.

На первом этапе вводится модуль-адаптер по схеме, представленной на рисунке 3.12.

На этом этапе монолитный расчётный модуль прозрачно для интерфейса ПВК подменяется модулем «Адаптер», способным взаимодействовать с интерфейсом ПВК с использованием текущего протокола.

«Адаптер» определяет, может ли запрошенная ПВК задача быть выполнена с использованием разработанных вычислительных сервисов, локальных или удалённых. В случае, если требуемый сервис разработан, адаптер использует его для выполнения задачи.

Если требуемый вычислительный сервис пока не разработан, задача переадресовывается старому расчётному модулю.

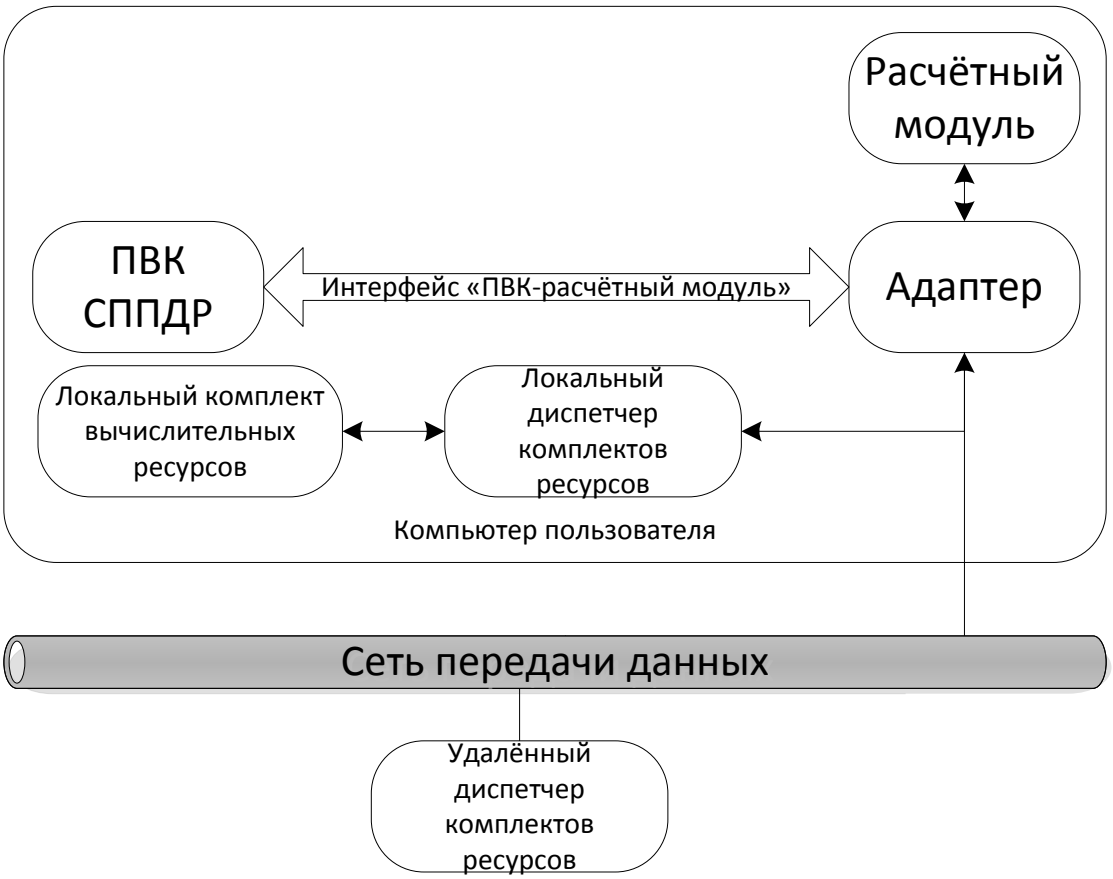

Рисунок 3.12 Этап перехода к использованию новых вычислительных сервисов

На этом этапе происходит разработка, тестирование и внедрение новых вычислительных сервисов, постепенное сокращение доли задач, решаемых старым расчётным модулем.

На втором этапе функциональность адаптера включается в ПВК СППДР, комплекс начинает напрямую работать с новыми вычислительными сервисами, используя предоставленный программный интерфейс. Состояние системы на этом этапе изображено на рисунке 3.13.

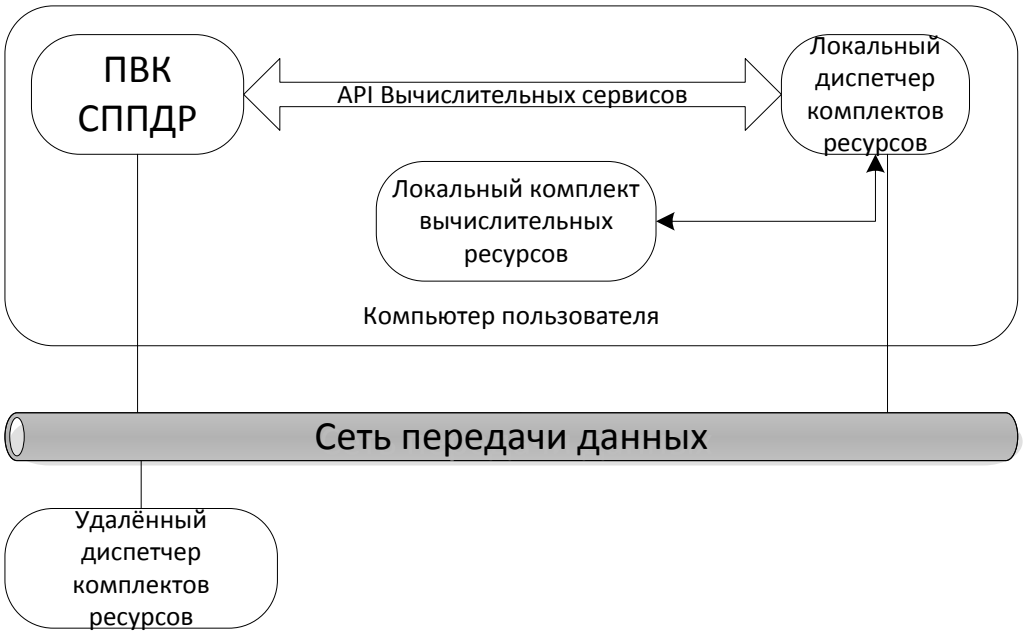

Рисунок 3.13 – Переход ПВК СППДР к использованию новых вычислительных сервисов

Предоставление вычислительных сервисов для ПВК СППДР

Модель *Software-as-a-Service* (программное обеспечение как услуга)

При этом подходе разработчикам клиентского комплекса предоставляются *API*-классы (*API – Application Programming Interface*) на языке разработки их программного комплекса, которые они используют для решения вычислительных задач. При этом *API* настраивается на использование удалённого диспетчера вычислительных ресурсов сервера, поддерживаемого разработчиками вычислительных сервисов.

API осуществляет удалённое взаимодействие с использованием бинарного протокола *ICE*.

Преимущества подхода:

- Простота установки у клиента: нет необходимости устанавливать и настраивать служебные и вычислительные сервисы на оборудовании клиента.
- Простота обновлений, не затрагивающих интерфейса: реализация функционала сосредоточена на сервере, управляемом разработчиками, что

позволяет им централизованно и прозрачно для клиента вносить изменения в реализации сервисов, если они не затрагивают интерфейсов.

Недостатки подхода:

– Клиент вынужден работать с удалёнными вычислительными ресурсами, соединение с которыми менее надёжно и быстро, нежели локальное.
– Вычислительные ресурсы клиента, возможно, используются не полностью.
– Увеличивается нагрузка на вычислительный сервер.

Установка вычислительных сервисов на оборудовании клиента

При этом подходе осуществляется установка и настройка вычислительной среды на оборудовании клиента: на локальном оборудовании настраиваются служебные сервисы управления комплектами вычислительных ресурсов, сервис диспетчер комплектов вычислительных ресурсов. *API*-классы, используемые разработчиками комплекса, настраиваются на работу с локальным диспетчером комплектов вычислительных ресурсов.

Клиенты могут комбинировать этот подход с предыдущим и дополнительно использовать удалённые вычислительные ресурсы.

Для централизованного распространения обновлений программных компонентов, установленных на оборудовании клиента, может быть использован механизм *IcePatch2,* входящий в *ICE*.

Преимущества подхода:

– клиенты работают с локальными ресурсами, что сокращает издержки, связанные с передачей данных:
– более полно используются вычислительные ресурсы клиента;
– снижается нагрузки на «облачный» вычислительный сервер;

Недостатки:

– относительная сложность установки и поддержки.

***Web*-сервисы**

Для ситуаций, в которых использование основного *API* вычислительных сервисов невозможно, например, для разработки «тонких» клиентов для отдельных задач возможно предоставление доступа через web-сервисы.

