ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э.Баумана, ОАО "Концерн "Сириус".

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru E-mail: prin@novtex.ru

Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

УДК 004.4

В. В. Липаев, д-р техн. наук, проф., гл. науч. сотр.,
Институт системного программирования РАН, г. Москва
e-mail: vlip28@mail.ru

Развитие базовых стандартов программной инженерии

Изложены цели и методические основы стандартизации программной и системной инженерии. Представлены основные особенности стандарта программной инженерии ISO/IEC 12207:1995 — Процессы жизненного цикла программных средств, развитие и содержание нового модернизированного стандарта ISO/IEC 12207:2008 и рекомендации по его применению. Отмечена сущность его взаимосвязи со стандартом системной инженерии ISO/IEC 15288:2005.

Ключевые слова: *стандартизация, жизненный цикл, программная инженерия, комплекс программ, системная инженерия, производство программных продуктов, комплексирование компонентов*

В настоящее время программную инженерию поддерживают ***около 50-ти международных стандартов***, которые совершенствуются и развиваются. ***Стандарты*** формализуют устоявшуюся терминологию, устанавливают общую структуру процессов жизненного цикла программных комплексов, на которую целесообразно ориентироваться в программной индустрии. Они определяют процессы, виды деятельности и задачи, которые используются при поставке, производстве, применении по назначению, сопровождении и прекращении применения программных продуктов. Стандарты устанавливают рекомендации и нормативы к действиям, которые могут использоваться при определении, управлении и совершенствовании процессов, сопровождающих жизненный цикл программных комплексов. Они предназначены для представления определенной совокупности действий, облегчающих отношения между потребителями, поставщиками и другими заинтересованными лицами в течение жизненного цикла программных продуктов. Широкий диапазон применения продуктов в ***программной индустрии*** привел к тому, что они и процессы их разработки зачастую рассматриваются как компоненты и ***составная часть систем*** и их создания.

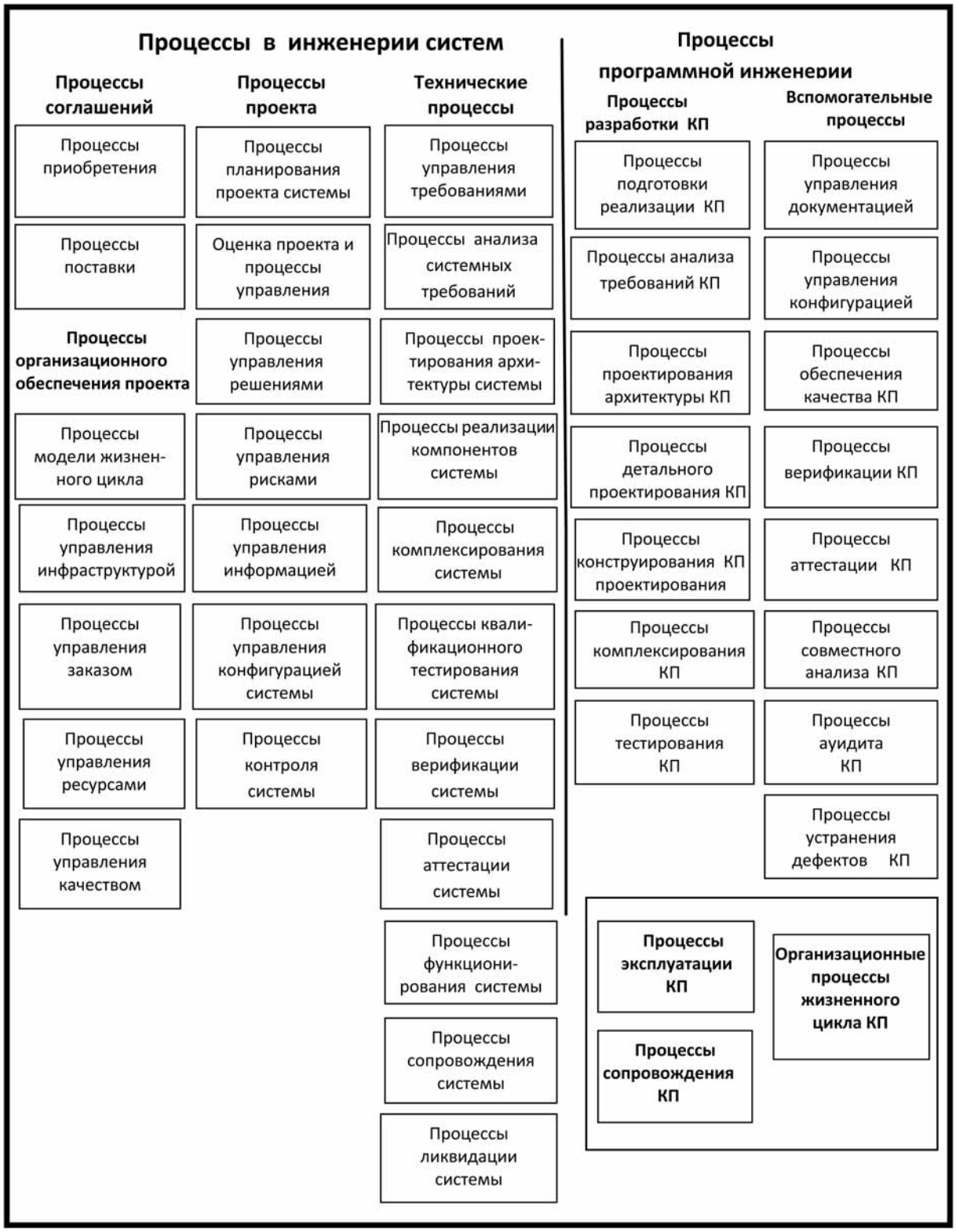
Основу стандартизации в программной инженерии многие годы составлял базовый стандарт **ISO/IEC 12207:1995** — Процессы жизненного цикла программных средств. Его непосредственно поддерживали перечисленные ниже основные стандарты.

— **ISO 15271:1998.** (ГОСТ Р — 2002). ИТ. Руководство по применению ISO 12207.

— **ISO 16326:1999.** (ГОСТ Р — 2002). ИТ. Руководство по применению ISO 12207 при административном управлении проектами.

— **ISO 19759:2005. SWEBOK.** Свод знаний о программной инженерии, а также ряд стандартов по частным процессам жизненного цикла комплексов программ, большинство из которых переведено на русский язык.

Современный вариант стандарта **ISO/IEC 12207:2008** разработан ***для повышения культуры и качества производства организациями, проектирующими, разрабатывающими, приобретающими системы и программные продукты.*** Процессы в стандарте составляют ***полную совокупность*** функций при ***проектировании и производстве заказных программных продуктов***. ***Модель жизненного цикла*** представляется в виде последовательности этапов — процессов, которые могут перекрываться и (или) повторяться циклически в соответствии с областью применения, размером, сложностью, потребностью в изменениях. Стандарт требует, чтобы в каждом проекте определялась подходящая модель жизненного цикла. Предпочтительно чтобы это была та модель, которая уже определялась для применения в ранее выполненных проектах. Разделы ***процессов комплексов программ*** (КП) включают две группы из 18 процессов: реализации и поддержки разработки комплексов (см. рисунок). По названиям



Основные процессы системной и программной инженерии

они в значительной степени подобны процессам в стандарте **ISO 12207:1995**, однако различаются полнотой и конкретностью содержания.

Задачи стандарта выражаются в форме требования, рекомендации или допустимых действий, предназначенных для поддержки достижения выходов процесса. ***Атрибуты*** характеризуют специфику каждого процесса, предварительно определенную цель процесса и его результаты, которые достигаются посредством выполнения определенных действий. Каждый процесс стандарта должен удовлетворять описанным критериям.

Совокупность процессов ***программной инженерии*** подобна ансамблю ***процессов системной инженерии,*** изображенных на рисунке. На нем представлена ***общая структура работ для жизненного цикла систем и для программных комплексов***. Важная характеристика системы стандартов — их целостность. Одним из итогов выполнения этих работ является гармонизация между собой стандартов **ISO/IEC 12207:2008 —** Процессы жизненного цикла программных комплексов и **ISO/IEC 15288:2005 —** Процессы жизненного цикла систем. Процессы ***системной инженерии*** в соответствии со стандартом **ISO/IEC 15288:2005** делятся на ***три группы***, включая процессы соглашений, проекта системы и технические процессы (всего 25 процессов). Стандарт устанавливает общие основы для описания жизненного цикла сложных систем, создаваемых специалистами, определяет детально структурированные процессы и соответствующую терминологию. В стандарте представлены также процессы, которые поддерживают определение, контроль и совершенствование жизненного цикла систем внутри организации или в рамках какого-либо проекта. Стандарт применим к полному жизненному циклу системы, включая замысел, проектирование, производство, эксплуатацию и снятие с эксплуатации, а также приобретение и поставку систем — действий, которые осуществляются внутри или вне выполняющей их организации. В настоящем стандарте не детализируются процессы жизненного цикла систем в терминах методов и процедур, необходимых для удовлетворения требований и достижений результатов процесса. Структура основных компонентов ***стандарта системной инженерии*** связана с ***программной инженерией*** (левая часть рисунка) и далее не комментируется.

Стандарт **ISO/IEC 12207:2008** устанавливает процессы ***жизненного цикла для заказных программных комплексов***, включая процессы и действия, применяемые во время сбора и конфигурации системы. ***Основная цель —*** представление общей структуры стандарта с такой позиции, чтобы покупатели, поставщики, разработчики, специалисты по обслуживанию, операторы, менеджеры и технический персонал, связанный с разработкой программного продукта, ***использовали общий язык.*** **Главные особенности стандарта ISO/IEC 12207 *в редакции* 2008 г.**

- включены и развиты положения **Дополнений 2002 и 2004 г. ISO/IEC 12207**;
- использована терминология, согласованная со стандартом **ISO/IEC 15288:2008**;
- предложено по возможности использовать наименование и структуру процессов аналогичную той, что содержится в стандарте **ISO/IEC 15288:2008**;
- представлена возможность сообществу пользователей получить полностью гармонизированные стандарты и обеспечена стабильность, а именно — стандарт в максимальной мере совместим с прошлыми редакциями;
- использованы результаты десятилетнего опыта разработки и применения стандартов **ISO/IEC 12207** и **ISO/IEC 15288**.

Структура стандарта имеет гибкий, модульный формат, что позволяет адаптироваться к потребностям детализации, которая необходима отдельному пользователю. Процессы делятся на три базовых типа:

- основные;
- вспомогательные;
- организационные.

Группа основных процессов жизненного цикла включает в себя базовые процессы, участвующие в создании и применении программного продукта. Выделяются пять основных групп: заказ; поставка; разработка; эксплуатация; сопровождение.

Процессы заказа определяют работы заказчика, т. е. организации, которая приобретает систему, программный продукт или программную услугу. Каждый этап подразумевает, что предыдущий завершен.

Подготовка заказа. Собрана информация, объясняющая необходимость разработки или модифицирования продукта.

Составлен и утвержден список системных требований.

Определены глобальные требования к программному продукту.

Рассмотрены варианты приобретения готового программного продукта, покупки и модернизации, создания "с нуля".

Проанализированы технические требования.

Подготовлен, документально оформлен и выполнен план заказа, который содержит:

— требования к системе;

— планируемую загрузку системы;

— тип реализуемого договора;

— обязанности организаций, участвующих в договоре;

— обеспечение методов реализации договора;

— анализ возможных рискованных ситуаций, а также методы управления такими ситуациями.

Определены и документально оформлены принятые правила и условия (критерии) реализации договора.

Подготовка заявки на подряд. Документально оформлены требования к заказу, состав которых зависит от вариантов его реализации. Соответствующая документация по заказу должна содержать:

— требования к системе;

— описание области применения системы;

— указания для участников торгов;

— список программных продуктов;

— сроки и условия реализации заказа;

— правила контроля над субподрядчиками;

— технические ограничения (например, по условиям эксплуатации).

Определено, какие из процессов, работ и задач, описанных в настоящем стандарте, применимы к условиям проекта, и соответствующим образом адаптированы.

Определены контрольные пункты договора, при выполнении которых анализируется и проверяется деятельность поставщика.

Требования к заказу представлены организации, выбранной для выполнения работ в процессе заказа.

Подготовка договора. Определена процедура выбора поставщика, включающая критерии оценки поступающих предложений по реализации заказа и их соответствие установленным требованиям.

Выбран поставщик, исходя из оценки предложений, поступивших от потенциальных поставщиков, их возможностей и других рассматриваемых факторов.

В зависимости от того, была ли проведена адаптация настоящего стандарта к условиям проекта, включение в текст договора (или указание на источник) адаптированного настоящего стандарта.

Подписан обновленный контракт с учетом корректировок, внесенных при обсуждении заказчика и поставщика.

Приемка и закрытие договора. Подготовлены контрольные примеры, контрольные данные, процедуры тестирования и условия проведения испытаний. Заказчик должен определить степень участия поставщика при проведении приемки.

Заказчиком проверена готовность поставщика к проведению приемки и проведению приемочных испытаний поставляемого программного продукта.

Заказчиком принят от поставщика продукт (при выполнении всех условий приемки).

После приемки заказчик принимает на себя ответственность за управлением конфигурацией поставленного программного продукта.

Разработка программного продукта. Данный процесс описывает все фазы разработки программного продукта (создание, тестирование и приведение к конечному результату, готовому к сдаче заказчику). Выбор метода разработки зависит от конкретной ситуации. Самый частый метод разработки — **V-модель**:

- подготовка программного комплекса;
- анализ требований технического задания;
- проектирование архитектуры программного комплекса;
- детальное проектирование программного комплекса;
- конструирование программного комплекса;
- комплексирование программного комплекса;
- тестирование.

Подготовка программного комплекса. Выбор модели жизненного цикла программного комплекса, соответствующей области реализации, величине и сложности проекта (если это не указано заказчиком в договоре).

Оформление выходных результатов в соответствии с процессом документирования.

Оформление возникающих проблем и устранение несоответствий, обнаруженных в программных продуктах и задачах, если таковые проявляются при разработке.

Выбор и адаптация стандартов, методов, инструментария, языков программирования (если они не установлены в договоре), которые будут использоваться для выполнения работ в процессе разработки и во вспомогательных процессах.

Разработка плана проведения процессов разработки, которые должны охватывать конкретные стандарты, методы, инструментарий, действия и обязанности, связанные с разработкой и квалификацией всех требований, включая безопасность и защиту.

Анализ требований технического задания. Анализ области применения разрабатываемой системы с точки зрения определения требований к ней.

Оформление требований к программному продукту, которые должны описывать:

— функциональные и технические требования, включая производительность, физические характеристики и окружающие условия среды, под которые должен быть создан программный объект архитектуры (далее — программный объект);

— требования к внешним интерфейсам программного объекта архитектуры;

— квалификационные требования;

— требования безопасности, включая требования, относящиеся к методам эксплуатации и сопровождения, воздействию окружающей среды и травмобезопасности персонала;

— требования защиты, включая требования, относящиеся к допустимой точности информации;

— эргономические требования, включая требования, относящиеся к ручным операциям, взаимодействию "человек—машина", персоналу и областям, требующим концентрации внимания человека, связанным с чувствительностью объекта к ошибкам человека и квалификацией персонала;

— требования к определению данных и базе данных;

— требования по вводу в действие и приемке поставляемого программного продукта на объекте(ах) эксплуатации и сопровождения;

— требования к документации пользователя;

— требования к эксплуатации объекта пользователем;

— требования к обслуживанию пользователя.