Сервер приложений, предоставляющий доступ к web-сервису, выступает в роли адаптера между клиентом и вычислительным сервисом. Для клиента *API* определяется *WSDL*-контрактом, а набор доступных сервисов ограничивается сервисами без сохранения состояния удалённого вычислительного ресурса.

***Web*-интерфейс**

В стандарты, регламентирующие работу вычислительных функций, иногда включают простейшее приложение «калькулятор», позволяющее рассчитать значение стандартизуемой функции при заданных параметрах.

В связи с этим может быть целесообразным предоставление web-интерфейса к основным реализованным в новой среде стандартным функциям, математическим моделям для использования в качестве «калькулятора».

Интеграция с ПВК «Веста-тренажёр»

ПВК «Веста-тренажёр» – диспетчерский тренажёрный программно-вычислительный комплекс – разработан на кафедре Автоматизированных систем управления РГУ нефти и газа имени И.М. Губкина и внедрён в ПДС ООО «Газпром трансгаз Санкт-Петербург, Саратов, Уфа», ООО «Газпром добыча Ноябрьск».

На данный момент разработанный вычислительный сервис расчёта режима ГТС интегрирован с ПВК «Веста» в тестовом режиме. Реализована возможность взаимозаменяемого использования текущего расчётного модуля или нового вычислительного сервиса.

Реализована возможность просмотра невязок «эталонного» решения, полученного текущим расчётным модулем, и решения полученного разработанным вычислительным сервисом в интерфейсе ПВК «Веста».

Решения, получаемые новым модулем соответствуют «эталонными», что обусловлено применением одинаковых алгоритмов и вычислительных методов.

Результаты, изложенные в данной главе, завершают решение задач диссертационного исследования.

1. Осуществлена программная реализация мультизадачной вычислительной среды, её ядра – управляющих сервисов – и ряда вычислительных сервисов её поддержки.
2. Проведены вычислительные эксперименты, показавшие корректность и высокую эффективность предложенных архитектурных решений и их программной реализации в виде управляющих служебных и вычислительных сервисов. Анализ экспериментов показал многократное ускорение расчётов как за счёт использования гетерогенных параллельных вычислений, так и за счёт разделения вычислительной нагрузки в распределённой среде, в частности, при использовании облачных вычислительных ресурсов *Amazon Web Services Elastic Compute Cloud*.
3. Разработаны элементы методологии архитектурной, структурной, алгоритмической модернизации ПВК СППДР для перехода к использованию вычислительного обеспечения в виде сервисов. На базе предложенной методологии в тестовом режиме осуществлена интеграция с ПВК «Веста-тренажёр».

Основные результаты и выводы

1. Спроектирована и реализована компонентная мультизадачная вычислительная среда, позволяющая осуществить унифицированное вычислительное обеспечение СППДР АСДУ на основе параллельных и распределённых вычислений, стандартных протоколов информационного взаимодействия, что является предпосылкой для создания единого вычислительного пространства АСДУ ЕСГ.
2. Впервые разработана архитектура и программно реализован базовый вычислительный сервис итерационного моделирования режимов систем газоснабжения, на основе распараллеленного расчета режимов объектов СГ, с использованием вычислений общего назначения на графических картах.
3. Разработана архитектура и программно реализован вычислительный сервис, организующий параллельное моделирование множества режимов СГ в распределённой среде, на примере многоуровневой итерационной процедуры решения задачи идентификации эмпирических параметров модели режимов СГ.
4. Разработанные сервисы параллельного и распределенного моделирования впервые предоставили возможность решать задачи совместного моделирования единых технологических режимов не только систем большой размерности (тысячи и десятки тысяч расчетных элементов), примерами которых являются региональные газотранспортные системы, но и систем газоснабжения, объединяющих такие разнородные технологические комплексы, как системы добычи, транспорта, подземного хранения и распределения газа.
5. Выполнено тестирование разработанных вычислительных сервисов в однопользовательском, сетевом, распределённом режимах, на примере реальной системы газоснабжения, показавшее:
   - корректность результатов разработанных вычислительных сервисов;

- многократное ускорение этапа моделирования объектов СГ сервисом моделирования режимов СГ за счёт использования вычислений общего назначения на графических картах с использованием технологии CUDA;
- масштабируемость многоуровневых итерационных процессов, основанных на многократном моделировании режимов СГ в распределённой среде – как в локальной сети, так и при использовании удалённых «облачных» вычислительных ресурсов AWS Elastic Compute Cloud, несмотря на заметные накладные расходы удалённого взаимодействия (порядка 10% от общего времени расчёта).

6. Разработаны унифицированные компоненты методологии архитектурной и программной модернизации ПВК моделирования СГ для их последующей интеграции с разработанными вычислительными сервисами, а также в единое информационное и вычислительное пространство АСДУ.
7. В тестовом режиме выполнена интеграция разработанных вычислительных сервисов с ПВК «Веста», внедрённым в ПДС ООО «Газпром трансгаз Санкт-Петербург, Саратов, Уфа», ООО «Газпром добыча Ноябрьск».

Список литературы

1. Альтшуль А.Д., Киселев П.Г. Гидравлика и аэродинамика. М., Изд. лит. по строительству, 1965. 273 с.
2. Атавин А.А., Карасевич А.М., Сухарев М.Г. Трубопроводные системы энергетики: модели, приложения, информационные технологии. М., ГУП Издательство «Нефть и газ» РГУ нефти и газа им. И.М. Губкина, 2000. 320 с.
3. Баландин И.А., Антипов Е.И., Раушкин Ю.В., Джураев Э.Ш., Жагфаров И.Ф., Пирогов А.В. Модернизация автоматической системы диспетчерского управления единой системой газоснабжения ОАО "Газпром" // Сб. Тез. международной научно-технической конференции ДИСКОМ 2012. 2012.
4. Баландин И.А. Значение современных средств автоматизации и систем поддержки принятия диспетчерских решений в диспетчерском управлении ЕСГ России // Сб. Тез. международной научно-технической конференции ДИСКОМ 2012. 2012.
5. Басакер Р., Саати Т. Конечные графы и сети. М., Наука, 1974. 368 с.
6. Белоусов В.Д., Блейхер Э.М., Немудров А.Г., Юфин В.А., Яковлев Е.И. Трубопроводный транспорт нефти и газа. М., Недра, 1978. 407 с.
7. Берман Р.Я., Бобровский С.А., Галиуллин З.Т. Оптимизация режимов работы закольцованных магистральных газопроводов // Газовая промышленность. 1967. № 3.
8. Берман Р.Я., Вольский Э.Л. Применение ЭВМ при эксплуатации газотранспортных систем. М., ВНИИЭГазпром, 1969. 75 с.
9. Бобровский С.А., Черникин В.И. Применение метода последовательной смены стационарных состояний для решения задач о переходных процессах // Известия вузов. Нефть и газ. 1963. № 2. С. 87-91.

10. Бобровский С.А., Щербаков С.Г., Яковлев Е.И. Трубопроводный транспорт газа. М., Наука, 1976. 475 с.
11. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М., ДМК Пресс, 2010. 232 с.
12. Борисов С.К., Даточный В.В. Гидравлические расчёты газопроводов. М., Недра, 1972. 112 с.
13. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М., Бином, СПб: Невский диалект, 1998. 560 с.
14. Вапник В.П., редактор. Алгоритмы и программы восстановления зависимостей. М., Наука, 1984. 816 с.
15. Васильев А.В., Леонов Д.Г., Сарданашвили С.А., Швечков В.А. Архитектурные решения интеграции расчетных комплексов моделирования режимов систем газоснабжения в единое информационное пространство автоматизированной системы диспетчерского управления ОАО «ГАЗПРОМ» // Газовая промышленность. Декабрь 2012. Т. 683, № 12. С. 62-66.
16. Васильев А.В., Сарданашвили С.А. Программная архитектура для использования параллельных вычислений при численном балансировании потоков газа на расчётном графе газотранспортной системы // Автоматизация, телемеханизация и связь в нефтяной промышленности. 2012. № 1. С. 38-42.
17. Васильев А.В., Сарданашвили С.А. Программная архитектура для эффективного использования гетерогенных параллельных вычислений при численном моделировании газотранспортных систем // Тезисы докладов девятой Всероссийской конференции Новые технологии в газовой промышленности. 2011. С. 14.
18. Васильев А.В., Сарданашвили С.А. Программная архитектура расчётно-вычислительного комплекса поддержки диспетчерского управления

режимами газотранспортных систем // Сб. Тез. Конференции актуальные проблемы развития нефтегазового комплекса России - 2012. 2012. С. 92.