Оценка технического задания (ТЗ) с учетом следующих критериев (при этом результаты оценок должны быть документально оформлены):

— учет потребностей заказчика;

— соответствие потребностям заказчика;

— тестируемость;

— выполнимость проектирования системной архитектуры;

— возможность эксплуатации и сопровождения.

Проектирование архитектуры программного средства. Определение общей архитектуры системы (архитектура верхнего уровня). В архитектуре должны быть указаны объекты технических и программных средств и ручных операций. Должно быть обеспечено распределение всех требований к системе между объектами архитектуры.

Оценка системной архитектуры и требований к объектам архитектуры с учетом следующих критериев:

— учет требований к системе;

— соответствие требованиям к системе;

— соответствие используемых стандартов и методов проектирования;

— возможность программных объектов архитектуры выполнять установленные для них требования;

— возможности эксплуатации и сопровождения.

Детальное проектирование программного средства. Трансформирование требований к программному объекту в архитектуру, которая описывает общую структуру объекта и определяет компоненты программного объекта.

Разработка и оформление общего (эскизного) проекта внешних интерфейсов программного объекта и интерфейсов между компонентами объекта.

Разработка и оформление общего проекта базы данных.

Разработка и оформление предварительной версии документации пользователя.

Разработка и оформление предварительных общих требований к тестированию программного объекта и графику сборки программного продукта.

Оценка архитектуры программного объекта и эскизные проекты интерфейсов и базы данных по следующим критериям:

— учет требований к программному объекту;

— внешняя согласованность с требованиями к программному объекту;

— внутренняя согласованность между компонентами программного объекта;

— соответствие методов проектирования и используемых стандартов;

— возможность технического проектирования;

— возможность эксплуатации и сопровождения.

Конструирование программного комплекса. Разработка технического проекта для каждого компонента программного объекта.

Разработка технического проекта внешних интерфейсов программного объекта, интерфейсов между компонентами программного объекта и между программными модулями.

Разработка технического проекта базы данных.

Определение требований к испытаниям и программе испытаний программных модулей.

Оценка технического проекта тестирования по следующим критериям:

— учет требований к программному объекту;

— внешнее соответствие спроектированной архитектуре;

— внутренняя согласованность между компонентами программного объекта и программными модулями;

— соответствие методов проектирования и используемых стандартов;

— возможность тестирования;

— возможность эксплуатации и сопровождения.

Комплексирование программного средства. Разработка и документальное оформление следующих продуктов:

— каждого программного модуля и базы данных;

— процедуры испытаний (тестирования) и данные для тестирования каждого программного модуля и базы данных.

Разработка плана сборки для объединения программных модулей и компонентов в программный комплекс. План должен включать требования к испытаниям (тестированию), процедуры тестирования, контрольные данные, обязанности исполнителя и программу испытаний.

Сбор программных модулей и компонентов.

Сбор программных объектов в единую систему вместе с объектами технической конфигурации, ручными операциями и, при необходимости, с другими системами.

Тестирование. Тестирование в соответствии квалификационным требованиям к программному продукту.

Оценка проекта, запрограммированного программного объекта, тестирование по следующим критериям:

— тестовое покрытие требований к программному объекту;

— соответствие ожидаемым результатам;

— возможность сборки и тестирования продукта и системы (при их проведении);

— возможность эксплуатации и сопровождения.

Тестирование системы и оценка по следующим критериям:

— тестовое покрытие требований к программному продукту и системе;

— соответствие ожидаемым результатам;

— возможность эксплуатации и сопровождения.

Проведение аудиторской проверки и доработка.

Эксплуатация. Процесс эксплуатации состоит из работ и задач оператора. Процесс охватывает эксплуатацию программного продукта и поддержку пользователей в процессе эксплуатации. Так как эксплуатация программного продукта входит в эксплуатацию системы, работы и задачи данного процесса связаны с системой.

Сопровождение. Процесс сопровождения состоит из работ и задач, выполняемых персоналом сопровождения. Данный процесс реализуется при изменениях (модификациях) программного продукта и соответствующей документации, вызванных возникшими проблемами (дефектами) или потребностями в модернизации или настройке. Целью процесса является изменение существующего программного продукта при сохранении его целостности. Данный процесс охватывает вопросы переносимости и снятия программного продукта с эксплуатации. Процесс заканчивается снятием программного продукта с эксплуатации.

Вспомогательные процессы жизненного цикла комплексов программ. Процесс документирования:

— подготовка процесса;

— проектирование и разработка;

— выпуск;

— сопровождение.

Процесс управления конфигурацией:

— подготовка процесса;

— определение конфигурации;

— контроль конфигурации;

— учет состояний конфигурации;

— оценка конфигурации;

— управление выпуском и поставка.

Процесс обеспечения качества:

— подготовка процесса;

— обеспечение продукта;

— обеспечение процесса;

— обеспечение системы качества.

Процесс верификации:

— подготовка процесса;

— верификация.

Процесс аттестации:

— подготовка процесса;

— аттестация.

Процесс совместного анализа:

— подготовка процесса;

— анализы управления проектом;

— технические анализы.

Процесс аудита:

— подготовка процесса;

— аудиторская проверка.

Процесс решения проблем (устранения дефектов):

— подготовка процесса;

— решение проблемы.

Организационные процессы жизненного цикла комплексов программ. Процесс управления:

— определение области управления;

— планирование;

— выполнение и контроль;

— проверка и оценка;

Процесс создания инфраструктуры:

— подготовка процесса;

— создание инфраструктуры;

— сопровождение инфраструктуры.

Процесс усовершенствования:

— создание процесса;

— оценка процесса;

— усовершенствование процесса.

Процесс обучения:

— подготовка процесса;

— разработка учебных материалов;

— реализация плана обучения.

Ответственность за работы и задачи вспомогательного и организационного процессов несет организация, выполняющая эти процессы. Организация гарантирует реальность существования и функциональные особенности конкретного процесса. Она организует и выполняет управление процессами на проектном уровне в соответствии с процессом управления; определяет инфраструктуру для данного процесса в соответствии с процессом создания инфраструктуры; адаптирует процессы к условиям проекта в соответствии с процессом адаптации и управляет процессами на организационном уровне в соответствии с процессами усовершенствования и обучения. В качестве методов обеспечения качества могут быть использованы: совместные анализы, аудиторские проверки, верификация и аттестация.

УДК 004.4'242, 004.4'23

В. М. Ицыксон, канд. техн. наук, доц., **А. В. Зозуля,** ассистент, СПбГПУ,
e-mail: vlad@ftk.spbstu.ru

Автоматизированная трансформация программ при миграции на новые библиотеки

Предлагается подход к автоматизированному портированию программ при переходе на использование новых библиотек. Определяются элементы библиотек, влияющие на взаимодействие с приложением. Описывается разработанный язык, позволяющий задавать частичные спецификации поведения библиотек. Проводится анализ семантической совместимости спецификаций двух библиотек. Формулируются правила преобразования программ в соответствии со спецификациями. Описывается разработанный прототип средства портирования.

Ключевые слова: *портирование приложений, библиотечное окружение, спецификация библиотек, семантика поведения функций, реинжиниринг, трансформация программ*

Введение

Индустрия разработки программного обеспечения в последние годы развивается очень высокими темпами. Это выражается в уменьшении времени, затрачиваемого фирмами на выход новых версий продукта, в многообразии выпускаемых программных продуктов и во множестве целевых программных и аппаратных платформ. Такие темпы развития отрасли определяются следующими факторами:

- бурным развитием нескольких альтернативных программных платформ (Windows, Linux, MacOS);
- массовым распространением производительных мобильных устройств (телефоны, смартфоны, планшеты, нетбуки);
- многообразием мобильных платформ, имеющих поддержку широкого круга разработчиков (Android, iOS, Symbian, Windows Phone и т.п.).

Успешными на таком рынке являются те компании-разработчики, которые могут предоставлять свои решения сразу для нескольких стационарных и мобильных платформ, своевременно их обновлять и подготавливать новые инновационные продукты.

Глобальная проблема поддержки нескольких программных и аппаратных платформ в настоящее время может решаться следующими способами:

- использование многоплатформенных решений и языков программирования (например Java);
- построение фреймворков, основанных на системах управления конфигурациями, позволяющих одновременно разрабатывать приложения для разных платформ;
- проведение портирования успешно работающих приложений на новые платформы.

Первый подход является наиболее универсальным ввиду распространенности языка Java, но имеет ограничения. Они в первую очередь связаны с эффективностью работы Java-приложений на разных платформах, а также с различиями в случае использования платформозависимых (*native*) библиотек.

Второй подход довольно тяжеловесен и требует от разработчиков априорного знания архитектуры всех целевых платформ. Его использование с самого начала разработки может существенно снизить эффективность разработки приложения и повысить ее стоимость. Кроме того, количество платформ постоянно меняется, что только усложняет разработку.

Третий подход, основанный на портировании существующих приложений, является наиболее популярным. Перенос на новую платформу происходит по мере необходимости, и на ранних этапах нет нужды поддерживать все многообразие платформ.

Основным недостатком портирования является необходимость переработки части программного обеспечения, взаимодействующего с библиотеками из целевого окружения. Как следствие — модифицированная программа должна пройти весь цикл проверки качества (тестирование, верификация).

Сходные проблемы возникают при портировании сторонних приложений для использования в новом окружении с новыми библиотеками.

Целью настоящей работы является разработка подходов к автоматизации портирования программного обеспечения в новые библиотечные окружения.

Связанные работы

Работ, которые полностью решают поставленную в данной статье задачу, нет, но имеются результаты исследований в смежных областях.

В ИСП РАН проводятся исследования в области проверки обратной совместимости различных версий библиотек [1] и анализа переносимости приложений между дистрибутивами Linux [2]. Анализируются бинарная и сигнатурная совместимости, а также соответствие стандарту LSB.

Группа подходов, основанных на универсальных языках преобразования текстов, таких как TXL [3] и Startego/XT [4], реализующих правила перезаписи и стратегии перезаписи, предназначена для описания сложных модификаций программ, управляемых правилами. Подходы могут быть применимы при трансформациях, предполагающих простое преобразование сигнатур.

Часть работ посвящена структурному реинжинирингу программного обеспечения. Например, система DMS [5] предназначена для реинжиниринга архитектур ПО. Сходное назначение имеет проект Rascal [6] — предметно-ориентированный язык (DSL), предназначенный для автоматизации анализа, трансформации и синтеза программ.

Для портирования программы, работающей с конкретной библиотекой, можно использовать шаблонный метод трансформации программы. В таких подходах (см., например, [7]) на специальном языке отдельно задаются шаблоны фрагментов программы, использующих эквивалентные конструкции двух библиотек. Сама трансформация программы проводится автоматически на основе построения преобразующего сценария, последовательно приводящего фрагмент одной программы к другой.

Группа подходов к автоматизации реинжиниринга представлена в сборнике [8], где рассматриваются вопросы реинжиниринга программ, написанных на устаревших языках программирования, а также подходы и инструментальные средства реструктуризации программ на основе реверс-инжиниринга.

Все указанные подходы, так или иначе, связаны с автоматизацией процесса преобразования программного обеспечения. Но без существенных модификаций они неприменимы к задаче миграции на новые библиотеки, так как ориентированы на определенный класс преобразований и не могут управляться семантикой библиотек**.** Необходима разработка новых подходов, позволяющих реализовывать семантически ориентированную трансформацию.

Методика портирования программного обеспечения

Предлагаемый в данной работе подход к портированию основывается на формализации семантики исходной и конечной библиотек. При этом предлагается специфицировать только ту часть семантики библиотек, которая непосредственно влияет на исходное приложение, абстрагируясь от тонкостей реализации возможностей библиотеки.

Для преобразования программной системы, использующей одну библиотеку, в функционально эквивалентную программную систему, использующую другую библиотеку, предлагается следующая методика:

- создание спецификации семантики исходной библиотеки;
- создание спецификации семантики новой библиотеки;
- проверка семантической совместимости обеих библиотек;
- в случае совместимости:
  - идентификация в программе всех мест, где происходит взаимодействие со старой библиотекой;
  - анализ возможности проведения автоматизированного преобразования для конкретного исходного кода;
  - преобразование программы с заменой всех примитивов взаимодействия со старой библиотекой на соответствующие новые, основываясь на анализе исходного кода программы и двух спецификаций.

Более подробно подход изложен в работе [9]. В следующих разделах конкретизируются отдельные положения этого подхода.

Спецификация взаимодействия программ и библиотек

Для осуществления трансформации необходимо определить все способы взаимодействия программы с библиотеками, а также специфицировать семантику самой библиотеки.

На рис. 1 (см. вторую сторону обложки) изображена гипотетическая программа, использующая библиотеку работы с сокетами[1]. В левой части рисунка располагаются компоненты, взаимодействующие с библиотекой, в правой части — элементы библиотечного ок-

[1] Здесь и далее в работе будут рассматриваться аспекты, связанные с языком программирования С. Такой выбор обусловлен высокой популярностью этого языка и актуальностью задачи миграции С-программ в новые окружения.

ружения. К таким элементам относятся библиотечные функции, функции, инициализирующие библиотеку и освобождающие память, подключаемые заголовочные файлы, глобальные переменные.

На рис. 1 (см. вторую сторону обложки) продемонстрированы основные механизмы взаимодействия программ и библиотек. К таким механизмам относятся:

- передача параметров в библиотечные функции;
- возврат значения библиотечной функцией;
- передача данных через глобальные переменные (объявленные как в основной программе, так и в библиотеке).

Передача данных через параметры библиотечных функций и получение результатов через возвращаемые значения являются одними из основных способов взаимодействия программы с библиотекой. При их спецификации необходимо задавать их общую семантику и, при необходимости, интерпретацию конкретных значений.

Использование глобальных переменных является наиболее сложно-контролируемым способом взаимодействия программ и библиотек. Широкое использование такого способа в библиотеке может сделать процесс автоматизированного портирования чрезвычайно сложным и потребовать проведения полного статического анализа программы.

Кроме способов взаимодействия библиотека характеризуется побочными эффектами, которые чаще всего и инкапсулируют основную семантику библиотеки. Побочные эффекты библиотек могут заключаться в выполнении следующих операций:

- изменении глобальных объектов программы;
- создании, модификации или удалении ресурсов — объектов операционной системы, имеющих свой жизненный цикл (файлы, потоки, семафоры, сокеты и т. п.);
- выполнении определенных действий с окружением (например, открытие файла, получение данных из сети и т. п.).

Такие побочные эффекты специфицируются как часть поведенческих описаний библиотечных функций.

Дополнительно для библиотек необходимо специфицировать:

- список подключаемых (для языка С — заголовочных) файлов;
- функции инициализации/деинициализации библиотек.

Список заголовочных файлов при портировании необходим для корректной модификации программы. Обычно для библиотеки такой список содержит несколько заголовочных файлов и редко пересекается со списками других библиотек.

Функции инициализации библиотек используются для спецификации начальных значений или создания объектов библиотеки (переменные и ресурсы). Функции завершения работы библиотеки осуществляют уничтожение необходимых объектов.

Для описания всех способов взаимодействия, обеспечиваемых конкретной библиотекой, побочных эффектов и поведения библиотечных функций разработан язык частичных спецификаций окружений PanLang, задача которого — дать программисту средство, позволяющее описать ту часть семантики библиотеки, которая отражает ее видимое поведение [10]. Язык основан на формализме, описанном в работе [11], и дает возможность строго выразить информацию о функционировании библиотеки, обычно задаваемую неформально в виде текстовых описаний, руководств пользователя и другой документации.

Задание спецификаций для библиотек сокетов

Рассмотрим как будет выглядеть описание библиотеки работы с сокетами в операционной системе Linux на языке PanLang.

Перечень используемых библиотекой заголовочных файлов задается директивой `requires`.

```
requires <sys/types.h>;
requires <sys/socket.h>;
```

При проведении портирования необходимо знать, какой семантикой нагружен каждый передаваемый в библиотечную функцию параметр и возвращаемое значение. Для этого вводятся специальные элементы модели — семантические типы. Они являются расширением обычных типов языка С и задают семантическую интерпретацию как самого типа данных, так и отдельных значений объектов этого типа. В примере ниже семантические типы `SOCKET_DOMAIN`, `SOCKET_PROTOCOL` и `SOCKET_TYPE` задают интерпретацию типа `int` при использовании в качестве параметров системного вызова `socket`. Тип `SOCKET_TYPE` дополнительно содержит семантическую интерпретацию значений типа `int`, когда этот тип используется как тип возвращаемого значения функции `socket`.

```
semantic type SOCKET_DOMAIN(int);
semantic type SOCKET_PROTOCOL(int);
semantic type SOCKET_TYPE(int) {
   ERR: -1;
   OK: [1 : +inf];
};
semantic type SOCKET_RESULT(int) {
   ERR: -1;
   OK: 0;
};
semantic type RECV_SEND_RESULT(int) {
   ERR: -1;
   OK: [0 : +inf];
};
```

Результатом работы функции `socket` является создание ресурса операционной системы — сокета.

В примере ниже показано описание ресурса SOCKET с четырьмя состояниями и атрибутом SHUTDOWN_HOW.

```
semantic type SHUTDOWN_HOW(int) {
   READ: SHUT_RD;
   WRITE: SHUT_WR;
   READ_WRITE: SHUT_RDWR;
};
resource SOCKET(SOCKET_TYPE) {
   states CREATED, CONNECTED, LISTENING, BOUND;
   attribute SHUTDOWN_HOW SHUTDOWN_TYPE;
};
```

Во время работы функции библиотеки могут иметь побочные эффекты, связанные с взаимодействием с окружением. В терминах разработанной модели они выражаются действиями, которые задают явную семантику побочного эффекта. Объявления действий для библиотеки сокетов приведены ниже.

```
action void SEND(SOCKET socket);
action void RECV(SOCKET socket);
```

Спецификация библиотечных функций состоит из задания сигнатуры функции и ее поведения. Сигнатура функции описывает в терминах определенных типов семантику аргументов и возвращаемого значения.

Поведение функций библиотеки задается императивно и может содержать операции работы с ресурсами, глобальными переменными и конструкции выполнения действий. Примеры описания нескольких функций библиотеки сокетов приведены ниже.

```
function SOCKET socket (
     SOCKET_DOMAIN, SOCKET_TYPE,
     SOCKET_PROTOCOL) {
   result = new SOCKET(CREATED);
}
function CONNECT_RESULT connect (
     SOCKET s, SERV_ADDR, ADDR_LEN) {
   state(s) = $CONNECTED;
}
function SEND_RESULT send (
     SOCKET s, MSG, MSG_SIZE, FLAGS) {
   if (state(s) == $CONNECTED) {
     action SEND (s);
   }
}
function RECV_RESULT recv (
     SOCKET s, MSG, MSG_SIZE, FLAGS) {
   if (state(s) == $CONNECTED) {
     action RECV (s);
   }
}
function SHUTDOWN_RESULT shutdown(
     SOCKET s, SHUTDOWN_HOW how) {
   attr(s, SHUTDOWN_TYPE) = how;
}
function CLOSE_RESULT close (SOCKET s) {
   delete(SOCKET) s;
}
```

Представленные примеры демонстрируют спецификацию части библиотеки BSD socket, использующейся в ОС Linux. Для автоматизированного переноса приложения в ОС Windows необходимо также иметь подобную спецификацию для аналогичной библиотеки Winsock. Она будет во многом похожа на спецификацию BSD socket, но с несколькими отличиями. Основные из них показаны в листингах.

Спецификация заголовочных файлов состоит из объявления одного подключаемого файла.

```
requires <Winsock2.h>;
```

Некоторые типы в Winsock отличаются списком или значениями используемых констант.

```
semantic type SHUTDOWN_HOW (int) {
   READ: SD_RECEIVE;
   WRITE: SD_SEND;
   READ_WRITE: SD_BOTH;
};
```

Дополнительно библиотека Winsock содержит описание специального ресурса, индицирующего инициализирована ли библиотека.

```
   resource WINSOCK_LIB {
   states NONINIT, INIT;
};
```

В секциях инициализации и завершения создается и освобождается глобальный ресурс WINSOCK_LIB, отображающий состояние библиотеки.

```
initializer{
   int winsocklib;
   winsocklib = new WINSOCK_LIB(NONINIT);
};
finalizer{
   delete(WINSOCK_LIB) winsocklib;
};
```

Инициализация и прекращение использования библиотеки Winsock осуществляется специальными функциями.

```
function WSA_RESULT WSAStartup(
     VERSION v = WSA_VERSION, DATA d) {
   state(winsocklib) = $INIT;
};
function WSA_RESULT WSACleanup() {
   state(winsocklib) = $NONINIT;
};
```

Представленные фрагменты частичных спецификаций окружений демонстрируют особенности создания спецификаций для различных библиотек. Эти спецификации задают семантику структуры и поведения двух библиотек, решающих одинаковые или похожие задачи. Каждая спецификация задает отображение библиотеки в свой семантический домен. В общем случае для задания правил портирования одной библиотеки в другую необходим еще один элемент —

правила соответствия (рис. 2, см. вторую сторону обложки). Правила соответствия определяют способы проецирования одного семантического домена в другой.

В приведенных примерах при создании спецификаций намеренно применялся упрощенный вариант задания, при котором использовался один целевой семантический домен, в который проецировалась семантика обеих библиотек (рис. 3, см. вторую сторону обложки). Такое проецирование реализовывалось неявно с помощью использования одинаковых имен семантических элементов: типов, типизированных констант и действий.

В более общем случае, когда каждая библиотека проецируется в свой семантический домен, правила соответствия доменов также описываются на языке PanLang. Описание этой части языка выходит за рамки данной работы.

Совместимость библиотек

Два библиотечных окружения будем называть совместимыми, если любое поведение программы, использующей одну библиотеку, может быть выражено с помощью примитивов другой библиотеки. Такое заключение может быть сделано на основе анализа частичных спецификаций библиотек. Каждая такая спецификация задает модель поведения компонентов библиотеки. Математически такая модель выражается конечным автоматом (или точнее — системой конечных автоматов). Эти конечные автоматы явно и неявно задаются ресурсами и поведенческими описаниями функций и определяют конечное или, чаще, бесконечное множество трасс, которые могут быть пройдены компонентами библиотеки.

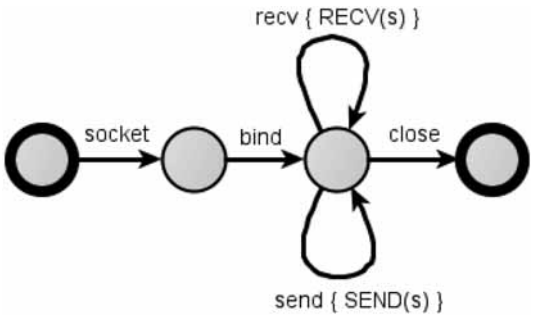


Рис. 4. Упрощенная модель поведения библиотеки работы с UDP-сокетами BSD socket

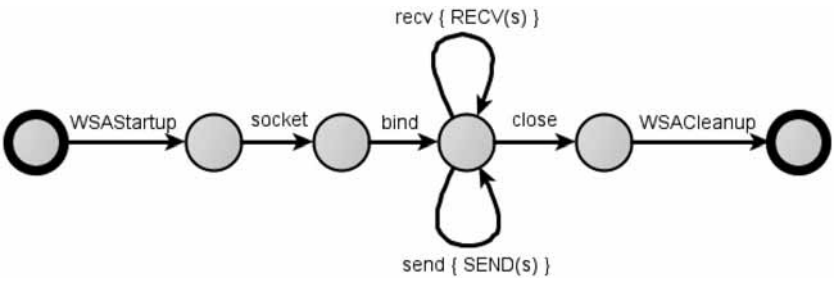


Рис. 5. Упрощенная модель поведения библиотеки работы с UDP-сокетами Winsock

Каждая трасса определяется состояниями, проходимыми компонентами библиотеки, и генерируемыми побочными действиями. На рис. 4 приведена упрощенная модель части библиотеки BSD socket, ориентированной на работу с UDP-сокетами. Узлы отображают состояния, а дуги соответствуют вызовам функций socket, bind, close, recv и send из API-библиотеки. Семантические действия, выполняемые функциями, отображаются на дугах в фигурных скобках.

На рис. 5 изображена аналогичная упрощенная модель библиотеки Winsock. Основное отличие в поведении заключается в необходимости явного вызова функций WSAStartup и WSACleanup, инициализирующих и завершающих работу библиотеки.

С точки зрения реализуемой библиотекой функциональности, важной является последовательность действий, генерируемая в процессе прохождения трассы. Последовательность выполняемых действий отображает динамическую семантику библиотеки при определенном поведении. Если все возможные последовательности действий одной библиотеки могут быть реализованы в модели другой библиотеки, то это означает, что библиотеки потенциально совместимы между собой. Потенциальная совместимость свидетельствует о возможности автоматизированной трансформации программы.

Возможные последовательности выполняемых действий, реализуемые моделью библиотеки BSD socket (см. рис. 4), могут быть, например, такими:

- <RECV>;
- <RECV, RECV>;
- <RECV, SEND, RECV>, SEND, ...;
- <SEND> и т. п.

Для того чтобы функциональность библиотеки BSD socket можно было реализовать с помощью Winsock, все трассы (в том числе перечисленные) должны реализовываться с помощью модели библиотеки, представленной на рис. 5.

Как можно увидеть при анализе моделей, изображенных на рис. 4 и 5, в данном тривиальном случае (имеются в виду упрощенные модели библиотек, работающие с UDP-сокетами и ограниченные перечисленными вызовами API) множество трасс действий библиотеки Winsock совпадает с множеством трасс действий библиотеки BSD socket. Это означает, что библиотеки между собой совместимы и возможно реализовать функциональность одной библиотеки средствами другой. Сами же вызовы библиотек и их последовательность могут существенно отличаться в начальной программе и трансформированной.

Если говорить более формально, то правило совместимости выглядит следующим образом: две библиотеки являются совместимыми, если множество трасс действий, порождаемых первым автоматом, является подмножеством множества трасс действий, порождаемых вторым автоматом. Таким образом, задача совместимости сводится к задаче проверки гомоморфизма

трассовых автоматов. В случае полной схемы правил портирования (см. рис. 2, вторая сторона обложки), задача совместимости сводится к более сложной задаче параметрического преобразования автоматов.

Трансформация программы

Трансформация программы проводится только в случае успешного прохождения процедуры проверки на совместимость исходной и целевой библиотек. При проведении трансформации решаются следующие крупные задачи:

- разбор текста программы и формирование абстрактного семантического графа (АСГ);
- идентификация элементов АСГ, отвечающих за взаимодействие с библиотекой;
- замена подключаемых файлов исходной библиотеки на файлы целевой библиотеки;
- добавление в граф узлов, соответствующих функциям преобразования типов (для параметров библиотечных функций, возвращаемых значений и глобальных объектов), функции преобразования синтезируются на основе описаний семантических типов;
- модификация узлов графа, отвечающих за вызовы функций, в соответствии с сигнатурами функций, их семантикой и порождаемыми действиями;
- модификация обращений к глобальным объектам.

На основе преобразованного АСГ восстанавливается текст программы в соответствии с синтаксисом и семантикой языка С.

Практическая реализация

Изложенный подход был реализован в прототипе программного средства миграции приложений, написанных на языке программирования С. Прототип состоит из следующих компонентов:

- парсер исходной программы и построитель модели;
- транслятор языка частичных спецификаций;
- модуль проверки совместимости;
- модуль преобразования;
- модуль восстановления программного кода.

Парсер программы используется для разбора исходного кода программы на языке С и построения абстрактного синтаксического дерева программы. Для парсинга исходного текста используется средство front-end CLang [12] системы анализа, трансформации и оптимизации программ LLVM [13]. Полученное после применения CLang XML-представление программы преобразуется в абстрактный семантический граф, представленный объектами Java. В дальнейшем построенный граф расширяется дополнительными семантическими элементами.

Транслятор частичных спецификаций получает на вход семантическое описание библиотеки на языке PanLang и транслирует его во внутреннее модельное представление. В состав построенной модели входят все описанные выше примитивы: семантические типы, ресурсы, сигнатуры функций, поведенческие описания функций.

Модуль проверки совместимости анализирует возможность выражения одной конечно-автоматной модели примитивами другой. При этом анализируется совместимость семантических доменов и соответствие друг другу порождаемых моделью трасс.

Модуль преобразования выполняет основную функцию — трансформацию модельного представления для использования его с новой библиотекой. Преобразованию подвергается абстрактный семантический граф, в соответствии с модельными представлениями спецификаций библиотек.

Модуль восстановления программного кода формирует целевую С-программу путем обхода модифицированного абстрактного семантического графа.

Разработанный прототип реализован на языке программирования Java в виде утилиты командной строки. Апробация прототипа происходила на наборе тестовых примеров, таких как:

- перевод приложения, использующего потенциально опасные функции системной библиотеки, на использование безопасных функций;
- перевод приложения, использующего стандартную библиотеку ввода-вывода С, на использование системных вызовов Unix;
- частичное портирование многопоточных приложений с библиотеки Posix Thread (*pthread*) на библиотеку Windows Threads и обратно;
- миграция сетевого приложения, использующего библиотеку сокетов, с операционной системы Linux в Windows и обратно.

Для всех перечисленных тестовых примеров результат портирования соответствовал ожиданиям, а полученные программы компилировались и успешно функционировали.

Текущая версия прототипа имеет некоторые ограничения. Не поддерживается портирование программ, содержащих указатели на функции библиотек, взаимодействующих с библиотеками через сложные структуры данных, состоящих более чем из одного модуля трансляции и оперирующих сложными выражениями.

Заключение

В данной работе представлены результаты научного исследования в области автоматизации переноса программ в новые библиотечные окружения. Разработан язык описания частичных спецификаций, определена процедура проверки совместимости двух библиотек, предложена методика реинжиниринга программ на основе частичных спецификаций. Предложенные алгоритмы и методы легли в основу прототипа средства автоматизации портирования.

Разработанные подходы к портированию программ и реализованный прототип могут рассматриваться как основа для построения промышленного средства миграции приложений, пригодного для ав-

томатизации реинжиниринга широкого класса программ.