19. Васильев А.В., Швечков В.А., Митичкин С.К. Технологии параллельных вычислений в ПВК «Веста» моделирования режимов систем газоснабжения // V Международная научно-техническая конференция ДИСКОМ-2012. 2012.
20. Васильев А.В. Проектирование и реализация компонентной среды параллельного и распределённого моделирования систем газоснабжения // Автоматизация, телемеханизация и связь в нефтяной промышленности. 2013. № 2. С. 32-38.
21. Васильев А.В. Эффективное численное моделирование транспорта газа многопользовательским сервером с использованием пулов типизированных ресурсов // Сб. Тез. 66-й Международной молодежной научной конференции Нефть и газ 2012. 2012. С. 8.
22. Воеводин В.В. Параллельные вычисления. СПб., БХВ-Петербург, 2004.
23. Вольский Э.Л., Константинова И.М. Режим работы магистрального газопровода. Л., Недра, 1970. 168 с.
24. Всё о мире суперкомпьютеров и параллельных вычислений [Электронный ресурс] // PARALLEL.RU – информационно-аналитический центр по параллельным вычислениям: [сайт]. URL: http://www.parallel.ru (дата обращения: 17.02.2013).
25. Гамм А.З., Голуб И.И. Наблюдаемость электроэнергетических систем. М., Наука, 1990. 200 с.
26. Горелова В.Л., Мельникова Е.Н. Основы прогнозирования систем. М., Высш. шк., 1986.
27. Григорьев Л.И., Владимиров А.И. Компьютерные технологии профессиональной подготовки инженерных кадров // Высшее образование в России. 1995. № 4.

28. Григорьев Л.И., Митичкин С.К. Организация информационного обеспечения тренажёра диспетчера ГТС // Газовая промышленность. 1988. № 4. С. 32-35.
29. Григорьев Л.И., Сарданашвили С.А., Герке В.Г. Основные проблемы теории диспетчерского управления // Газовая промышленность. 2002. № 12.
30. Григорьев Л.И., Сарданашвили С.А., Дятлов В.А. Компьютеризированная система подготовки диспетчерского персонала в транспорте газа. М., Изд-во "Нефть и газ", 1996. 195 с.
31. Григорьев Л.И., Сарданашвили С.А., Вербило А.С., Герке В.Г. Теоретические и практические аспекты подготовки диспетчеров газотранспортных обществ // Газовая промышленность. 2003.
32. Гроп Д. Методы идентификации систем. М., Мир, 1979. 302 с.
33. Д. Бэкон Т.Х. Операционные системы. Параллельные и распределенные системы. СПб., Питер; Киев: Издательская группа BHV, 2004. 800 с.
34. Дейч А.М. Методы идентификации динамических объектов. М., Энергия, 1979. 240 с.
35. Жидкова М.А. Переходные процессы в магистральных газопроводах. Киев, Наук. думка, 1979. 256 с.
36. Идельчик И.Е. Справочник по гидравлическим сопротивлениям. Л., Госэнергоиздат, 1960. 464 с.
37. Кашьяп Р.Л., Рао А.Р. Построение динамических стохастических моделей по экспериментальным данным. М., Наука, 1983. 384 с.
38. Кочуева О.Н. Методы гидравлического расчёта закольцованных трубопроводных систем в задачах оперативного управления для подготовки специалистов // Автоматизация, телемеханизация и связь в нефтяной промышленности. 1995. № 6. С. 20-23.
39. Кристофидес Н. Теория графов. Алгоритмический подход. М., изд-во "Мир", 1978. 429 с.

40. Леонов Д.Г., Васильев А.В., Митичкин С.К., Швечков В.А. Архитектурные решения создания единого вычислительного пространства распределённой многоуровневой СППДР, основанные на эволюционном развитии клиент-серверной среды ПВК «Веста» // V Международная научно-техническая конференция ДИСКОМ-2012. 2012.
41. Леонов Д.Г., Швечков В.А. Организация хранения данных в распределенном вычислительном комплексе при решении задач диспетчерского управления режимами ГТС // Автоматизация, телемеханизация и связь в нефтяной промышленности. 2005. № 9. С. 29-33.
42. Леонов Д.Г. Анализ применения объектно-ориентированного подхода к проектированию экспертных обучающих систем // Автоматизация, телемеханизация и связь в нефтяной промышленности. 1995. № 6. С. 10-13.
43. Леонов Д.Г. Объектно-ориентированная технология разработки систем поддержки принятия диспетчерских решений в транспорте газа // Автоматизация, телемеханизация и связь в нефтяной промышленности. 2000. № 5. С. 11-17.
44. Максимов Ю.И. Расчёт и оптимизация эксплуатационных режимов работы и параметров газоснабжающих систем // Экономика, организация и управления в газовой промышленности. 1971.
45. Меренков А.П., Сеннова Е.В., Сумароков С.В. Математическое моделирование и оптимизация систем тепло-, водо-, нефте- и газоснабжения. Новосибирск: Наука, 1992. 407 с.
46. Меренков А.П., Хасилев В.Я. Теория гидравлических цепей. М., Наука, 1985. 279 с.
47. Митичкин С.К., Сарданашвили С.А., Швечков В.А. Применение сетевых и распределённых компьютерных тренажёрных комплексов для решения задач подготовки диспетчерского персонала в подразделениях газотранспортных и газодобывающих предприятий // Математическое

моделирование трубопроводных систем энергетики. Иркутск. - ИСЭМ СО РАН. 2010. С. 534.

48. Митичкин С.К., Сарданашвили С.А. Компьютерные тренажёры производственно-диспетчерской службы газотранспортного предприятия: возможности и перспективы // Коллективная монография – Трубопроводные системы энергетики: математическое моделирование и оптимизация. Наука, Новосибирск. 2010. С. 419.
49. Наумец А.Е., Лебедев В.Г., Григорьев Л.И. Интеллектуальный анализ данных в системах поддержки принятия диспетчерских решений (СППДР) // Сб. Тез. международной научно-технической конференции ДИСКОМ 2012. 2012.
50. Новицкий Н.Н., Сеннова Е.В., Сухарев М.Г. Гидравлические цепи. Развитие теории и приложения. Новосибирск: Наука, 2000. 273 с.
51. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. Спб., Питер, 2000. 672 с.
52. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. Учебник. Спб., Питер, 2001. 539 с.
53. Оре О. Теория графов. М., Наука, 1980. 336 с.
54. Ортега Д. Введение в параллельные и векторные методы решения линейных систем. М., Мир, 1991. 365 с.
55. Панкратов В.С., Вербило А.С. Автоматизированная система диспетчерского управления ГТС. М., ООО "ИРЦ Газпром", 2001.
56. Панкратов В.С., Герке В.Г., Сарданашвили С.А., Митичкин С.К. Комплекс моделирования и оптимизации режимов работы ГТС. М., ООО "ИРЦ Газпром", 2002. 56 с.
57. Панкратов В.С., Дубинский А.В., Сиперштейн Б.И. Информационно-вычислительные системы в диспетчерском управлении газопроводами. Л., Недра, 1988. 60 с.

58. Панкратов В.С., Никишин В.К., Вербило А.С. АРМ диспетчера газотранспортного объединения. М., ВНИИЭГазпром, 1990. 32 с.
59. Перельцваиг Ю.М., Исследование и разработка вычислительных методов оптимизации технологических режимов магистральных газопроводов. М. Диссертация на соискание учёной степени кандидата наук 1978.
60. Редкозубов С.А. Статистические методы прогнозирования в АСУ. М., 1981.
61. Робачевский А.М., Немнюгин С.А., Стестик О.Л. Операционная система UNIX. СПб., 2005. 656 с.
62. Сарданашвили С.А., Ваулина Е.В. Применение объектно-ориентированной технологии решения задач планирования режимов газодобывающего предприятия // Наука и технология углеводородов. 1999. № 2.
63. Сарданашвили С.А., Митичкин С.К., Егоров А.В. Оптимизация режимов транспорта газа по газотранспортным сетям // Газовая промышленность, сер. Экономика, организация и управление производством в газовой промышленности. 1991. № 2. С. 8-15.
64. Сарданашвили С.А., Митичкин С.К., Леонов Д.Г., Свистунов А.А., Швечков В.А. Архитектура программного комплекса моделирования, оптимизации и прогнозирования режимов трубопроводного транспорта газа "ОКМ РГУ-LT" // Сб. Тез. Международной конференции ДИСКОМ 2004. 2004. Т. 3. С. 25-34.
65. Сарданашвили С.А., Митичкин С.К. Оптимизация режимов транспорта газа по газотранспортным сетям // Газовая промышленность / Экономика, организация и управление производством в газовой промышленности. 1991. № 2.
66. Сарданашвили С.А. Идентификация параметров моделей, описывающих нестационарное течение газа методом чувствительности // Изв. АН СССР / Техническая кибернетика. 1971. № 6.
67. Сарданашвили С.А. Расчётные методы и алгоритмы (трубопроводный

транспорт газа). М.,ФГУП Изд-во "Нефть и газ" РГУ нефти и газа им. И.М. Губкина, 2005. 577 с.