Направления дальнейших исследований связаны в первую очередь с преодолением существующих ограничений разработанного прототипа, а также с адаптацией созданных подходов на другие языки программирования. Наиболее интересным направлением адаптации является расширение подхода на объектно-ориентированные языки программирования, такие как Java, C++ и C#.

Список литературы

1. **Ponomarenko A., Rubanov V., Khoroshilov A.** A system for backward binary compatibility analysis of shared libraries in Linux // Proc. of Software Engineering Conference in Russia (CEE-SECR). 5th Central and Eastern European. Moscow, 2009. P. 25—31.

2. **Rubanov V.** Automatic Analysis of Applications for Portability Across Linux Distributions // Proc. of the Third International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2009). Electronic Communications of EASST. 2009. Vol. 20. P. 1—9.

3. **Cordy J. R.** The TXL source transformation language // Science of Computer Programming, 2006. N 61 (3). P. 190—210.

4. **Bravenboer M., van Dam A., Olmos K.** Eelco Visser Program Transformation with Scoped Dynamic Rewrite Rules. Technical Report UU-CS-2005-005. Department of Information and Computing Sciences. Utrecht University.

5. **The DMS®** Software Reengineering Toolkit. URL: http://www.semdesigns.com/Products/DMS/DMSToolkit.html.

6. **Klint P., van der Storm T., Vinju J.** RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation // Proc. of Ninth IEEE International Working Conference on Source Code Analysis and Manipulation. 2009. P. 168—177.

7. **Itsykson V., Timofeev D.** Source code modification technology based on parameterized code patterns // Proc. of Software Engineering Conference in Russia (CEE-SECR). 6th Central and Eastern European. Moscow, 2010. P. 207—213.

8. **Автоматизированный** реинжиниинг программ / Под. ред. проф. А. Н. Терехова и А. А. Терехова. СПб.: Изд-во С.-Петербургского университета, 2000. 332 с.

9. **Itsykson V., Zozulya A., Glukhikh M.** Automated Program Reengineering when Porting Software to a New Environment Described by Partial Specifications // Proc. of 2nd Workshop "Program Semantics, Specification and Verification". St. Petersburg, 2011. P. 111—119.

10. **Ицыксон В. М., Глухих М. И.** Язык спецификаций поведения программных компонентов // Научно-технические ведомости СПбГПУ. 2010. № 101. С. 63—71.

11. **Ицыксон В. М., Зозуля А. В.** Формализм для описания частичных спецификаций компонентов программного окружения // Научно-технические ведомости СПбГПУ. 2011. № 128. С. 81—91.

12. **CLang**: a C Language Family Front-end for LLVM. URL: http://clang.llvm.org/.

13. **The LLVM** Compiler Infrastructure. URL: http://llvm.org/

УДК 004.424.32, 621.039

В. А. Васенин[1], д-р физ. мат.-наук, проф., зав. лаб., e-mail: vasenin@msu.ru
М. А. Кривчиков[1], программист 2-й кат., e-mail: maxim.krivchikov@gmail.com
А. Е. Крошилин[2], д-р техн. наук, зам. ген. дир., e-mail: getget@online.ru
В. Е. Крошилин[3], д-р физ.-мат. наук, проф., e-mail: vkrosh@online.ru
А. Д. Рагулин[1], программист, e-mail: alex.der.igel@gmail.com
В. А. Роганов[1], стар. науч. сотр., e-mail: raduga@butovo.com
[1]НИИ механики МГУ имени М. В. Ломоносова,
[2]ОАО "ВНИИАЭС",
[3]Механико-математический факультет МГУ имени М. В. Ломоносова

Распараллеливание расчетного кода улучшенной оценки "БАГИРА" для моделирования трехмерной теплогидродинамики многофазных сред в составе полномасштабной суперкомпьютерной модели "Виртуальная АЭС"

Рассматриваются подходы к созданию высокопроизводительной параллельной версии расчетного кода "БАГИРА" для улучшенной оценки теплогидравлических характеристик атомных электростанций с водо-водяными энергетическими реакторами; приводится перспективная схема распараллеливания вычислений.

В связи с необходимостью ускорения решения системы линейных уравнений с разреженной матрицей приводятся результаты тестирования наиболее перспективных программных реализаций для суперЭВМ, использующих различные численные методы.

Ключевые слова: *моделирование многофазных потоков, теплогидравлический расчетный код, расчетный код улучшенной оценки, ядерные реакторы, водо-водяные ядерные реакторы, суперкомпьютеры, параллельные вычисления*

Введение

Вопросы безопасности существующих и проектируемых атомных электростанций (АЭС) со временем становятся еще более актуальными в свете происходящих в последние десятилетия чрезвычайных ситуаций на АЭС в различных странах. В условиях, когда серьезной альтернативы атомной энергетике пока не найдено, а цена ошибок при эксплуатации АЭС высока, следует прилагать активные усилия, направленные на исследования и прогнозирование потенциально возможных неблагоприятных событий и следующих за ними ситуаций чрезвычайного характера.

В настоящее время во Всероссийском научно-исследовательском институте по эксплуатации атомных электростанций (ВНИИАЭС) разрабатывается супер-

компьютерная полномасштабная модель атомной электростанции, которая именуется "Виртуальная АЭС". Такая модель поможет в значительной степени продвинуться в решении следующих задач:

- расчет штатных режимов эксплуатации существующих и проектируемых перспективных АЭС;
- качественная тренировка и обучение персонала на имитационных моделях (симуляторах) АЭС в режиме реального времени;
- выработка рекомендаций по предотвращению и минимизации последствий всевозможных нештатных ситуаций, которые могут произойти на существующих и проектируемых АЭС;
- упреждающее моделирование в целях подготовки и принятия оперативных (в режиме реального времени) решений по предотвращению и выходу из нештатных и чрезвычайных ситуаций.

Полномасштабная модель АЭС создается на основе хорошо зарекомендовавших себя расчетных кодов, среди которых важную роль играет получивший лицензию в Ростехнадзоре РФ и уже используемый на практике в режиме тестовых испытаний программный комплекс для улучшенной оценки (*best-estimate code* [1]) параметров модели АЭС "БАГИРА" [2—4]. Этот комплекс создан для моделирования в реальном масштабе времени процессов трехмерной теплогидромеханики двухфазного теплоносителя в различных компонентах и цепях первого и второго контуров АЭС.

Вместе с тем повышение точности вычислений с использованием полной модели АЭС, уменьшение масштаба, и, как следствие, увеличение размеров расчетной сетки в различных ее компонентах, а также настоятельная необходимость в организации режима упреждающего моделирования требуют более высокопроизводительной, чем используемая в настоящее время, реализации расчетных кодов. Для этого проводится адаптация кодов существующей модели АЭС к платформам современных суперЭВМ, основной особенностью которых является массовый параллелизм и распределенная по вычислительным узлам оперативная память. Компактные варианты подобных супервычислительных устройств в перспективе позволят оснастить автономными системами моделирования поведения АЭС любые заинтересованные в этом организации, включая различные учебные заведения и ситуационные центры.

Однако современным суперкомпьютерам свойственны некоторые ограничения, которые делают распараллеливание данного программного комплекса достаточно сложной для решения задачей.

Постановка и актуальность задачи расчета трехмерной теплогидродинамики многофазных сред

С помощью традиционно используемого одномерного приближения не всегда возможно с должной степенью адекватности описать течение в циркуляционных контурах АЭС. В частности, ограниченный характер такого подхода проявляется при моделировании течений в корпусах реакторов и парогенераторов, когда распределение параметров теплоносителя по нескольким направлениям неоднородно. Как показывают данные наблюдений на действующих АЭС и соответствующий численный анализ, многомерные эффекты нередко становятся существенными (или даже определяющими). Такие факторы, например, проявляются при асимметричном распределении расхода теплоносителя по петлям; в случае работы энергоблока на неполном числе петель; если нарушен теплоотвод со стороны второго контура; при внесении положительной или отрицательной реактивности в ограниченный сектор активной зоны; в случае отказа части спринклерных систем безопасности; при локальном разрушении активной зоны.

Таким образом, корректное описание многомерных течений многофазных сред чрезвычайно важно с точки зрения всестороннего анализа безопасности АЭС. Несмотря на перечисленные обстоятельства, в подавляющем большинстве современных теплогидравлических программ улучшенной оценки для моделирования многофазных сред в АЭС применяются, в силу простоты их программной реализации, одномерные математические модели. Использование суперкомпьютеров с большим числом процессоров позволяет применять для этого трехмерную модель механики многофазных сред.

Особенности используемых численных методов

Теплогидравлика активной зоны реактора и контуров АЭС с водо-водяными энергетическими реакторами (ВВЭР) относится к разряду сложно обусловленных процессов. Наличие активно взаимодействующих пара и воды, а также всевозможных их сочетаний, нестационарный, турбулентный характер течений в активной зоне реактора и других компонентах системы делает адекватное физическое описание, которое ожидается от расчетного кода улучшенной оценки, непростой задачей. Вычислительное ядро такого расчетного кода занимает десятки тысяч строк и имеет мало общего с аналогичными программными компонентами, которые традиционно используются для расчета течения идеальной несжимаемой жидкости.

С учетом изложенного выше, рассматривается общий случай течения многофазной смеси, который включает следующие положения:

- смесь состоит из жидкой и газообразной (пар и неконденсирующийся газ) фаз;
- смесь неравновесна по температуре, т. е. температуры фаз могут отличаться друг от друга и от температуры насыщения;
- смесь неравновесна по скорости;
- моделируются нестационарные, трехмерные течения смеси.

Выписываются дифференциальные уравнения сохранения массы, энергии и импульса механики многофазных сред. Учитывается турбулентный тепло- и массоперенос. Для расчета турбулентного перемешивания применяется модель Прандтля, использующая понятие длины смешения.

Слагаемые в уравнениях представляются как линейные функции давления и объемной скорости смеси. С использованием полученных линейных соотношений и дифференциальных уравнений строятся неявные по давлению и объемной скорости конечно-разностные уравнения. Получается полунеявная численная схема, которая является неявной по давлению, объемной скорости смеси и явной по всем остальным переменным. Исключая объемную скорость из конечно-разностных уравнений, строится уравнение, связывающее давление в расчетной ячейке с давлениями во всех граничащих с ней ячейках. В общем трехмерном случае ячейка граничит с шестью ячейками. По этой причине получается "семидиагональная" матрица для определения давления на новом временном слое. По рассчитанному давлению определяются все остальные переменные.

Рассмотрим факторы, важные для получения эффективной суперкомпьютерной версии данного расчетного кода.

Специфика суперкомпьютерной версии расчетного кода улучшенной оценки "БАГИРА"

Наиболее высокопроизводительные версии суперкомпьютерных программ практически во всех случаях, так или иначе, используют три перечисленных далее вида оптимизации:

- максимально эффективное распараллеливание счета;
- специализацию по операциям и данным;
- низкоуровневую оптимизацию, включая минимизацию объема пересылаемых данных.

Характеристики современных суперЭВМ дают повод ожидать качественного скачка как в скорости, так и в точности моделирования процессов, протекающих в АЭС. На настоящее время мы имеем значительно более высокую производительность современных суперЭВМ, которая позволяет проводить моделирование сложно организованных объектов и процессов с куда большей степенью их детализации и скоростью вычислений, чем несколько лет назад. После соответствующей доработки базовая среда моделирования допускает возможность использования MPI-процессов, которые могут задействовать все вычислительные ядра суперкомпьютера. Как следствие, разделив моделируемую систему на достаточно большое число частей, можно попробовать действовать прямолинейно и, на манер явной схемы, обсчитывать все эти части независимо, периодически проводя необходимый локальный обмен данными. Подобное разбиение области моделирования на отдельные взаимодействующие компоненты используется достаточно часто (см., например, [5]).

В случае использования явных схем результат вычисляется через несколько соседних точек данных, однако они часто оказываются неустойчивыми. Неявные схемы используют уравнения, которые выражают данные через несколько соседних точек результата. Они, как правило, устойчивы, однако их использование влечет за собой необходимость решать на каждом шаге систему линейных алгебраических уравнений (СЛАУ). Полунеявные схемы — это схемы расчета, когда на одних шагах применяется явная, а на других неявная схема.

К сожалению, изложенный выше прямолинейный подход к моделированию обладает определенными недостатками. Прежде всего они связаны с явлениями на стыках подлежащих расчету областей, которые могут негативно влиять на устойчивость счета и даже привести к так называемому явлению *разболтки*. Термином *разболтка* здесь именуется неадекватное поведение вычислительных методов по отношению к моделируемым реальным физическим процессам. Следует заметить, что в данном случае вовсе не имеются в виду реально наблюдаемые на атомных станциях процессы, которые также бывают нестационарными.

Принимая во внимание отмеченные выше обстоятельства, представляется целесообразным попытаться *адаптировать* существующие и хорошо отлаженные алгоритмы к параллельной среде исполнения, несмотря на то, что они разрабатывались для последовательных ЭВМ. В этих алгоритмах на определенных шагах расчета решается система линейных уравнений, причем *в идеале* это должно делаться для всей области. Именно такая *полная* модель призвана ликвидировать проблему стыков, улучшить точность расчетов и максимально повысить уровень адекватности описания процессов, протекающих в виртуальной АЭС. Однако в этом случае мы имеем уже не слабосвязанную систему практически независимых вычислительных процессов, а систему, сочетающую относительно независимые (по расчетной области) вычисления со сбором коэффициентов глобальной матрицы достаточно большого размера. Систему уравнений, которая задается глобальной матрицей, придется решать параллельно, в реальном времени и даже быстрее.

Квант времени на один такой шаг моделирования не столь велик и по результатам экспериментов в рассматриваемом случае составляет единицы миллисекунд. Именно этот фактор и делает решаемую задачу нетривиальной. Причина в том, что современные суперЭВМ, которые ориентированы на MPI в качестве базового средства распараллеливания, в отношении таких *глобальных* полномасштабных взаимодействий имеют определенную слабость.

Ниже кратко рассматриваются временные характеристики каждой стадии счета.

Первичные результаты профилирования вычислительного ядра кода "БАГИРА"

Единичный шаг моделирования динамики многофазного потока делится на стадии и представлен соответствующей последовательностью вызовов отдельных процедур для пересчета параметров среды. Первичное профилирование кода дало следующее распределение счетного времени по ключевым (наиболее ресурсоемким) процедурам:

- Hetth (9 %) — расчет потока теплоты;
- Sthd04 (9 %) — линеаризация физических уравнений;
- Sthd06 (20 %) — вычисление коэффициентов матрицы;
- Sthd08 (23 %) — решение системы линейных уравнений;
- Sthd09 (5 %) — расчет основных параметров в ячейках;
- Sthd10 (25 %) — итоговый пересчет всех характеристик среды.

Дадим краткую характеристику каждого из перечисленных модулей в части их распараллеливания.

Распараллеливание подготовки матричных коэффициентов и пересчета характеристик многофазной среды

Модуль Hetth рассчитывает теплоперенос. Принимается во внимание радиоактивность и перемещение среды. Модуль представлен в виде двух последовательно идущих циклов, каждый из которых распараллеливается независимо и заканчивается синхронизацией отработавших процессов.

Модуль Sthd04 выполняет линеаризацию уравнений. При выполнении расчета коэффициентов для каждой ячейки берутся также значения из соседних ячеек. В дальнейшем предполагается использовать один из методов распараллеливания по пространству.

Модуль Sthd06 занимается подготовкой коэффициентов матрицы для шага полунеявной схемы и рассчитывает как одномерные, так и трехмерные фрагменты. Здесь также была использована пространственная декомпозиция, сводящаяся к расщеплению счетных циклов на отдельные фрагменты.

Модуль Sthd08 осуществляет решение системы линейных уравнений итерационным методом верхней релаксации. На настоящее время наиболее эффективным решением оказалась смена алгоритма с итерационного на точный. Прекрасно показали себя реализации различных алгоритмов в Open Source библиотеках Pastix, UMFPack, а также коммерческая библиотека PARDISO. Корректность точного решения проверялась несколькими временными срезами работы системы и сравнениями с решениями традиционным методом.

После анализа результатов этих экспериментов вызов модуля Sthd08 был заменен на вызов специально разработанной библиотеки SMatrix, которая может инкапсулировать в себе сразу несколько методов решения СЛАУ. Целесообразность такой замены обусловлена тем, что для разреженных матриц разного размера и конфигурации лидирующие позиции по эффективности занимают разные алгоритмы. Особенности разработанной библиотеки SMatrix будут рассмотрены ниже.

Модули Sthd09 и Sthd10 выполняют расчет интегральных и всех остальных характеристик среды для всех вычислительных ячеек после вычисления давления. При решении этих подзадач применялось распараллеливание вычислительного процесса с ускорением, практически линейным по числу процессоров.

Кратковременный характер каждой из перечисленных выше стадий расчета повлек за собой необходимость возможности реализации на различных архитектурных уровнях вычислительной среды в целях минимизации накладных расходов при согласовании данных. Без использования подобной специфики получение высоких коэффициентов ускорения в таких задачах представляется труднодостижимым.

Эффективное использование иерархии вычислительных уровней суперЭВМ

Хорошо известно, что принцип локальности — один из ключевых принципов организации высокопроизводительных вычислений. Используемая в настоящее время вычислительная среда, как правило, имеет следующую иерархическую структуру:

- территориально-распределенная Grid-среда;
- MPI-2-кластер с коммуникационной средой InfiniBand;
- иерархия архитектур SMP[NUMA] с несколькими многоядерными CPU;
- несколько ядер, имеющих общий кэш последнего уровня;
- специализированные ускорители [GPU, FPGA...].

Современный общепринятый стандарт MPI совершенно оправданно используется на базовом (кластерном) уровне суперкомпьютера, так как обеспечивает масштабируемость и переносимость параллельных программ. Однако использование MPI на верхнем уровне никак не отменяет возможности эффективно задействовать программные средства, доступные на остальных уровнях. Причина в том, что число вычислительных ядер на кристалле и число процессоров на системной плате имеют тенденцию расти *вместе с ростом производительности суперЭВМ*. Через несколько лет вполне реально ожидать чипы с сотнями и более вычислительных ядер. Наличие объемной общей кэш-памяти у всех этих ядер делает актуальной задачу эффективного задействования параллелизма *внутри многоядерных процессоров*. Тем более, что стандарт MPI давно предусматривает несколько моделей работы вычислительных потоков с коммуникационной средой.

Необходимо также учитывать современную тенденцию к использованию специальных вычислительных ускорителей типа GPU или FPGA. Эффективной

"боевой единицей" для кластерного уровня становится связка "многоядерные CPU + спецвычислители".

В данной работе была использована простая программная прослойка *SCALL — масштабируемый вызов* или *супервызов*. Такая прослойка упрощает распараллеливание Фортран-программ и позволяет максимально использовать для расчетов возможности наиболее низких уровней, таких как многоядерные процессоры и ускорители. При этом она хорошо сочетается с существующими реализациями MPI типа OpenMPI и не требует для работы никаких дополнительных библиотек.

Абстракция супервызова в целом не меняет языковой модели вычислений и выглядит для прикладного программиста как новая возможность параллельного запуска на счет произвольного числа *заданий*. Каждое задание можно разделить на произвольное число фрагментов, которые смогут выполняться параллельно.

Например, вместо вызова подпрограммы CALL sthd102n можно написать CALL scall(sthd102n,12). Здесь осуществляется *супервызов* той же самой процедуры, а также дается указание, что можно задействовать *до двенадцати счетных ядер суперЭВМ* для вычисления ее *двенадцати независимых фрагментов*. В случае если необходимо распараллеливание на все доступные ядра, вместо положительного значения можно передать 0.

Парный вызов swait служит для ожидания готовности результата работы, который вызываемая Фортран-процедура обычно оставляет в common-блоках. Вызов swait вовсе не обязан следовать непосредственно за вызовом scall (в таких случаях можно использовать ожидающий супервызов scallw). Другими словами, основная программа запускает параллельно работающие *задания*, но ее управляющая логика в целом остается прежней, за исключением того, что она может параллельно заниматься подготовкой других заданий. При этом периодически приходится выполнять вызовы swait. Отметим, что если указанное задание еще не досчитано до конца, то основной поток также принимает участие в вычислениях (выступает в роли еще одного вычислительного ядра). Таким образом, данный подход совместим с привычной моделью последовательных вычислений для традиционной однопроцессорной ЭВМ. Это обстоятельство позволяет эффективно вести отладку и контролировать корректность распараллеленного кода. В целом при таком подходе происходит динамическое распараллеливание, а именно — по мере выполнения частей работы фоновые вычислительные потоки подхватывают следующие фрагменты доступных заданий и так далее.

При первом обращении к scall происходит инспектирование аппаратных возможностей платформы, чтобы общее число дополнительных порожденных потоков не превысило количество физических счетных единиц в системе. В частности, для двухъядерной системы порождается всего один фоновый аппаратный поток, а для двенадцатиядерной — 11 фоновых потоков.

Для разделения на фрагменты подпрограмму sthd102n необходимо модифицировать следующим образом.

- В определении вызываемой процедуры следует указать два аргумента, которые автоматически передаются при использовании супервызова

```
SUBROUTINE sthd102n(isc, nsc)
```

- Аргументы `isc` и `nsc` принимают значения индекса потока в группе данного задания и общее количество фрагментов в задании (аналогичны MPI-понятиям rank и size).
- Если вызываемая процедура содержит цикл по всей области (thinmax1 соответствует числу расчетных ячеек), то оригинальный заголовок конструкции полного цикла

```
DO i = 1, thinmax1
```

заменяется на заголовок цикла для выполнения фрагмента работы, соответствующей индексу потока в группе:

```
CALL ibounds(isc, nsc, ib, 1,thinmax1)
DO i = ib(1), ib(2)
```

Здесь использована подпрограмма-утилита `ibounds`, осуществляющая выделение из общего количества шагов цикла соответствующего фрагмента. Она возвращает пару целых чисел в массив из двух целых чисел:

```
INTEGER*4 ib(2)
```

Наблюдаемое ускорение переделанной процедуры зачастую линейно для небольшого числа ядер даже для сверхлегких гранул протяженностью в десятки микросекунд.

Специализированные матричные алгоритмы и их программные реализации

При использовании полунеявной вычислительной схемы возникает необходимость решать систему линейных алгебраических уравнений на каждой итерации. В контексте данной задачи оригинальный алгоритм расходовал до четверти общего времени, что было вполне приемлемо. Однако, когда задействуется большое число процессорных ядер, время решения СЛАУ начинает быстро превышать совокупное время счета остальных частей, и это обстоятельство становится главным узким местом. Таким образом, необходимо эффективно распараллелить стадию решения СЛАУ.

Оптимизация решений систем линейных уравнений с разреженными матрицами задействует куда более интеллектуальные и сложные алгоритмы, чем в случае плотных матриц. Этот факт обусловлен тем, что сильно неоднородная структура коэффициентов допускает такие эффективные трансформации и декомпозиции на подзадачи, которые просто не имеет смысла делать для плотных матриц.

Обычно такие алгоритмы состоят из следующих четырех основных шагов:

- переупорядочивание строк и столбцов матрицы с целью сократить дальнейший счет, а в частных случаях — свести матрицу к блочно-треугольной;
- анализ или символьная факторизация, определяющие ненулевые структуры разложения;
- численная факторизация, вычисляющая L и U (для наиболее общего случая LU-разложения, т. е. представленная в виде произведения нижнетреугольной и верхнетреугольной матриц);
- собственно решение системы, прямая и обратные подстановки по известному LU-разложению.

На настоящее время существует большое число реализаций подобных алгоритмов, в том числе ориентированных на использование MPI-кластеров. При этом необходимо отметить, что не существует реализации, которая была бы эффективнее других в скорости счета во всех случаях. На разных классах конфигурации разреженных матриц лидирующие позиции могут занимать разные алгоритмы.

В частности, интерес представляют следующие реализации, способные эффективно задействовать современные мультикомпьютеры:

```
DSCPACK; PaStiX; SuperLU_DIST; S+; WSMP;
PARDISO; UMFPACK.
```

Для их тестирования были написаны программы на основе примеров использования модулей dlinsolx2 и pddrive2_ABglobal, которые прилагаются к исходному коду библиотек SuperLU и SuperLU_DIST. Схема работы программ выглядит следующим образом.

1. Загрузка матрицы, правой части и эталонного решения разреженной линейной системы. При этом используются как актуальные для данного кода матрицы, так и матрицы, возникающие в смежных задачах моделирования теплогидравлики контуров АЭС.

2. Решение линейной системы. Вывод нормы разности найденного и известного решений системы. При первом решении системы в памяти сохраняются данные о структуре разреженной матрицы. С помощью анализа этих данных на следующих итерациях, согласно логики работы библиотек типа SuperLU, повышается производительность вычислений.

3. Решение линейной системы необходимое число раз с сохранением времени выполнения основной процедуры решения.

4. Вывод минимального (Min), среднего (Avg) и максимального (Max) времени решения.

Наименьшее время счета было получено на восьми процессорах (табл. 1). Ускорение относительно времени счета на одном процессоре составляет около 40 %.

На восьми процессорах ускорение относительно времени счета на одном процессоре у PaStiX получается более 300 % (табл. 2). Наименьшее минимальное время счета (около 4 мс) было получено на 16 процессорах.

Таблица 1.

Результаты тестирования библиотеки SuperLU_DIST (матрица размером > 2000)

Число процессоров/узлы	Время счета, мкс		
	Min	Avg	Max
1/1	33 145	33 789	34 601
2/2	28 132	28 454	29 020
2/1	30 146	30 425	30 804
4/4	25 942	26 362	26 786
4/2	25 808	26 057	26 554
4/1	30 504	30 792	31 484
8/8	**23 559**	**23 912**	**25 047**
8/4	23 960	24 356	25 328
8/2	26 570	26 804	27 799
8/1	36 331	37 009	38 009

Таблица 2

Результаты тестирования библиотеки PaStiX (матрица размером > 2000)

Число процессоров	Время счета, мкс		
	Min	Avg	Max
1	38 264	40 336	48 815
2	19 110	25 251	30 076
3	13 578	15 579	20 666
4	10 946	14 286	16 765
5	9593	12 292	16 198
6	11 614	13 098	19 717
7	10 923	11 407	17 554
8	9318	9522	16 327
9	9011	11 598	18 717
10	6476	9206	15 675
11	7530	10 825	52 775
12	7015	11 272	24 916
13	6098	14 858	67 155
14	5339	15 159	84 159
15	5208	19 356	106 447
16	**3913**	**18 338**	**171 934**

Автоконфигурируемая библиотека SMatrix

Дальнейшая проработка и оптимизация решателя СЛАУ привела к созданию библиотеки SMatrix, которая позволяет:

- безболезненно подключать и заменять Open Source и коммерческие библиотеки, предназначенные для решения СЛАУ с разреженными матрицами без модификации расчетных кодов;
- осуществлять экспресс-тестирование корректности и скорости работы различных алгоритмов на актуальных матрицах, которые могут меняться на разных стадиях и входных данных даже в пределах одного расчетного кода;
- автоматически выбирать (после обучающего прогона) наиболее быстрый из доступных (подключенных при сборке) алгоритмов для каждой конкретной конфигурации матрицы.

Программный интерфейс SMatrix в основном состоит из трех функций, а именно для задания конфигурации ненулевых элементов матрицы, решения серии СЛАУ и освобождения внутренних ресурсов после работы алгоритма:

```
smatcfg_t smatrix_new(int n, int nnz, const
int* ia, const int* ja);

void smatrix_solve(smatcfg_t, const real*
a, int k, const real *b, real *x);

void smatrix_delete(smatcfg_t);
```

Подключаемый модуль (драйвер конкретного алгоритма) должен предоставить структуру следующего вида:

```
typedef struct alg_str {
   str7_t name, *opts;
   int (*max_n)(), cached_max_n;
   void (*init_d)(smatcfg_t,const real* a);
   void (*solve)(smatcfg_t, const real* a,
      int k,const real* b, real *x);
   void (*cleanup)(smatcfg_t);
} *alg_t;
```

Поля структуры имеют следующий смысл:

`name` — краткое наименование алгоритма.

`opts` — перечисление возможных режимов работы алгоритма.

`max_n` — указатель на функцию, которая возвращает максимальный размер матрицы, для которого поддерживается работа алгоритма. Вызывается не более одного раза. Эта функция может вернуть 0, если данный алгоритм вообще не способен работать на данном оборудовании (например, не установлена лицензия на коммерческую библиотеку или отсутствует необходимый аппаратный ускоритель).

`init_d` — указатель на функцию инициализации внутренних структур данных алгоритма. Вызывается не более одного раза перед первым решением СЛАУ.

`solve` — указатель на функцию решения серии k систем с матрицей a и массивом из k правых частей.

`cleanup` — указатель на функцию освобождения ресурсов. Вызывается при окончании работы с данной конфигурацией матрицы.

Наилучшие времена на малой матрице (размером 122) показали библиотеки UMFPACK и PARDISO. Общий выигрыш во времени счета по отношению к оригинальному алгоритму достигал 500 %.

Далее в кратком изложении остановимся на особенностях используемой в настоящее время целевой аппаратной платформы.

Характеристики целевой аппаратной суперкомпьютерной платформы

СуперЭВМ, которая в контексте настоящей работы рассматривается как целевая (базовая), представляет собой *компактную* высокопроизводительную установку, разработанную РФЯЦ-ВНИИЭФ в рамках создания серии подобных систем. Данная СуперЭВМ способна совершать до триллиона операций в секунду.

Ключевым моментом для потенциальных потребителей этой установки является именно ее компактность и автономность. Конструктивно она представляет собой вариант, близкий к компьютеру персонального использования, который не требует для своей установки и сопровождения каких-либо особых условий, работает под управлением ОС Linux и имеет предустановленный набор программного обеспечения для поддержки высокопроизводительного счета.

Такими суперЭВМ можно оснащать практически любые категории организаций, включая тренажерные комплексы для персонала, небольшие исследовательские институты и конструкторские бюро.

Ниже приведен краткий список основных характеристик компактной суперЭВМ (рис. 1).