68. Селезнёв В.Е., Алёшин В.В., Клишин Г.С. Методы и технологии численного моделирования газопроводных систем. М., Едиториал УРСС, 2002. 448 с.
69. Селезнёв В.Е., Клишин Г.С. Численный анализ и оптимизация газодинамических режимов транспорта природного газа. М., Едиториал URSS, 2003. 224 с.
70. Сидлер В.Г., Новицкий Н.Н. Идентификация трубопроводных систем как гидравлических цепей с переменными параметрами // Изв. АН СССР / Энергетика и транспорт. 1984. № 4. С. 155-162.
71. Ставровский Е.Р., Сухарев М.Г. Универсальная программа расчёта газосборных сетей // Газовая промышленность. 1967. № 7.
72. Стратегия информатизации ОАО "Газпром". Приложение к постановлению правления ОАО "Газпром". 2008.
73. Страуструп Б. Язык программирования С++. Специальное издание. М., ООО "Бином-Персс", 2005. 1104 с.
74. Сухарев М.Г., Карасевич А.М. Технологический расчёт и обеспечение надёжности газо- и нефтепроводов. М., изд. "Нефть и газ", 2000. 271 с.
75. Сухарев М.Г., Ставровский Е.Р., Брянских В.Е. Оптимальное развитие систем газоснабжения. М., Недра, 1981. 294 с.
76. Сухарев М.Г., Ставровский Е.Р. Оптимизация систем транспорта газа. М., идз-во "Недра", 1971. 277 рр.
77. Сухарев М.Г., Ставровский Е.Р. Расчёт систем транспорта газа с помощью вычислительных машин. М., идз-во "Недра", 1971. 208 с.
78. Сухарев М.Г., Тимохов А.В., Фёдоров В.В. Курс методов оптимизации. М., Наука, 1986. 328 с.
79. Сухарев М.Г. О выборе метода при расчёте на ЭВМ течений по сетям //

Кибернетика. 1969. № 6.

80. Сухарев М.Г. Об одном методе расчёта газосборных сетей на вычислительных машинах // Нефть и газ. 1965. № 6.
81. Таненбаум Э., ван Стеен М. Распределённые системы. Принципы и парадигмы. Спб., Питер, 2003. 880 с.
82. Таненбаум Э. Компьютерные сети. СПб: Питер, 2007. 992 с.
83. Таненбаум Э. Современные операционные системы. 3-е издание. Спб., Питер, 2011. 1120 с.
84. Трахтенгерц Э.А., Стёпин Ю.П., Андреев А.Ф. Компьютерные методы поддержки принятия управленческих решений в нефтегазовой промышленности. М., Синтег, 2005. 592 с.
85. Форд Л., Фалкерсон Д. Потоки в сетях: пер. с англ. М., Наука, 1966. 276 с.
86. Хампель Ф., Рончетти Э., Рассеу П., Штаэль В. Робастность в статистике. Подход на основе функций влияния. М., Мир, 1989. 512 с.
87. Хьюбер П. Робастность в статистике. М., Мир, 1984. 340 с.
88. Хьюз К., Хьюз Т. Параллельное и распределённое программирование с использованием C++. М., Издательский дом "Вильямс", 2004. 672 с.
89. Чарный И.А. Неустановившееся движение реальной жидкости в трубах (2-е издание). М., Недра, 1975. 296 с.
90. Чарный И.А. Основы газовой динамики. М., Наука, 1961. 200 с.
91. Швечков В.А., Леонов Д.Г. Построение многопользовательского сетевого программного комплекса для решения задач диспетчерского управления // Сб. Тез. докладов Международной конференции ДИСКОМ 2004. 2004. С. 43-44.
92. Швечков В.А., Сарданашвили С.А. Сервис-ориентированная архитектура как инструмент интеграции информационного обеспечения в гетерогенной распределённой АСДУ ЕСГ России // Автоматизация в промышленности.

2007. № 5. С. 20-24.

93. Швечков В.А., Автоматизация диспетчерского управления в газотранспортной отрасли на основе технологий параллельных и распределённых вычислений. РГУ нефти и газа им. И.М. Губкина Диссертация на соискание учёной степени кандидата технических наук 2007.
94. Швечков В.А. Технологии параллельных вычислений для решения расчётных задач диспетчерского управления транспортом газа // Автоматизация в промышленности. 2006. № 7. С. 43-46.
95. Шмуллер Д. Освой самостоятельно UML за 24 часа. М., Издательский дом "Вильямс", 2002. 352 с.
96. Юфин В.А. Трубопроводный транспорт нефти и газа. М., Недра, 1978. 407 с.
97. Armstrong J., Virding R., Wikstrom C., and Williams M. Concurrent programming in Erlang (2nd Edition). Vol 2. Prentice Hall, 1996. 358 pp.
98. Armstrong J. Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf, 2007. 536 pp.
99. Barry D. Web Services, Service-Oriented Architectures, and Cloud Computing, Second Edition: The Savvy Manager's Guide (The Savvy Manager's Guides). Morgan Kaufmann, 2013. 248 pp.
100. Beck K. Test Driven Development: By Example. Addison-Wesley Professional, 2002. 240 pp.
101. Bell N., Garland M. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. 2012. Version 0.3.0.
102. Borkar S., Chien A.A., "The future of microprocessors," Communications of the ACM, Vol. 54, No. 5, 2011. pp. 67-77.
103. Buyya R., Cortes T., and Jin H., "Single system image," International Journal of High Performance Computing Applications, Vol. 15, No. 2, 2001. pp. 124-135.

104. Cade M., Sheil H. Sun Certified Enterprise Architect for Java EE Study Guide (2nd Edition). Prentice Hall, 2010. 216 pp.
105. Campbell C., Miller A. Parallel Programming with Microsoft Visual C++: Design Patterns for Decomposition and Coordination on Multicore Architectures (Patterns & Practices). Microsoft Press, 2011. 208 c.
106. Cerami E. Web Services Essentials. O'Reilly Media, 2002. 308 pp.
107. Cibaro P., Claeys K., Cozzolino F., and Garbner J. Professional WCF 4: Windows Communication Foundation with.NET 4. Wrox, 2010. 480 pp.
108. Core Math Library (ACML) [Электронный ресурс] // AMD Developer Central: [сайт]. URL: http://developer.amd.com/tools/cpu-development/amd-core-math-library-acml/ (дата обращения: 17.02.2013).
109. CUDA Toolkit Documentation [Электронный ресурс] // NVIDIA Developer Zone: [сайт]. URL: http://docs.nvidia.com/cuda/index.html (дата обращения: 17.02.2013).
110. Demming R., Duffy D. Introduction to the Boost C++ Libraries; Volume I - Foundations. Datasim Education BV, 2010. 310 pp.
111. Demming R., Duffy D. Introduction to the Boost C++ Libraries; Volume II - Advanced Libraries. Datasim Education BV, 2012. 356 pp.
112. Feathers M. Working effectively with legacy code. Prentice Hall, 2004. 434 pp.
113. Flynn M.J., Some computer organizations and their effectiveness. Computers, IEEE Transactions on, Vol. 100, No. 9, 1972. pp. 948-960.
114. Forum M.P.I. MPI: A Message-Passing Interface Standard, Version 3.0. High Performance Computing Center Stuttgart, 2012. 852 pp.
115. Fowler M., Beck K., Brant J., Opdyke W., and Roberts D. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999. 464 pp.
116. Francesco Cesarini S.T. Erlang programming. O'Reilly Media, 2009. 498 pp.

117. Gamma E., Helm R., Johnson R., and Vlissides J. Design patterns: elements of reusable object-oriented software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

118. Gaster B., Howes L., Kaeli D., Mistry P., and Schaa D. Heterogeneous Computing with OpenCL, Second Edition: Revised OpenCL 1.2 Edition. Morgan Kaufmann, 2012. 308 pp.

119. Gregory K., Miller A. C++ AMP: Accelerated Massive Parallelism with Microsoft Visual C++. Microsoft Press, 2012. 358 pp.

120. Hebert F. Learn You Some Erlang for Great Good! A Beginner's Guide. No Starch Press, 2013. 624 pp.

121. Henning M., Spruiell M., "Distributed programming with ice," ZeroC Inc. Revision, Vol. 3, 2003.

122. Henning M., Vinoski S. Advanced CORBA Programming with C++. Addison-Wesley Professional, 1999. 1120 pp.

123. Henning M., "A new approach to object-oriented middleware," Internet Computing, IEEE, Vol. 8, No. 1, 2004. pp. 66-75.

124. Henning M., "Massively multiplayer middleware," Queue, Vol. 1, No. 10, 2004. P. 38.

125. Henning M., "The rise and fall of CORBA," Queue, Vol. 4, No. 5, 2006. pp. 28-34.

126. Hewitt C., Bishop P., and Steiger R. A universal modular actor formalism for artificial intelligence // Proceedings of the 3rd international joint conference on Artificial intelligence. 1973. pp. 235-245.

127. Hewitt C, "Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot," DTIC Document, Tech. rep. 1972.

128. Hewitt C., "Viewing control structures as patterns of passing messages," Artificial intelligence, Vol. 8, No. 3, 1977. pp. 323-364.