Рис. 1. Компактная суперЭВМ АПК-1

Теоретическая пиковая производительность	1,0 Тфлоп/с
Число процессорных ядер	144 шт.
Максимальный объем оперативной памяти	до 768 Гбайт
Емкость дисковой памяти	до 24 Тбайт
Операционная система	Linux
Акустический уровень шума	Менее 50 дБА
Габариты (В × Ш × Г)	650 мм × 325 мм × 725 мм
Вес	60 кг
Число материнских плат/процессоров на плате	3 шт./4—6 шт.
Система охлаждения процессоров	Жидкостная
Система межпроцессорных обменов	InfiniBand QDR
Сеть управления и мониторинга	Ethernet
Подключение к локальной сети предприятия	Gigabit Ethernet
Система шумоподавления	Пассивная
Электропитание	220 В, 50 Гц
Потребляемая мощность	Не более 2,2 кВт
Стоимость (в зависимости от комплектации)	от 1,6 млн руб.

В настоящее время ведутся испытания опытного образца следующей модели серии компактных суперЭВМ (рис. 2). Его основные характеристики приведены ниже.

Пиковая производительность	3,0 Тфлоп/с
Число процессорных ядер	384 (24 × 16 ядер)
Оперативная память	768 Гбайт
Дисковая память	108 Тбайт
Стоимость	4,0 млн руб.

Рис. 2. Перспективная модель компактной суперЭВМ

Промежуточные и прогнозируемые результаты

На этапе работ, результаты которого представлены в настоящей публикации, основное внимание было уделено выяснению потенциала для распараллеливания расчетного кода улучшенной оценки БАГИРА. Запуск отдельных модулей при этом проводился зачастую в изолированном режиме, так как это позволяло быстрее оценить ситуацию и возможности для ускорения каждой стадии расчетного цикла.

Уже на ранней стадии работ было зафиксировано ускорение в режиме неполного согласования по данным пропорционально небольшому числу используемых ядер. После этого основное внимание было уделено ускорению стадии решения СЛАУ, поскольку именно она становилась основным узким местом, до устранения которого на высокий итоговый коэффициент ускорения нельзя было бы рассчитывать в принципе. Напомним, что в последовательной версии решение СЛАУ занимает около четверти общего времени счета.

После серьезного анализа существующих методов и практической проработки различных подходов удалось существенно ускорить решение СЛАУ и поднять прогнозируемую верхнюю границу достижимого итогового коэффициента ускорения вычислений.

Принимая во внимание полученный на первом этапе результат, не следует забывать, что достижение высоких коэффициентов ускорения всегда сопряжено с трудностями ускорения несущественных на первый взгляд операций. Такие операции, как правило, не принимаются во внимание при первичном анализе кода, однако они начинают занимать заметную долю времени счета именно после успешного распараллеливания наиболее ресурсоемких подзадач. По этой причине при дальнейшем решении поставленной задачи могут потребоваться еще более значительные усилия для того, чтобы реализовать на практике весь вскрытый потенциал для распараллеливания расчетного кода "БАГИРА" для улучшенной оценки теплогидравлических характеристик АЭС с ВВЭР.

Заключение
Перспективы дальнейшей адаптации расчетного комплекса к суперЭВМ

Проведенные на данном этапе работы являются лишь первым шагом к созданию высокопроизводительного вычислительного комплекса "БАГИРА" для улучшенной оценки характеристик теплогидравлических процессов в АЭС.

В настоящее время проводятся работы по дальнейшей оптимизации кода расчетного комплекса, которые позволят максимально задействовать массовый параллелизм, присущий существующим и перспективным архитектурам суперЭВМ. Такие ЭВМ могут состоять не только из традиционных CPU, но также включать специализированные аппаратные ускорители.

Поддержка массового параллелизма во время счета открывает возможности для дальнейшего увеличения числа расчетных ячеек, что может положительно сказаться на точности решения. При этом следует иметь в виду, что применение суперкомпьютеров, как правило, дает тем больший выигрыш, чем более сложную задачу они решают.

Необходимо отметить еще один качественно новый возможный результат от применения суперЭВМ применительно к рассматриваемой предметной области. Он заключается в том, что ускоренное решение обратных задач, в данном случае поиск оптимальных значений тех или иных параметров теплогидравлических процессов, востребовано при проектировании АЭС. При наличии большого числа вычислительных ядер можно запускать параллельно несколько независимых процессов моделирования и на основе полученных результатов модифицировать параметры проектируемой установки.

Список литературы

1. **Mazumdar M., Marshall J. A., Awate P. A., Chay S. C., McLain D. K.** Review of the metodology for statistical evaluation of reactor safety analyses // NASA STI/Recon Technical Report N, 1975. Vol. 76.

2. **Крошилин А. Е., Крошилин В. Е., Смирнов А. В.** Численное исследование трехмерных течений пароводяной смеси в корпусе парогенератора ПГВ-1000 // Теплоэнергетика. 2008. № 5. С. 12—19.

3. **Крошилин А. Е., Крошилин В. Е., Смирнов А. В.** Применение программного комплекса БАГИРА для расчетного анализа аварийного режима с малой течью из I контура на стенде ПСБ-ВВЭР // Теплоэнергетика. 2006. № 9. С. 41—48.

4. **Веселовский А. Н., Животягин А. Ф., Крошилин А. Е., Крошилин В. Е.** Полномасштабные тренажеры для АЭС на базе программного комплекса БАГИРА // Теплоэнергетика. 1999. № 6. С. 38—44.

5. **Макаров В. Л., Бахтизин А. Р., Васенин В. А., Роганов В. А., Трифонов И. А.** Средства суперкомпьютерных систем для работы с агент-ориентированными моделями // Программная инженерия. 2011. № 6. С. 2—14.

ИНФОРМАЦИЯ

Уважаемые коллеги!

Приглашаем Вас к авторскому участию в журнале "Программная инженерия"!

МАТЕРИАЛЫ, ПРЕДСТАВЛЯЕМЫЕ В РЕДАКЦИЮ

- Статья, оформленная в соответствии с требованиями.
- Иллюстрации и перечень подрисуночных подписей.
- ФИО авторов, название статьи, аннотация и ключевые слова на английском языке.
- Сведения об авторах (ФИО, ученая степень, место работы, занимаемая должность, контактные телефоны, e-mail).

ПОРЯДОК ОФОРМЛЕНИЯ ИНФОРМАЦИОННОЙ ЧАСТИ

- Индекс ***УДК*** размещается в левом верхнем углу первой страницы.
- ***Сведения об авторах*** на русском языке размещаются перед названием статьи и включают инициалы и фамилию авторов с указанием их ученой степени, звания, должности и названия организации и места ее расположения (если это не следует из ее названия). Указывается также e-mail и/или почтовый адрес хотя бы одного автора или организации.
- За *сведениями об авторах* следует ***название статьи***.
- После названия статьи отдельным абзацем дается ***краткая аннотация***, отражающая содержание статьи (что в ней рассмотрено, приведено, обосновано, предложено и т.д.).
- *Затем с*ледуют ***ключевые слова***.

ТЕКСТ СТАТЬИ

Статью рекомендуется разбить на разделы с названиями, отражающими их содержание.

Рекомендуемый объем статьи 15 страниц текста, набранного на стандартных листах формата А4 с полями не менее 2,5 см шрифтом размером 14 pt с полуторным межстрочным интервалом, с использованием компьютерного текстового редактора Word. В указанный объем статьи включаются приложения, список литературы, таблицы и рисунки. Страницы статьи должны быть ***пронумерованы***. В необходимых случаях по решению главного редактора или руководства издательства объем статьи может быть увеличен.

В формулах русские и греческие буквы следует набирать прямо; латинские буквы, обозначающие скаляры, *курсивом*; величины, обозначающие векторы и матрицы, должны быть выделены полужирным шрифтом и набраны прямо (допускается также набор всех величин, обозначенных латинскими буквами, в т.ч. матриц и векторов, светлым курсивом). Стандартные математические обозначения (например, log, sin и т.д.) должны быть набраны прямо.

Номера формул располагаются справа в круглых скобках. Нумерация формул — сквозная, причем нумеруются те формулы, на которые имеются ссылки.

Рисунки должны быть выполнены качественно (графическая обработка рисунков в редакции не предполагается). В журнале все рисунки воспроизводятся в черно-белом варианте, за исключением цветных рисунков, размещаемых по усмотрению редакции на обложке.

Статья может быть отправлена по e-mail: prin@novtex.ru.

Дополнительные пояснения авторы могут получить в редакции журнала лично, по телефонам: (499) 269-53-97, 269-55-10, либо по e-mail

УДК 004.383.4

Г. С. Речистов, млад. науч. сотр., e-mail: grigory.rechistov@phystech.edu,
А. А. Иванов, науч. сотр.,
П. Л. Шишпор, млад. науч. сотр.,
В. М. Пентковский, д-р техн. наук, руководитель лаборатории,
Лаборатория суперкомпьютерных технологий для биомедицины,
фармакологии и малоразмерных структур,
Московский физико-технический институт

Моделирование компьютерного кластера на распределенном симуляторе. Валидация моделей вычислительных узлов и сети

Демонстрируется подход к задаче моделирования многопроцессорного кластера, состоящего из нескольких многоядерных компьютеров, соединенных высокопроизводительной сетью. Описываются используемый симулятор, окружение для проведения тестов и методы проведения измерений. Представлены результаты валидации моделей отдельных вычислительных узлов и сети, их соединяющей; предложены решения для преодоления обнаруженных ограничений и пути увеличения точности симуляции.

Ключевые слова: *распределенная симуляция, Simics, многоядерные системы, производительность симуляции, масштабируемость, кластер, Linpack*

Введение

Расчеты, необходимые для современного научного исследования, могут занять неприемлемо большое время если проводить их на единственной ЭВМ, пусть даже максимально мощной и многопроцессорной. Другая опасность состоит в том, что не хватит емкости оперативной памяти или иных ресурсов для того, чтобы вместить все необходимые для работы данные. В таких случаях используется группа компьютеров, соединенных сетью, при этом задача разбивается и распределяется по этим узлам. Такая организация вычислений получила общепринятое название "кластер".

Перед создателями нового кластера зачастую стоит задача определения его оптимальной конфигурации, позволяющей достигнуть наибольшей производительности. При этом необходимо учитывать, какие конкретно приложения будут запускаться на этом кластере — они напрямую определяют требования на оборудование.

Для предсказания значений производительности создаваемой компьютерной системы удобной методикой является симуляция — моделирование еще не существующей аппаратуры с помощью программы, выполняющейся на уже доступных компьютерах. Данный подход используется на всех этапах разработки — от первых спецификаций и описания набора инструкций отдельного процессора до полной системы с работающей операционной системой и прикладными приложениями. Это позволяет обнаруживать ошибки проектирования на ранних этапах, снижая цену их исправления. Точность моделирования может варьироваться от высокой, демонстрируемой на сверхточных потактовых симуляторах, обязанных показывать поведение, неотличимое на уровне отдельных тактов от поведе-

ния настоящей системы, до достаточной в функциональных моделях, тем не менее способных загружать операционные системы и пользовательские приложения и при этом работать достаточно быстро. При симуляции многопроцессорных систем мы наталкиваемся на ряд следующих серьезных препятствий, обусловленных масштабами задачи:

- при последовательной симуляции всех моделируемых процессоров на одном реальном скорость работы будет ничтожно мала, поэтому необходимо использовать параллельные системы, что, в свою очередь, требует сложных схем синхронизации;
- ресурсов одной машины также может не хватить для содержания в себе модели целого кластера, поэтому модель сама по себе должна быть распределенной, т. е. выполняться на кластере.

Изучению и увеличению производительности приложений, использующих различные парадигмы многопроцессорных вычислений, посвящено множество работ. Описанию парадигмы передачи сообщений, такой как MPI, посвящены работы [1] и [2]; описанию систем с общей памятью и использующих OpenMP — работа [3]. Существуют также попытки создать адаптивные или гибридные системы, использующие лучшее из обоих подходов [4]. Моделированию суперкомпьютеров еще до их построения "в железе" посвящено несколько работ. Из них можно отметить BigSim [5] — симулятор IBM Blue Gene, а также MPI-SIM [6].

Данная работа описывает характеристики кластера, устанавливаемого для задач вычислительной биологии, симуляционную модель, построенную для анализа узких мест в его производительности и результаты измерений, выполненные на этой модели.

Подход к моделированию кластера

Работа, описанная в данной статье, является лишь одним из этапов многоступенчатого плана: на существующем оборудовании создается модель будущего кластера, характеристики которого превосходят характеристики текущего в десятки раз. Когда будущий кластер воплощается в реальность, уже на нем строится модель следующего поколения. Это позволяет своевременно учитывать тенденции в развитии аппаратного обеспечения и уменьшить нагрузку на разработчиков модели.

Характеристики моделируемого кластера

На рис. 1 приведена схема кластера, модель которого описана в данной работе. В табл. 1 приведены некоторые характеристики составляющих его компонентов.

Вычисления организованы следующим образом. Программа пользователя запускается на головном узле, затем она распределяется по вычислительным узлам, где и проводится большая часть вычислительной работы. Головной узел предоставляет сервисы координации общего хода работы, а также дисковое хранилище, доступное по протоколу NFS. Вычислительные узлы не имеют своего хранилища.

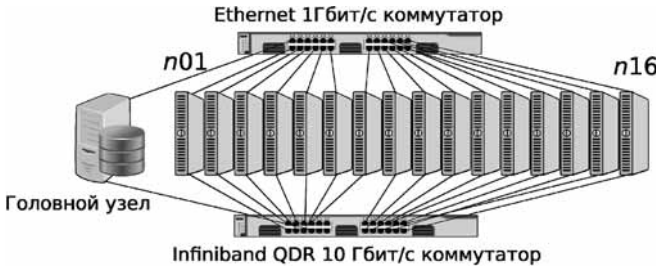


Рис. 1. Схема кластера:
*n*01—*n*16 — вычислительные узлы кластера

Таблица 1

Параметры кластера

Параметр	Значение
Число вычислительных узлов	16
Процессоры узлов	Два Intel Xeon 5580, 3,33 ГГц, суммарно 12 ядер
Объем памяти головного узла	48 Гбайт
Дисковое хранилище головного узла	3 Тбайт
ОЗУ каждого вычислительного узла	32 Гбайт DDR3
Полное число вычислительных ядер	192
Полный объем ОЗУ для вычислений	512 Гбайт

Используемый симулятор

Для построения модели был использован симулятор Simics [7]. Выбор был обусловлен его следующими достоинствами: полносистемная симуляция, учитывающая все аспекты работы как отдельного компьютера, так и системы связанных между собой узлов; высокая скорость функциональных моделей систем; возможность балансировать между скоростью работы и точностью во время симуляции; возможность сбора трасс событий для последующего детального изучения фактов, влияющих на производительность; множество доступных моделей устройств в поставке, позволяющих быстро создавать прототипы систем; легкость разработки и подключения новых моделей устройств — от центральных процессоров до сетевых карт, систем кэшей и криптографических ускорителей; наличие средств автоматизации процесса симуляции измерений с помощью сценариев.

Скорость и масштабируемость симуляции в Simics обеспечивается следующими возможностями:

- использованием двоичной трансляции [8] и аппаратных расширений виртуализации Intel VTx для достижения максимальной скорости работы (загрузка операционной системы внутри Simics может замедляться менее чем в 10 раз по сравнению с обычным ее исполнением);
- многопоточной симуляцией процессоров, позволяющей полностью использовать ресурсы ЭВМ;

- запуском программы в распределенном на несколько компьютеров режиме;
- технологией page sharing [9], используемой для экономии памяти.

Замечания о точности используемых моделей

К сожалению, требования точности соответствия процесса и результатов симуляции реальному процессу и скорости работы самой модели часто противоречат друг другу. Авторами начата разработка полуаналитического метода для корректировки результатов, полученных в настоящей модели. При использовании трасс исполнения и характеристик аппаратуры, измеренных при помощи инструмента Intel VTune [10] или потактовых симуляторов, авторы планируют получать достаточно точные оценки производительности моделируемых систем.

В настоящей работе концентрируется внимание на двух аспектах производительности полной модели — процессоров и сетевых соединений.

Прежде всего, рассмотрим следующие особенности устройства ядра процессора, влияющие на производительность, без учета прилежащих к нему систем кэшей и памяти:

- современные процессоры Intel имеют так называемую out-of-order архитектуру, т. е. машинные инструкции могут исполняться в порядке, отличном от того, какой присутствует в памяти программы.
- наличие конвейера и нескольких декодирующих и вычислительных устройств позволяет эффективно выполнять несколько команд за такт.
- длинные и сложные команды архитектуры IA-32 разбиваются на более мелкие микрокоманды, которые могут быть исполнены в порядке, зависящем от текущего состояния вычислительного тракта процессора.

Существуют решения, моделирующие указанные выше аспекты микроархитектуры [11]. Однако необходимость учитывать такие подробности существенно снижает скорость самого симулятора.

Модели процессоров в Simics не учитывают out-of-order характер исполнения — инструкции обрабатываются в том порядке, в котором они найдены в памяти. Кроме того, по умолчанию на исполнение одной инструкции тратится ровно один такт симулируемого времени, тогда как в реальности длительность исполнения инструкции зависит от множества факторов, в первую очередь от типа этой команды. Это называется "функциональным моделированием".

В симуляторе Simics есть конфигурационный параметр, определяющий сколько тактов симулируемого процессора занимает исполнение одной инструкции. По умолчанию этот параметр равен 1. Для увеличения точности симулятора необходимо добавлять к системе отдельный модуль, определяющий задержки в циклах для каждой инструкции. Однако даже при использовании этого модуля вопрос об учете внеочередного исполнения команд остается открытым.

Последствия этого обстоятельства на результаты симуляции будут показаны далее, в секции результатов.

Simics предоставляет модели сетевых карт Ethernet для использования их в составе моделей полных вычислительных узлов. Устройства 1/10 Gigabit Ethernet поддерживаются используемой на модели кластера операционной системой GNU/Linux Debian 6, а также задействованы на реальной системе.

В Simics также есть простая модель сетевого коммутатора (*switch*), которая характеризуется значением задержки при передаче пакета и пропускной способностью. В модели также поддерживается режим с "бесконечной" пропускной способностью, в которой пакеты теряются лишь если они пришли в сеть одновременно с точностью до такта микропроцессора. Именно этот режим и использовался в экспериментах с целью определить оптимальную задержку и пропускную способность сетевого коммутатора.

Тестовые задачи

Задачей данной работы являлось построение модели кластера и апробация методик измерения его производительности. Для этого был проведен ряд измерений на настоящей аппаратуре и на модели. Были использованы две следующие программы.

- *High Performance Linpack* [12] (далее Linpack). Данный тест, решающий систему линейных уравнений методом Гаусса, широко используется для оценки производительности систем на вычислениях с плавающей точкой. Кроме того, результаты, показанные на этом тесте, используются для составления списка Top 500 [13] суперкомпьютеров. В работе [14] подробно рассмотрены существующие варианты Linpack для архитектуры IA-32. В работе [15] описан алгоритм, используемый в данном тесте для решения системы уравнений.
- *Netperfmeter* [16]. Приложение для измерения скорости соединения по протоколам TCP, UDP и SCTP для Unix-систем. Позволяет задать сценарий, согласно которому в некоторые моменты времени будут создаваться и закрываться соединения. В конце работы приложение выводит статистику для прошедшего трафика. В этой работе программа использовалась для оценки максимальной скорости передачи данных между двумя компьютерами.

Результаты измерений производительности узла на тесте Linpack

Данный раздел описывает результаты, полученные при верификации модели узла кластера, проведенной посредством сравнения результатов теста Linpack, полученных на реальной машине и на модели узла кластера. Отметим, что для достижения максимальной производительности на тесте Linpack необходима перекомпиляция этого теста для той машины, на которой будет проводиться запуск. Дело в том, что при компиляции тест оптимизируется в соответствии с ха-

рактеристиками используемой машины. Описанная процедура по сборке была проведена при запуске теста Linpack как на реальной машине, так и на симулируемой. Конфигурационный файл теста был использован один и тот же при всех запусках.

Linpack запускался в двух конфигурациях — на одном и на четырех ядрах. Значения размеров задачи N брались в диапазоне от 100 до 32 000. При бóльших размерах задачи Linpack сообщал о невозможности выделить блок памяти нужного размера.

Реальная ЭВМ. Измерения проводились на системе со следующими характеристиками: процессор — Intel (R) Xeon (R) 5150 2,66 ГГц, 4 ядра, 16 Гбайт ОЗУ, операционная система Red Hat Enterprise Linux Server release 5.4 (x86_64).

На рис. 2 приведены графики зависимости производительности Linpack от размера задачи для одного и четырех ядер. В табл. 2 приведены значения четырех параметров для запусков: $P_{\max}$ — наибольшая продемонстрированная производительность, измеренная в гигафлопсах (флопс от англ. "*floating point operations per second*"); $N_{\max}$ — размер задачи, при которой достигнута $P_{\max}$; $N_{1/2}$ — размер задачи, при котором достигается половина от $P_{\max}$; P_{peak} — пиковая, теоретическая производительность для машины, равная сумме числа инструкций над числами двойной точности на всех устройствах всех ядер системы, выполняемых за секунду работы. Также приведено значение эффективности системы, равное отношению максимальной производительности к теоретической: $P_{\max}/P_{peak}$.

Моделируемая система. Аналогичная зависимость была получена на модели вычислительного узла кластера. Измерения проводились на системе со следующими характеристиками: процессор Intel (R) Core i7 2,66 ГГц, 4 ядра, ОЗУ 16 Гбайт, ОС Debian 6.0.2 (64-битная).

На рис. 3 приведены графики зависимости производительности Linpack внутри симулятора от размера задачи для одного и четырех моделируемых ядер. В табл. 3 приведены значения $P_{\max}$, $N_{\max}$, $N_{1/2}$, P_{peak} и эффективности для модели.

Анализ результатов. Полученные значения пиковой производительности для симулятора в четыре раза меньше аналогичных величин для реальной ЭВМ. Как уже было сказано, это вызвано различным значением числа команд, исполняемых за такт.

Другим интересным фактом является то, что достигнутая эффективность на симулируемой системе выше, чем на реальной: более 90 % против 80. Это объясняется отсутствием задержек, вызванных промахами системы кэшей — все доступы к памяти в модели всегда завершаются за один такт. В реальности это соответствовало бы системе, имеющей все свои данные

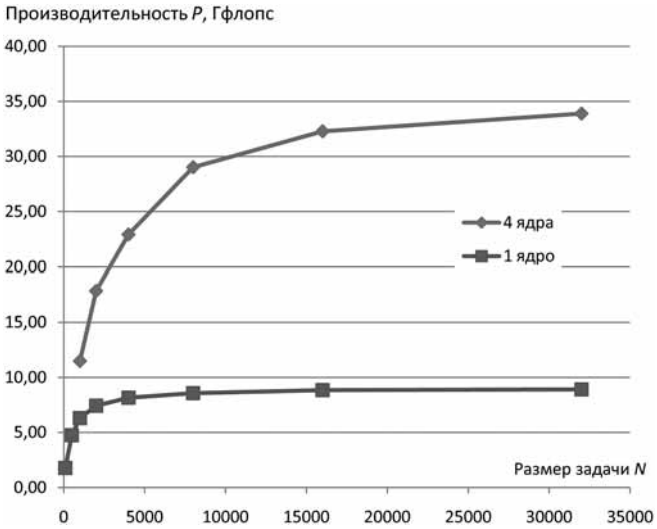


Рис. 2. Измерения Linpack на реальной машине

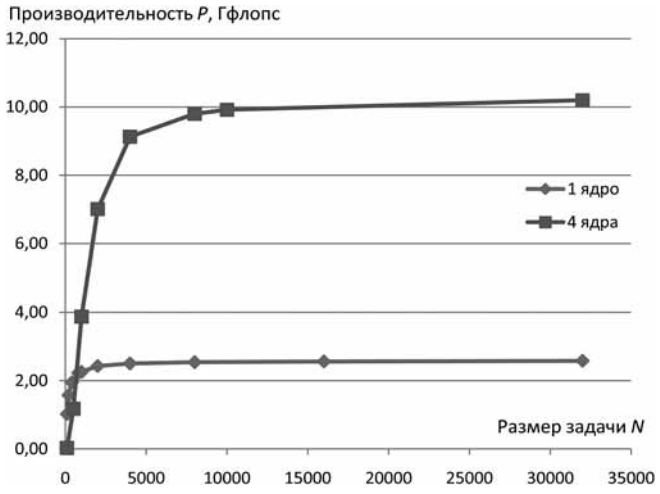


Рис. 3. Измерения Linpack внутри симулятора

Таблица 2

Результаты прогона Linpack на реальной машине

Величина	Одно ядро	Четыре ядра
$P_{\max}$, Гфлопс	8,91	33,9
$N_{\max}$	32 000	32 000
$N_{1/2}$	400	1800
P_{peak}, Гфлопс	10,64	42,56
Эффективность, %	83,74	79,65

Таблица 3

Результаты прогона Linpack на симулируемой машине

Величина	Одно ядро	Четыре ядра
$P_{\max}$, Гфлопс	2,58	10,1
$N_{\max}$	32 000	32 000
$N_{1/2}$	150	1200
P_{peak}, Гфлопс	2,66	10,64
Эффективность, %	97,00	94,92

(для $N = 32\,000$ это около 7,6 Гбайт) в кэше первого уровня. Таким образом, модель дает представление о верхней границе производительности для реальной системы с очень эффективным кэшом или очень быстрой памятью.

Построенная модель позволяет сравнивать между собой эффективность различных конфигураций, однако непосредственное сравнение реальных систем с симулируемыми требует пересчета результатов, учитывающего различное значение отношения "инструкций за такт".

Результаты измерений на тесте Netperfmeter

Для тестирования использовались две машины (реальные в одном случае и симулируемые в другом). Командная строка запуска на активной стороне канала:

```
netperfmeter remotehost: 9000 –runtime 60
–tcp const0:exp2000:const0:exp2000:.
```

Это соответствует TCP-соединению со следующими характеристиками: полная длительность приема и передачи — 60 с, исходящее соединение — неограниченная (программно) скорость, исходящий поток имеет обратно-экспоненциальное распределение размеров пакетов со средним значением 2000 байт и максимальным 16 000 байт; входящий поток имеет аналогичные характеристики.

Реальная система. В физической системе были установлены сетевые карты Intel PRO/1000 PCI Gigabit Ethernet. Они были соединены через коммутатор с пропускной способностью 1 Гбит/с. Были получены следующие результаты:

- исходящий канал: скорость передачи данных 137 Мбайт/с;
- входящий канал: скорость передачи данных 110 Мбайт/с;
- потери: 267 кбайт/с.

Моделируемая система. В симулируемой системе использовалась модель сетевой карты Intel PRO/1000 PCI-Express Gigabit Ethernet. Полученные результаты:

- исходящий канал: скорость передачи данных 18 Мбайт/с;
- входящий канал: скорость передачи данных 8 Мбайт/с;
- потери: 19 кбайт/с.

Анализ результатов. Результаты тестирования соединения между реальными машинами показывают значения, близкие к теоретическому пределу (1000 Мбит/с = = 125 Мбайт/с). Это означает, что Netperfmeter может быть использован для адекватной оценки соединений.

В момент проведения тестов значение задержки сетевых пакетов было равно 400 мкс. Таким образом, для пакетов длиной 2000 байт эффективная пропускная способность равна приблизительно один пакет на один акт обмена, т. е. $2 \cdot 10^3$ байт/$4 \cdot 10^{-4}$ с = $5 \cdot 10^6$ байт/с = = 4,8 Мбайт/с, что по порядку величины совпадает с полученными ранее результатами.

Эти данные показали, что сетевая подсистема требовала дополнительной наладки. При исследовании причин низкой производительности оказалось, что максимально разрешенный размер пакета был равен лишь 2000 байтам, что существенно меньше возможностей гигабитной сетевой карты. Это обстоятельство было исправлено и последующие измерения работы Linpack проводились уже с большими пакетами.

Результаты измерения производительности сети на тесте Linpack

В данном разделе описано измерение нагрузки на сетевой коммутатор в моделируемом кластере, а также влияние времени задержки пакетов на общее время исполнения Linpack. Измерения проводились на последовательных запусках Linpack с размером матрицы $N = 4000$ и параметрами распараллеливания задачи $P \times Q$ [12] 1×192, 6×132 и 16×12. Значение N было выбрано максимально большим в условиях приемлемого времени исполнения тестов. $P \times Q$, равный 1×192, соответствует сетевому потоку, обеспечивающему минимальную нагрузку на коммутатор. Числа 6×32 и 16×12 были выбраны в предположении возрастании нагрузки на сеть. Точного описания параметров P и Q в руководстве по Linpack найдено не было, но был обнаружен совет, предлагающий варьировать $P \times Q$, чтобы достичь максимальной производительности при фиксированных параметрах сети.

Для надежности измерений Linpack запускался в режиме множественных экспериментов, где один и тот же тест исполнялся 4 раза, и только результаты последних трех испытаний принимались во внимание.

Параметры сетевого потока измерялись при помощи утилиты Wireshark [17]. Во всех результатах приводится средняя нагрузка, поскольку сетевой поток захватывался сразу после начала эксперимента и останавливался точно к концу теста. График зависимости потока от времени показал, что для всех экспериментов пиковая нагрузка незначительно превышала среднюю. Результаты измерений приведены в табл. 4.

Влияние задержки на производительность Linpack. В Simics значение задержки сетевого пакета связано с механизмом синхронизации времени в моделируемых узлах. Опуская излишние детали, отметим, что при уменьшении задержки частота синхронизации симулируемого времени между узлами модели кластера

Таблица 4

Характеристики сетевого коммутатора при запуске теста Linpack на модели кластера

Параметры теста Linpack (при $N = 4000$) $P \times Q$	Нагрузка, Мбит/с	Средний размер пакета, байты	Производительность, Гфлопс
1×192	1300	5800	0,56
6×32	400	3700	1,1
16×12	150	1100	0,46

Таблица 5

Влияние задержки сетевого пакета на производительность теста Linpack в модели кластера

Параметры теста Linpack (при $N = 4000$) $P \times Q$	Производительность, Гфлопс, при задержке				
	80 мкс	160 мкс	240 мкс	320 мкс	400 мкс
1 × 192	0,56	0,55	0,56	0,57	0,56
6 × 32	1,1	1,1	1,1	1,1	1,1
16 × 12	4,6	4,6	4,6	4,5	4,5

возрастает. Это может привести к значительному замедлению симуляции. В связи с этим было важно проверить, насколько влияет малое время задержки сетевого пакета на сообщаемую Linpack производительность. В табл. 5 показаны результаты измерений для различных выбранных значений задержки.