129. Hoare C.A.R. Communicating Sequential Processes. Prentice Hall, 1985. 256 pp.
130. Hoberock J., Bell N. Thrust: A Parallel Template Library. 2010. Version 1.3.0.
131. Intel Math Kernel Library (Intel MKL) 11.0 [Электронный ресурс] // Intel Developers Zone: [сайт]. URL: http://software.intel.com/en-us/intel-mkl (дата обращения: 17.02.2013).
132. Jain P., Schmidt D.C. Service configurator: A pattern for dynamic configuration of services // Proceedings of the Conference. 1997. Vol. 3.
133. Johnson M. Job-Scheduling Tools: What you Need to Know For It Operations Management. Tebbo, 2011. 130 pp.
134. Karniadakis G., Kirby R. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation. Cambridge University Press, 2003. 630 pp.
135. Kerievsky J. Refactoring to Patterns. Addison-Wesley Professional, 2004. 400 pp.
136. Kirk D.B., Wen-mei W.H. Programming massively parallel processors: a hands-on approach. Morgan Kaufmann, 2010. 258 pp.
137. Lakos J. Large-Scale C++ Software Design. Addison-Wesley Professional, 1996. 896 pp.
138. Martin R.C., "Design principles and design patterns," Object Mentor, 2000.
139. Martin R.C., "The open-closed principle," More C++ gems, 1996. pp. 97-112.
140. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008. 464 pp.
141. McLean S., Naftel J., and Williams K. Microsoft.NET Remoting (Pro-Developer). Microsoft Press, 2002. 336 pp.
142. Meyers S. Effetive C++. Third Edition. 55 Specific Ways to Improve Your Programs and Designs. Addison-Wesley, 2012. 297 pp.
143. Moore G.E., others, "Cramming more components onto integrated circuits,"

Proceedings of the IEEE, Vol. 86, No. 1, 1998. pp. 82-85.

144. Papazoglou M. Web Services: Principles and Technology. Prentice Hall, 2007. 784 pp.
145. Pathak N. Pro WCF 4: Practical Microsoft SOA Implementation. Apress, 2011. 472 pp.
146. Quinn M. Parallel Programming in C with MPI and OpenMP. McGraw-Hill Education, 2008. 480 pp.
147. Rauber T., Runger G. Parallel programming: For multicore and cluster systems. Springer, 2010.
148. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. O'Reilly Media, 2007. 336 pp.
149. Sanders J., Kandrot E. CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional, 2010. 290 pp.
150. Schaling B. The Boost C++ Libraries. XML Press, 2011. 262 pp.
151. Shalloway A., Trott J.R. Design Patterns Explained. A New Perspective on a Object-Oriented Design. Addison-Wesley, 2005. 429 pp.
152. Shoham Y., "Agent-oriented programming," Artificial intelligence, Vol. 60, No. 1, 1993. pp. 51-92.
153. Stevens P. Using UML: Software Engineering with Objects and Components (2nd Edition). Addison-Wesley Professional, 2006. 272 pp.
154. Sutter H., Larus J., "Software and the Concurrency Revolution," Queue, Vol. 3, No. 7, September 2005. pp. 54-62.
155. Sutter H., "The free lunch is over: A fundamental turn toward concurrency in software," Dr. Dobb’s Journal, Vol. 30, No. 3, 2005. pp. 202-210.
156. Sutter H. Welcome to the Jungle [Электронный ресурс] // Sutter's Mill. Herb Sutter on Software and Concurrency: [сайт]. URL: http://herbsutter.com/welcome-to-the-jungle/ (дата обращения: 17.02.2013).

157. Szyperski C. Component Software: Beyond Object-Oriented Programming (2nd.Edition). Addison-Wesley Professional, 2011. 624 pp.
158. Thai T. Learning DCOM. O'Reilly Media, 1999. 504 pp.
159. The Internet Communication Engine (Ice) [Электронный ресурс] // Welcome to ZeroC, the Home of Ice: [сайт]. URL: http://zeroc.com/ice.html (дата обращения: 17.02.2013).
160. Wienke S., Springer P., Terboven C., and an Mey D., "OpenACC—First Experiences with Real-World Applications," Euro-Par 2012 Parallel Processing, 2012. pp. 859-870.

Приложение А. Программный код вычислительных сервисов

Листинг 1. Абстрактный интерфейс менеджера параллельного моделирования объектов

```
/** Абстрактный интерфейс менеджера параллельного моделирования объектов.*/
#include <vector>             // Вектор используется для хранения объектов
#include "work_params.h"      // Определение типа рабочих параметров
#include "calculated_params.h"// Определение типа расчётных параметров

class ParallelManagerI {
public:
  // Установить рабочие параметры для вектора управляемых объектов
  virtual void SetWorkParams(std::vector<WorkParams> const &work_params) = 0;
  virtual void CalculateAll() = 0; // Моделировать все управляемые объекты
  virtual void GetCalculatedParams(// Вернуть вектор расчётных значений
      std::vector<CalculatedParams> *calculated_params) = 0;
};
```

Листинг 2. Базовый менеджер параллельного моделирования трубопроводов

```
/** Реализация базового менеджера моделирования труб,
не использующего параллельные вычисления.*/
#include "parallel_manager_pipe_singlecore.h" // заголовочный файл
#include "model_pipe_sequential.h"            // модель трубы
#include <vector>                             // вектор для хранения моделей

// Принять вектор объектов под управление
void ParallelManagerPipeSingleCore::
    TakeUnderControl(std::vector<PassportPipe> const &passports) {
  for(auto p = passports.begin(); p != passports.end(); ++p) {
    PassportPipe passport = *p;
    ModelPipeSequential model(&passport);
    models_.push_back(model);
  }
}
// Установить вектор рабочих параметров
void ParallelManagerPipeSingleCore::
    SetWorkParams(std::vector<WorkParams> const &work_params) {
  auto wp = work_params.begin();
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    Gas gas_in;
    gas_in.composition.density_std_cond = wp->den_sc_in();
    gas_in.composition.co2 = wp->co2_in();
    gas_in.composition.n2 = wp->n2_in();
    gas_in.work_parameters.p = wp->p_in();
    gas_in.work_parameters.t = wp->t_in();
    m->set_gas_in(&gas_in);

    Gas gas_out;
    gas_out.work_parameters.p = wp->p_out();
    m->set_gas_out(&gas_out);
    ++wp;
  }
}
// Выполнить моделирование всех управляемых объектов
void ParallelManagerPipeSingleCore::CalculateAll() {
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    m->Count();
  }
}
// Вернуть вектор расчётных параметров
void ParallelManagerPipeSingleCore::
    GetCalculatedParams(std::vector<CalculatedParams> *calculated_params){
  calculated_params->erase(calculated_params->begin(), calculated_params->end());
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    CalculatedParams cp;
    cp.set_q         ( m->q() );
    cp.set_dq_dp_in  ( m->dq_dp_in() );
    cp.set_dq_dp_out ( m->dq_dp_out() );
    cp.set_t_out     ( m->gas_out().work_parameters.t );
    calculated_params->push_back(cp);
  }
}
```

Листинг 3. Менеджер параллельного моделирования трубопроводов с использованием *OpenMP*

```
/** Реализация менеджера параллельного моделирования труб с использованием OpenMP
*/
#include "parallel_manager_pipe_openmp.h" // заголовочный файл
#include "model_pipe_sequential.h"            // модель трубы
#include <vector>                             // вектор для хранения моделей

// Принять вектор объектов под управление
void ParallelManagerPipeSingleCore::
    TakeUnderControl(std::vector<PassportPipe> const &passports) {
  for(auto p = passports.begin(); p != passports.end(); ++p) {
    PassportPipe passport = *p;
    ModelPipeSequential model(&passport);
    models_.push_back(model);
  }
}

// Установить вектор рабочих параметров
void ParallelManagerPipeSingleCore::
    SetWorkParams(std::vector<WorkParams> const &work_params) {
  auto wp = work_params.begin();
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    Gas gas_in;
    gas_in.composition.density_std_cond = wp->den_sc_in();
    gas_in.composition.co2 = wp->co2_in();
    gas_in.composition.n2 = wp->n2_in();
    gas_in.work_parameters.p = wp->p_in();
    gas_in.work_parameters.t = wp->t_in();
    m->set_gas_in(&gas_in);

    Gas gas_out;
    gas_out.work_parameters.p = wp->p_out();
    m->set_gas_out(&gas_out);
    ++wp;
  }
}

// Выполнить моделирование всех управляемых объектов
void ParallelManagerPipeSingleCore::CalculateAll() {
  #pragma omp parallel for
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    m->Count();
  }
}

// Вернуть вектор расчётных параметров
void ParallelManagerPipeSingleCore::
    GetCalculatedParams(std::vector<CalculatedParams> *calculated_params){
  calculated_params->erase(calculated_params->begin(), calculated_params->end());
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    CalculatedParams cp;
    cp.set_q         ( m->q() );
```

```
    cp.set_dq_dp_in  ( m->dq_dp_in() );
    cp.set_dq_dp_out ( m->dq_dp_out() );
    cp.set_t_out     ( m->gas_out().work_parameters.t );
    calculated_params->push_back(cp);
  }
}
```

Листинг 4. Обобщённый менеджер параллельного моделирования с использованием *OpenMP*

```
/** Обобщённый менеджер параллельного моделирования с использованием OpenMP*/
#include <vector>                    // Хранение вектора моделей
#include "work_params.h"             // Определение типа рабочих параметров
#include "calculated_params.h"       // Определение типа расчётных параметров

template <class Model, class Passport>
class GeneralParallelManager : public ParallelManagerPipeI{
public:
 // Принять под управление вектор объектов, допускающих создание по паспорту
 virtual void TakeUnderControl(std::vector<Passport> const &passports) {
   for(auto p = passports.begin(); p != passports.end(); ++p) {
      Passport passport = *p;
      Model model(&passport);
      models_.push_back(model);
    }
 }
 virtual void SetWorkParams(std::vector<WorkParams> const &work_params) {
   auto wp = work_params.begin();
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    Gas gas_in;
    gas_in.composition.density_std_cond = wp->den_sc_in();
    gas_in.composition.co2 = wp->co2_in();
    gas_in.composition.n2 = wp->n2_in();
    gas_in.work_parameters.p = wp->p_in();
    gas_in.work_parameters.t = wp->t_in();
    m->set_gas_in(&gas_in);

    Gas gas_out;
    gas_out.work_parameters.p = wp->p_out();
    m->set_gas_out(&gas_out);
    ++wp;
  }

 };
 // Моделировать вектор объектов, поддерживающих метод  Count
 virtual void CalculateAll() {
   #pragma omp parallel for
   for(int i = 0; i < models_.size(); ++i) {
     models_[i].Count();
   }
};
 virtual void GetCalculatedParams(std::vector<CalculatedParams>
*calculated_params) {
   calculated_params->erase(calculated_params->begin(), calculated_params->end());
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    CalculatedParams cp;
    cp.set_q         ( m->q() );
    cp.set_dq_dp_in  ( m->dq_dp_in() );
    cp.set_dq_dp_out ( m->dq_dp_out() );
    cp.set_t_out     ( m->gas_out().work_parameters.t );
    calculated_params->push_back(cp);
```

```
  }
 };
private:
  std::vector<Model> models_; // Хранение вектора объектов
};
```

Листинг 5. Обобщённый менеджер параллельного моделирования с использованием библиотеки *CUDA Thrust*

```
/** Обобщённый менеджер параллельного моделирования с использованием CUDA Thrust*/
#include <vector>                    // Хранение вектора моделей
#include "work_params.h"             // Определение типа рабочих параметров
#include "calculated_params.h"       // Определение типа расчётных параметров

template <class Model>
CountFunctor { // Шаблонный функтор для моделирования объекта
public:
  void operator ()(Model _m) {_m.Count();} };
}

template <class Model, class Passport>
class GeneralParallelManager : public ParallelManagerPipeI{
public:
 // Принять под управление вектор объектов, допускающих создание по паспорту
 virtual void TakeUnderControl(std::vector<Passport> const &passports) {
   for(auto p = passports.begin(); p != passports.end(); ++p) {
      Passport passport = *p;
      Model model(&passport);
      models_.push_back(model);
    }
 }

 virtual void SetWorkParams(std::vector<WorkParams> const &work_params) {
   auto wp = work_params.begin();
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    Gas gas_in;
    gas_in.composition.density_std_cond = wp->den_sc_in();
    gas_in.composition.co2 = wp->co2_in();
    gas_in.composition.n2 = wp->n2_in();
    gas_in.work_parameters.p = wp->p_in();
    gas_in.work_parameters.t = wp->t_in();
    m->set_gas_in(&gas_in);

    Gas gas_out;
    gas_out.work_parameters.p = wp->p_out();
    m->set_gas_out(&gas_out);
    ++wp;
  }
 };

 // Моделировать вектор объектов, поддерживающих метод  Count
 virtual void CalculateAll() {
   models_device_ = models_host_; // Отправка данных в память видеокарты
   // Параллельное моделирование объектов на CUDA-устройстве
   thrust::for_each(models_device_.begin(), models_device.end(), CountFunctor() );
   models_host = models_device_; // Получение результатов из памяти видеокарты
 };

 virtual void GetCalculatedParams(std::vector<CalculatedParams>
*calculated_params) {
```

```
   calculated_params->erase(calculated_params->begin(), calculated_params->end());
  for(auto m = models_.begin(); m != models_.end(); ++m) {
    CalculatedParams cp;
    cp.set_q         ( m->q() );
    cp.set_dq_dp_in  ( m->dq_dp_in() );
    cp.set_dq_dp_out ( m->dq_dp_out() );
    cp.set_t_out     ( m->gas_out().work_parameters.t );
    calculated_params->push_back(cp);
  }
 };
private:
  thrust::host_vector<Model> models_;         // Хранение вектора управляемых
объектов
  thrust::device_vector<Model> models_device_;// Вектор объектов в памяти CUDA
};
```

Листинг 6. Оптимизированный менеджер параллельного моделирования трубопроводов на *CUDA C*

```
/** \file parallel_manager_pipe_cuda.cpp
Реализация менеджера параллельного моделирования труб, на основе CUDA C.*/
#include "parallel_manager_pipe_cuda.cuh"

#include <vector>
#include "calculated_params.h"

// Для векторных типов CUDA double2, double4 и т.д.
#include <vector_types.h>

#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

#include "model_pipe_sequential_functions_cuda.cuh"
#include "gas_count_functions_cuda.cuh"

#ifdef CUPRINTF
  #include "cuprintf.cu"
#endif
__global__
void FindQResultCudaKernel(
        int size,
        double* den_sc, double* co2, double* n2,
        double2* p_and_t, double* p_target,
        double* length,
        double2* d_in_out,
        double4* hydr_rough_env_exch,
        double* q_result,
    double* t_result,
    double* dq_dp_in_result,
    double* dq_dp_out_result) {
  #ifdef CUPRINTF
    cuPrintfRestrict(0, 0);
    cuPrintf("FindQResultCudaKernel-----------------------------------\n");
  #endif
      int index = threadIdx.x + blockIdx.x * blockDim.x;
  while(index < size)  {
            // Загружаем данные
            // Состав газа
    #ifdef CUPRINTF
      cuPrintf("Loading data from memory--------------------------------\n");
    #endif
            double den_sc_ = den_sc[index];
            double co2_ = co2[index];
            double n2_ = n2[index];
            // Давление и температура на входе
            double2 p_and_t_ = p_and_t[index];
            // Пасспотные параметры трубы
            double length_ = length[index];
            double2 d_in_out_ = d_in_out[index];
            double4 hydr_rough_env_exch_ = hydr_rough_env_exch[index];
```

```
        double p_target_ = p_target[index];
#ifdef CUPRINTF
  cuPrintf("Pipe Passport parameters:-------------------------------\n");
  cuPrintf("length = %f\n", length_);
  cuPrintf("d_in   = %f\n", d_in_out_.x);
  cuPrintf("d_out  = %f\n", d_in_out_.y);
  cuPrintf("hydr   = %f\n", hydr_rough_env_exch_.x);
  cuPrintf("rough  = %f\n", hydr_rough_env_exch_.y);
  cuPrintf("env    = %f\n", hydr_rough_env_exch_.z);
  cuPrintf("exch   = %f\n", hydr_rough_env_exch_.w);
  cuPrintf("\nWork parameters:-------------------------------------\n");
  cuPrintf("p      = %f\n", p_and_t_.x);
  cuPrintf("p_out  = %f\n", p_target_);
  cuPrintf("t      = %f\n", p_and_t_.y);
  cuPrintf("den_sc = %f\n", den_sc_);
  cuPrintf("co2    = %f\n", co2_);
  cuPrintf("n2     = %f\n", n2_);
#endif
  // Вычисляем базовые свойства газового потока
  double r_sc_ = FindRStandartConditionsCuda(den_sc_);
  double t_pc_ = FindTPseudoCriticalCuda(den_sc_, co2_, n2_);
  double p_pc_ = FindPPseudoCriticalCuda(den_sc_, co2_, n2_);
#ifdef CUPRINTF
  cuPrintf("\nCalculating base gas props:---------------------------\n");
  cuPrintf("r_sc = %f\n", r_sc_);
  cuPrintf("t_pc = %f\n", t_pc_);
  cuPrintf("p_pc = %f\n", p_pc_);
#endif
        double q_out = 0;
        double p_out = 0;
        double t_out = 0;

int segments = static_cast<int>(length_)/10;
if(segments < 1) { segments = 1; }
#ifdef CUPRINTF
  cuPrintf("\nCalculating # of segments:----------------------------\n");
  cuPrintf("segments = %d\n", segments);
#endif

  FindSequentialQCuda(
    p_target_, // давление, которое должно получиться в конце
    p_and_t_.x, p_and_t_.y,  // рабочие параметры газового потока на входе
    den_sc_, co2_, n2_, // состав газа
    d_in_out_.x, d_in_out_.y,
    hydr_rough_env_exch_.y, //rough
    hydr_rough_env_exch_.x, //hydr
    hydr_rough_env_exch_.z, //env
    hydr_rough_env_exch_.w, //heat_exch
    length_/segments, segments, // длина сегмента и кол-во сегментов
    &t_out, &q_out); // out - параметры, значения на выходе )

q_result[index] = q_out;
t_result[index] = t_out;
// Рассчитываем производные-----------------------------------------
// Следим, чтобы шаг дифференцирования был не больше (Рвых - Рвх) / 2.
```

```
double eps = (p_and_t_.x - p_target_)/2; // dP/2.
if(eps > 0.000005) { eps = 0.000005; } // Значение по умолчанию - 5 Па.
double t_dummy; // t, возвращаемое при расчёте производных не нужно.
double q_p_in_plus_eps;
FindSequentialQCuda(
  p_target_ , // давление, которое должно получиться в конце
  p_and_t_.x + eps, p_and_t_.y,  // рабочие параметры газового потока на входе
  den_sc_, co2_, n2_, // состав газа
  d_in_out_.x, d_in_out_.y,
  hydr_rough_env_exch_.y, //rough
  hydr_rough_env_exch_.x, //hydr
  hydr_rough_env_exch_.z, //env
  hydr_rough_env_exch_.w, //heat_exch
  length_/segments, segments, // длина сегмента и кол-во сегментов
  &t_dummy, &q_p_in_plus_eps); // out - параметры, значения на выходе )
double q_p_in_minus_eps;
FindSequentialQCuda(
  p_target_, // давление, которое должно получиться в конце
  p_and_t_.x - eps, p_and_t_.y,  // рабочие параметры газового потока на входе
  den_sc_, co2_, n2_, // состав газа
  d_in_out_.x, d_in_out_.y,
  hydr_rough_env_exch_.y, //rough
  hydr_rough_env_exch_.x, //hydr
  hydr_rough_env_exch_.z, //env
  hydr_rough_env_exch_.w, //heat_exch
  length_/segments, segments, // длина сегмента и кол-во сегментов
  &t_dummy, &q_p_in_minus_eps); // out - параметры, значения на выходе )
dq_dp_in_result[index] = (q_p_in_plus_eps - q_p_in_minus_eps) / (2*eps);

double q_p_out_plus_eps;
FindSequentialQCuda(
  p_target_ + eps, // давление, которое должно получиться в конце
  p_and_t_.x, p_and_t_.y,  // рабочие параметры газового потока на входе
  den_sc_, co2_, n2_, // состав газа
  d_in_out_.x, d_in_out_.y,
  hydr_rough_env_exch_.y, //rough
  hydr_rough_env_exch_.x, //hydr
  hydr_rough_env_exch_.z, //env
  hydr_rough_env_exch_.w, //heat_exch
  length_/segments, segments, // длина сегмента и кол-во сегментов
  &t_dummy, &q_p_out_plus_eps); // out - параметры, значения на выходе )
double q_p_out_minus_eps;
FindSequentialQCuda(
  p_target_ - eps, // давление, которое должно получиться в конце
  p_and_t_.x, p_and_t_.y,  // рабочие параметры газового потока на входе
  den_sc_, co2_, n2_, // состав газа
  d_in_out_.x, d_in_out_.y,
  hydr_rough_env_exch_.y, //rough
  hydr_rough_env_exch_.x, //hydr
  hydr_rough_env_exch_.z, //env
  hydr_rough_env_exch_.w, //heat_exch
  length_/segments, segments, // длина сегмента и кол-во сегментов
  &t_dummy, &q_p_out_minus_eps); // out - параметры, значения на выходе )
dq_dp_out_result[index] = (q_p_out_plus_eps - q_p_out_minus_eps) / (2*eps);
```

```
    #ifdef CUPRINTF
      cuPrintf("\nCalculating results:-----------------------------------\n");
      cuPrintf("q                   = %f\n", q_result[index]);
      cuPrintf("t_out               = %f\n", t_result[index]);
      cuPrintf("dq_dp_in            = %f\n", dq_dp_in_result[index]);
      cuPrintf("--q_p_in_plus_eps     = %f\n", q_p_in_plus_eps);
      cuPrintf("--q_p_in_minus_eps    = %f\n", q_p_in_minus_eps);
      cuPrintf("dq_dp_out           = %f\n", dq_dp_out_result[index]);
      cuPrintf("--q_p_out_plus_eps    = %f\n", q_p_out_plus_eps);
      cuPrintf("--q_p_out_minus_eps   = %f\n", q_p_out_minus_eps);
      cuPrintf("eps                 = %f\n", eps);
    #endif

          index += gridDim.x * blockDim.x;
      } // end while (index < size)
}

ParallelManagerPipeCUDA::ParallelManagerPipeCUDA() {
  number_of_pipes = 0;
}
void ParallelManagerPipeCUDA::
  TakeUnderControl(std::vector<PassportPipe> const &passports) {
    number_of_pipes = passports.size();
    AllocateMemoryOnHost();
    SavePassportsAsStructureOfArrays(passports);
    AllocateMemoryOnDevice();
    SendPassportsToDevice();
}
void ParallelManagerPipeCUDA::
    SetWorkParams(std::vector<WorkParams> const &work_params) {
  SaveWorkParamsToStructureOfArrays(work_params);
  SendWorkParamsToDevice();
}
void ParallelManagerPipeCUDA::CalculateAll() {
  #ifdef CUPRINTF
    cudaPrintfInit();
  #endif
  FindQResultCudaKernel<<<512, 64, 0>>>(
                 number_of_pipes,
                 den_sc_dev_, co2_dev_, n2_dev_,
                 p_in_and_t_in_dev_, p_target_dev_,
                 length_dev_,
                 d_in_out_dev_,
                 hydr_rough_env_exch_dev_,
                 q_result_dev_,
      t_result_dev_,
      dq_dp_in_result_dev_,
      dq_dp_out_result_dev_);
  #ifdef CUPRINTF
    cudaPrintfDisplay(stdout, true);
    cudaPrintfEnd();
  #endif
  SendCalculatedParamsToHost();
}
```

```
void ParallelManagerPipeCUDA::
    GetCalculatedParams(std::vector<CalculatedParams> *calculated_params){
  calculated_params->erase(
      calculated_params->begin(), calculated_params->end() );
  calculated_params->reserve(number_of_pipes);

  auto t = t_result_host.begin();
  auto dq_dp_in = dq_dp_in_result_host.begin();
  auto dq_dp_out = dq_dp_out_result_host.begin();
  for(auto q = q_result_host.begin(); q != q_result_host.end(); ++q) {
    CalculatedParams cp;
    cp.set_q(*q);
    cp.set_dq_dp_in(*dq_dp_in);
    cp.set_dq_dp_out(*dq_dp_out);
    cp.set_t_out(*t);
    calculated_params->push_back(cp);
    ++dq_dp_in;
    ++dq_dp_out;
    ++t;
  }
}

void ParallelManagerPipeCUDA::AllocateMemoryOnHost() {
  // Пасспортные параметры.
  length_host               .reserve(number_of_pipes);
  d_in_out_host             .reserve(number_of_pipes);
  hydr_rough_env_exch_host  .reserve(number_of_pipes);
  // Рабочие параметры.
  den_sc_host               .reserve(number_of_pipes);
  co2_host                  .reserve(number_of_pipes);
  n2_host                   .reserve(number_of_pipes);
  p_in_and_t_in_host        .reserve(number_of_pipes);
  p_target_host             .reserve(number_of_pipes);
  // Рассчитанные параметры.
  q_result_host             .reserve(number_of_pipes);
  t_result_host             .reserve(number_of_pipes);
  dq_dp_in_result_host      .reserve(number_of_pipes);
  dq_dp_out_result_host     .reserve(number_of_pipes);
}
void ParallelManagerPipeCUDA::AllocateMemoryOnDevice() {
  // Пасспортные параметры.
  length_dev_vec              .resize(number_of_pipes);
  d_in_out_dev_vec            .resize(number_of_pipes);
  hydr_rough_env_exch_dev_vec .resize(number_of_pipes);
  // Рабочие параметры.
  p_in_and_t_in_dev_vec       .resize(number_of_pipes);
  p_target_dev_vec            .resize(number_of_pipes);
  den_sc_dev_vec              .resize(number_of_pipes);
  co2_dev_vec                 .resize(number_of_pipes);
  n2_dev_vec                  .resize(number_of_pipes);
  // Рассчитанные параметры.
  q_result_dev_vec            .resize(number_of_pipes);
  t_result_dev_vec            .resize(number_of_pipes);
  dq_dp_in_result_dev_vec     .resize(number_of_pipes);
  dq_dp_out_result_dev_vec    .resize(number_of_pipes);
```

```
  // Формируем указатели на память device.
  using thrust::raw_pointer_cast;
  den_sc_dev_             = raw_pointer_cast(&den_sc_dev_vec             [0]);
  co2_dev_                = raw_pointer_cast(&co2_dev_vec                [0]);
  n2_dev_                 = raw_pointer_cast(&n2_dev_vec                 [0]);
  p_in_and_t_in_dev_      = raw_pointer_cast(&p_in_and_t_in_dev_vec      [0]);
  p_target_dev_           = raw_pointer_cast(&p_target_dev_vec           [0]);
  length_dev_             = raw_pointer_cast(&length_dev_vec             [0]);
  d_in_out_dev_           = raw_pointer_cast(&d_in_out_dev_vec           [0]);
  hydr_rough_env_exch_dev_= raw_pointer_cast(&hydr_rough_env_exch_dev_vec [0]);
  q_result_dev_           = raw_pointer_cast(&q_result_dev_vec           [0]);
  t_result_dev_           = raw_pointer_cast(&t_result_dev_vec           [0]);
  dq_dp_in_result_dev_    = raw_pointer_cast(&dq_dp_in_result_dev_vec    [0]);
  dq_dp_out_result_dev_   = raw_pointer_cast(&dq_dp_out_result_dev_vec   [0]);
}
void ParallelManagerPipeCUDA::
    SavePassportsAsStructureOfArrays(
        std::vector<PassportPipe> const &passports) {
  for(auto p = passports.begin(); p != passports.end(); ++p) {
    double length = p->length_;
    double2 d_in_out = make_double2(p->d_inner_, p->d_outer_);
    double4 hydr_rough_env_exch =
        make_double4(
            p->hydraulic_efficiency_coeff_,
            p->roughness_coeff_,
            p->t_env_,
            p->heat_exchange_coeff_ );
    length_host.push_back(length);
    d_in_out_host.push_back(d_in_out);
    hydr_rough_env_exch_host.push_back(hydr_rough_env_exch);
  }
}
void ParallelManagerPipeCUDA::
    SaveWorkParamsToStructureOfArrays(
        std::vector<WorkParams> const &work_params){
  for(auto wp = work_params.begin(); wp != work_params.end(); ++wp) {
    den_sc_host       .push_back( wp->den_sc_in() );
    co2_host          .push_back( wp->co2_in()    );
    n2_host           .push_back( wp->n2_in()     );
    p_in_and_t_in_host.push_back(
                     make_double2( wp->p_in(),
                                   wp->t_in() )     );
    p_target_host      .push_back( wp->p_out()    );
  }
}

void ParallelManagerPipeCUDA::SendPassportsToDevice() {
  // Магия thrust. Присваивание dev_vec = host_vec отправляет вектор на Device.
  length_dev_vec              = length_host;
  d_in_out_dev_vec            = d_in_out_host;
  hydr_rough_env_exch_dev_vec = hydr_rough_env_exch_host;
}
void ParallelManagerPipeCUDA::SendWorkParamsToDevice() {
  den_sc_dev_vec        = den_sc_host;
```

```
  co2_dev_vec           = co2_host;
  n2_dev_vec            = n2_host;
  p_in_and_t_in_dev_vec = p_in_and_t_in_host;
  p_target_dev_vec      = p_target_host;
}
void ParallelManagerPipeCUDA::SendCalculatedParamsToHost() {
  q_result_host         = q_result_dev_vec;
  t_result_host         = t_result_dev_vec;
  dq_dp_in_result_host  = dq_dp_in_result_dev_vec;
  dq_dp_out_result_host = dq_dp_out_result_dev_vec;
}
```

Листинг 7. Абстрактный интерфейс решателя разреженных систем линейных алгебраических уравнений

```
/** Абстрактный интерфейс решателя СЛАУ SlaeSolverI. */
#pragma once
#include <vector>
class SlaeSolverI {
 public:
   virtual void Solve( // Решить систему уравнений
     std::vector<int> const &A_indexes,  // Вектор индексов значимых коэф-в
     std::vector<double> const &A_values,// Вектор значений значимых коэф-в
     std::vector<double> const &B,       // Вектор правой части
     std::vector<double> *X              // Решение – out-параметр
  ) = 0;
};
```

Листинг 8. Решатель разреженных систем уравнений с использованием библиотек *Intel MKL* и *CVM*

```
/** \file test_slae_solver_cvm.cpp
  Реализация SlaeSolverCVM.*/
#include "slae_solver_cvm.h" // Заколовочный файл
#include <vector>

#include "cvm.h" // Библиотека CVM для матричных операций – слинкована с Intel MKL
#include <math.h>

void SlaeSolverCVM::Solve(
    std::vector<int> const &A_indexes,
    std::vector<double> const &A_values,
    std::vector<double> const &B,
    std::vector<double> *X) {
  // Длина вектора B = Длина строки матрицы A.
  int size = B.size();
  cvm::srmatrix A(size);
  auto a_val = A_values.begin();
  for(auto a_index = A_indexes.begin(); a_index != A_indexes.end();
      ++a_index) {
    // Заполняем матрицу A CVM - нумерация с единицы.
    int row = *a_index / size;
    int col = *a_index - row*size;
    A(row + 1, col + 1) = *a_val;
    ++a_val;
  }
  cvm::rvector B_cvm(size);
  std::copy(B.begin(), B.end(), B_cvm.begin() );
  cvm::rvector X_cvm(size);
  X_cvm.solve(A, B_cvm);
  X->resize(size);
  std::copy(X_cvm.begin(), X_cvm.end(), X->begin() );
}
```

Листинг 9. Решатель разреженных систем уравнений с использованием библиотеки *CuSp*

```
/** \file slae_solver_cusp.cpp
  Реализация SlaeSolverCusp.*/
#include "slae_solver_cusp.cuh" // Заголовочный файл
#include <vector>

#include <cusp/coo_matrix.h>
#include <cusp/monitor.h>
#include <cusp/precond/smoothed_aggregation.h>
#include <cusp/precond/ainv.h>
#include <cusp/krylov/bicgstab.h>
#include <cusp/krylov/cg.h>
#include <cusp/krylov/gmres.h>

#include <cusp/csr_matrix.h>
#include <cusp/krylov/bicg.h>
#include <cusp/gallery/poisson.h>
#include <cusp/io/matrix_market.h>

template <typename Monitor>
void report_status(Monitor& monitor) // Отобразить статус работы сервиса
{
    if (monitor.converged())
    {
        std::cout << "  Solver converged to " << monitor.tolerance() << "
tolerance";
        std::cout << " after " << monitor.iteration_count() << " iterations";
        std::cout << " (" << monitor.residual_norm() << " final residual)" <<
std::endl;
    }
    else
    {
        std::cout << "  Solver reached iteration limit " <<
monitor.iteration_limit() << " before converging";
        std::cout << " to " << monitor.tolerance() << " tolerance ";
        std::cout << " (" << monitor.residual_norm() << " final residual)" <<
std::endl;
    }
}

void SlaeSolverCusp::Solve( // Решить СЛАУ
    std::vector<int> const &A_indexes,
    std::vector<double> const &A_values,
    std::vector<double> const &B,
    std::vector<double> *X) {

  int size = B.size();
  int non_zeros = A_indexes.size();

  cusp::coo_matrix<int, double, cusp::host_memory>   A_host(size, size,
non_zeros);
  for(int i = 0; i < A_indexes.size(); ++i) {
```

```
    int index = A_indexes[i];
    int row = index / size;
    int col = index - ( row*size );
    A_host.row_indices   [i] = row;
    A_host.column_indices[i] = col;
    A_host.values        [i] = A_values[i];
  }

  cusp::coo_matrix<int, double, cusp::device_memory> A_dev (size, size,
non_zeros);
  A_dev = A_host;

  cusp::array1d<double, cusp::host_memory> x_host(size, 0);
  cusp::array1d<double, cusp::host_memory> b_host( B.begin(), B.end() );
  cusp::array1d<double, cusp::device_memory> x_dev(size, 0);
  cusp::array1d<double, cusp::device_memory> b_dev;
  b_dev = b_host;

  // Установить критерий останова:
  // Максимальное количество итераций    = 500
  // Требуемая относительная точность    = 1e-6
  cusp::default_monitor<double> monitor(b_dev, 500, 1e-6);

  cusp::precond::diagonal<double, cusp::device_memory> M(A_dev);
  cusp::krylov::bicgstab(A_dev, x_dev, b_dev, monitor, M);

  x_host = x_dev;
  cusp::io::write_matrix_market_file(x_host, "x_Saratov50.mtx");

  X->resize(size);
  thrust::copy(x_host.begin(), x_host.end(), X->begin());

  // Отобразить статус работы
  report_status(monitor);
}
```