Анализ результатов. Модель показала, что нагрузка на сеть при работе Linpack даже при наибольших наблюдавшихся размерах пакетов много меньше (на четыре порядка) возможностей QDR Infiniband-коммутатора (т. е. для Linpack топология сети приемлема). Заслуживает внимания факт, что наблюдается различный размер транзакций между узлами сети при разных значениях $P \times Q$. Необходимо изучить, будет ли возрастать размер пакетов и, следовательно, нагрузка на сеть с ростом размера матрицы при бóльших значениях параметра P теста Linpack.

Производительность Linpack не зависит от величины задержки сетевого пакета при значениях менее 400 мкс. Время обсчета очередного блока данных в Linpack больше времени сетевой транзакции, и это маскировало латентность сети. Этот факт позволил проводить все запуски Linpack для задержки сетевого пакета в 400 мкс, что значительно ускорило процесс симуляции.

Заключение

Результаты исследований, проведенных на отдельных узлах, а также на полной модели кластера, показали соответствие реальных и моделируемых характеристик ЭВМ и пригодность выбранного подхода для изучения характеристик его производительности на простой функциональной модели. При этом необходимо делать поправку на обнаруженные ограничения и разрабатывать метод коррекции результатов для получения достаточно точных значений производительности. Разрабатывается методология оценки производительности с использованием трасс симуляции и параметрами ЦПУ, памяти, полученными при помощи VTune или потактовых моделей, подключаются модели кэшей, уточняется набор трассируемых событий.

Большой практический интерес представляет изучение характера работы разных пользовательских приложений на модели кластера. В нашем случае это приложения молекулярной динамики. Выработанные на основе экспериментов с Linpack и Netperfmeter методики будут адаптированы к более сложным приложениям, основанным на пакете Gromacs [18].

Следует отметить, что нашей целью является не только исследование кластера в его текущей конфигурации, но и поиск оптимальных конфигураций с учетом его развития (в 2012 г. пиковая производительность кластера будет наращена в несколько раз путем увеличения числа входящих в него вычислительных узлов). Использование симулятора для оценок производительности выглядит многообещающе — результаты симуляции должны позволить вырабатывать рекомендации по оптимальному развитию кластера.

Список литературы

1. **Demaine E.** A threads-only MPI implementation for the development of parallel programs // Proc. of HPCS97. Winnipeg, Manitoba, Canada. 1997. P. 153—163.

2. **Riesen R.** A Hybrid MPI Simulator // Proc. of IEEE International Conference on Cluster Computing. Barcelona, Spain. 2006. P. 1—9.

3. **Hu Y. C., Honghui L., Cox A., Willy Z.** OpenMP for networks of SMPs // Journal of Parallel and Distributed Computing. 2000. Vol. 60. P. 1512—1530.

4. **Carribault P., Perache M., Jourdren H.** Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC // Proc. of the 6th International Workshop on OpenMP. Tsukuba, Japan. 2010. P. 80—93.

5. **Zheng G., Kakulapati G., Kale L. V.** BigSim: A parallel simulator for performance prediction of extremely large parallel machines // Proc. of Parallel and Distributed Processing Symposium. Santa Fe, New Mexico, USA. 2004. Vol. 1. P. 78.

6. **Prakash S., Bagrodia R. L.** MPI-SIM: using parallel simulation to evaluate MPI programs // Proc. of the Winter Simulation Conference. Washington DC. 1998. P. 467—474.

7. **Magnusson P.S., Christensson M., Eskilson J.** et al. Simics: A full system simulation platform // Computer. 2002. Vol. 35. P. 50—58.

8. **Sites R. L., Chernoff A., Kirket M. B.** et al. Binary translation // Communications of the ACM. 1993. Vol. 36. N 2. P. 69—81.

9. **Bugnion E., Devine S., Rosenblum M**. Disco: Running commodity operating systems on scalable multiprocessors // Proc. of the sixteenth ACM Symposium on operating systems principles. 1997. P. 143—156.

10. **Intel** Corporation. Intel® VTune Performance Analyzer 9.1 for Linux* — Documentation. URL: http://software.intel.com/en-us/articles/intel-vtune-performance-analyzer-for-linux-documentation.

11. **Matt Y. T.** PTLsim User's Guide and Reference. 2007. URL: http://www.ptlsim.org/Documentation/PTLsimManual.pdf.

12. **High** performance Linpack benchmark. URL: http://www.netlib.org/Linpack/.

13. **TOP500** supercomputing sites. URL: http://www.top500.org.

14. **Речистов Г. С.** Бенчмарк Linpack для архитектуры Intel IA-32: существующие варианты, сравнительный анализ возможностей и демонстрируемой производительности // Труды 54-й научной конференции МФТИ. Изд-во МФТИ, 2011. Т. 1. С. 71—72.

15. **Dongarra J. J.**, **Luszczek P., Petitet A.** The Linpack benchmark: Past, present, and future // Concurrency and Computation: Practice and Experience. 2003. Vol. 15. P. 20.

16. **Dreibholz T.** Netperfmeter: A TCP/UDP/SCTP/DCCP network performance meter tool. URL: http://www.iem.uni-due.de/~dreibh/netperfmeter/.

17. **Orebaugh A.**, **Ramirez G., Burke J.** Wireshark & Ethereal Network Protocol Analyzer Toolkit. Syngress Publishing, 2007. 540 p.

18. **Van Der Spoel D.,** Lindahl E. et al. Hess B. GROMACS: Fast, exible, and free // Journal of Computational Chemistry. 2005. Vol. 26. N. 16. P. 1701—1718.

УДК 004.94

В. А. Данилкин, аспирант, e-mail: danilkinva@gmail.com,
А. А. Трухачев, канд. техн. наук, доц., e-mail: AATrukhachev@mephi.ru,
А. Г. Бельтов, канд. техн. наук, доц., e-mail: beltov@mephi.ru,
НИЯУ "МИФИ", г. Москва

Построение модели перестроения транспортных средств в транспортных потоках

Рассмотрено расширение модели разумного водителя Трайбера на случай движения транспортных средств на многополосных магистралях. В отличие от ранее предлагаемых моделей описывается процесс перестроения в максимальном приближении к реальности. Это позволяет определить дополнительные задержки при смене полосы и рассчитывать на лучшее соответствие эмпирическим данным. Построенная модель была проверена в процессе имитационного моделирования. Полученные результаты хорошо соотносятся с данными о влиянии перестроений на параметры транспортного потока.

Ключевые слова: *транспортные потоки, модели перестроения транспортных средств, имитационное моделирование*

Введение

Общеизвестна статистика: за последние двадцать лет число личных автомобилей в России, приходящихся на тысячу человек населения, возросло более чем в 4 раза [1]. В результате столь динамичной автомобилизации в крупных областных центрах возник глобальный дефицит дорожного пространства. В таком городе как Москва дороги были спроектированы и построены из расчета максимального количества в 500 тыс. личных автомобилей, поэтому они, естественно, не справляются с имеющимся трехмиллионным потоком транспортных средств (ТС). На локальном уровне это оказывает влияние лишь на непосредственных участников дорожного движения, увеличивая их издержки на транспорт. Но достигнув такого масштаба, что большая часть крупных городов простаивает в пробках в течение целого дня, несоответствие существующего транспортного потоков и дорожной сети с каждым годом вносит все больший отрицательный вклад в экологию, экономику и социальное развитие страны. В силу междисциплинарности, отсутствия единого теоретического базиса и глубокой связанности с повседневной жизнью транспорта, наблюдение, измерение и моделирование являются важнейшими компонентами исследований для понимания тех процессов, которые протекают при движении транспортных потоков (ТП) [2, 3].

Микроскопическая модель разумного водителя Трайбера

При движении по прямой дороге влияние участников ТП друг на друга носит продольный характер. Ситуация меняется, когда происходит маневрирование (перестроение) ТС. Здесь необходимо, как минимум, учитывать действия водителей на смежных полосах для обеспечения безопасности движения. Важно отметить, что при значительном количестве разработанных микроскопических моделей поведения водителей на дороге, которые учитывают наличие других участников — модель оптимальной скорости Ньюэлла [4], модель следования за лидером Дженерал Моторс [5], модель разумного водителя Трайбера [10] и др. [6—9], модели смены полосы движения [11—18] описывают лишь условия и мотивацию водителей при

совершении перестроения. По мнению авторов, особый интерес представляет именно сам процесс перестроения, ведь именно вследствие того, что ТС меняет полосу за некоторое время Δt, и создаются задержки движения, например, при слиянии нескольких потоков в один в узком месте.

Решая задачу моделирования ТП на локальном участке улично-дорожной сети (УДС) важно использовать модели, максимально реалистично описывающие поведение водителей на дороге для минимизации расхождения с эмпирическими данными. Согласно работе [19], как показали численные эксперименты, наиболее "удачной" моделью из класса микроскопических является модель разумного водителя Трайбера. Согласно ей для ТС α, следующего в потоке за ТС $\alpha - 1$, ускорение определяется следующим соотношением:

$$\frac{dv_\alpha}{dt} = a\left[1 - \left(\frac{v_\alpha}{v_0}\right)^\delta - \left(\frac{s^*}{s_\alpha}\right)^2\right], \qquad (1)$$

где

$$s^* = s_0 + \max\left(v_\alpha T + \frac{v_\alpha(v_\alpha - v_{\alpha-1})}{2\sqrt{ab}}, 0\right),$$

$$s_\alpha = x_{\alpha-1}(t) - x_\alpha - l_\alpha,$$

x_α — одномерный вектор координат ТС α; v_α — скорость ТС α.

Параметрами этой модели являются:

a, b — комфортные ускорение и торможение соответственно;

v_0 — желаемая скорость ТС α;

T — время реакции водителя;

l_α — длина ТС;

$v_{\alpha-1}$ — скорость лидирующего ТС;

s_0 — минимальная дистанция до лидера при движении в заторе;

δ отвечает за поведение при разгоне, при $\delta = 0$ имеет место экспоненциальный по времени разгон, а при большом значении происходит "комфортное" увеличение скорости.

В дополнении к соотношению (1) запишем дифференциальное уравнение изменения положения ТС в пространстве в скалярном виде [4]:

$$\frac{dx_\alpha}{dt} = v_\alpha. \qquad (2)$$

Система уравнений (1) и (2) описывает скорость и положение ТС в однополосном ТП и хорошо применима для решения таких задач, как моделирование очередей перед светофорными объектами или эволюция затора.

Модель перестроения MOBIL

В работе [11] Трайбером были предложены условия, при выполнении которых происходит смена полосы ТС:

$$\tilde{a}_n \geqslant -b_{safe}, \qquad (3)$$

$$\tilde{a}_c - a_c + p(\tilde{a}_n - a_n + \tilde{a}_o - a_o) > \Delta a_{th}, \qquad (4)$$

где a — ускорение ТС; p — фактор вежливости водителей; Δa_{th} — значение порогового ускорения ТС.

Поясним условия (3) и (4) на рис. 1.

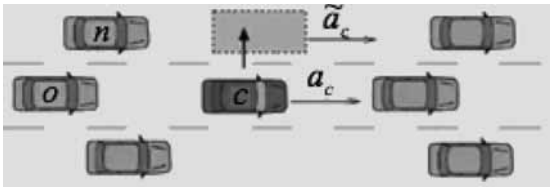


Рис. 1. Перестроение ТС

Тильдой обозначены ускорения ТС при наличии на их полосе перестроенного ТС, т. е. $\tilde{a}_n$ — ускорение ТС n при лидирующем ТС c. Первое неравенство показывает, что при перестроении ТС c перед ТС n, торможение ТС n должно превышать значения максимально безопасного торможения $-b_{safe}$. Таким образом, при выполнении неравенства (3) в случае перестроения ТС c, ТС n гарантированно не врежется в нового лидера. Второе неравенство, согласно работе [5] называется "критерием заинтересованности". При его выполнении движение ТС c на новой полосе более выгодно, чем на текущей. Также условие (4) учитывает влияние ТС n и o на перестроение ТС c.

Модель разумного водителя вместе с условиями перестроения образуют математический аппарат с помощью которого можно решать широкий спектр задач по моделированию движения ТП на многополосной дороге. Однако допущение о мгновенности смены полосы ТС может создать сильное расхождение с реальными данными. Для определения задержек, возникающих при смене полосы, рассмотрим физику процесса перестроения.

Расширение модели перестроения

Будем считать, что ТС совершает прямолинейное движение по правой полосе двухполосной дороги. Также примем за допущение, что

$$W = w + \Delta w, \qquad (5)$$

где W — ширина полосы, w — ширина ТС, Δw — расстояние, гарантирующее отсутствие столкновений ТС при их параллельном движении на соседних полосах.

Таким образом, при поперечном смещении ТС на малое значение Δx будем считать, что оно частично занимает полосу, на которую осуществляется перестроение.

Как показали наблюдения, для совершения перестроения ТС водитель может действовать, в общем случае, согласно двум стратегиям:

- **плавное перестроение** — кратковременная смена угла рулевого колеса и возвращение в исходное положение в целях отклонения ТС на небольшой угол для медленного маневра;
- **быстрое перестроение** — резкая смена угла рулевого колеса на больший угол, чем при плавном перестроении, для перемещения на соседнюю полосу. При перемещении части кузова на соседнюю полосу происходит смена положения рулевого колеса на противоположное для стабилизации ТС в полосе перестроения.

Далее будем рассматривать вторую стратегию перестроения, поскольку она представляет больший интерес с точки зрения влияния водителей ТС друг на друга.

Если автомобиль двигается и передние колеса повернуты на некий угол φ, то можно сделать вывод о том, что движение происходит по окружности. Согласно работе [20] движение тела по окружности совершается с угловой скоростью ω, а положение тела на окружности характеризуется углом θ. Скалярная зависимость данных величин определяется выражением:

$$\frac{d\theta}{dt} = \omega, \tag{6}$$

а соотношение между скалярами угловой и линейной скорости определяется как

$$\omega = \frac{v}{R}, \tag{7}$$

где R — радиус окружности, v — скорость ТС.

Определим радиус окружности траектории, по которой двигается ТС.

Расстояние между передней и задней осью обозначим L. На рис. 2 показан треугольник с вершинами

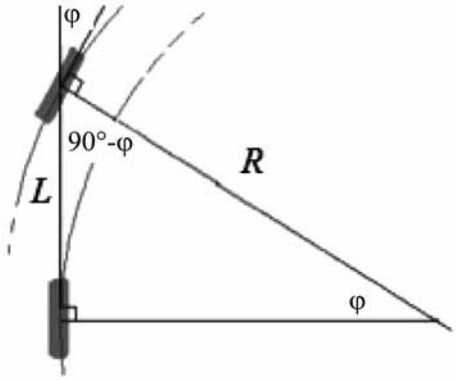


Рис. 2. Движение колес ТС при повороте

в центре окружности и в центре каждого из колес. Угол с задним колесом составляет 90°. Угол между передним колесом составляет 90° − φ. Это означает, что угол при центре окружности равен φ. Соответственно радиус окружности рассчитывается по формуле:

$$R = \frac{L}{\sin(\varphi)}. \tag{8}$$

Учитывая формулы (6)—(8), получаем зависимость для описания положения ТС на окружности при совершении перестроения:

$$\frac{d\theta}{dt} = \frac{v}{L}\sin(\varphi). \tag{9}$$

Вводя продольную y и поперечную x координаты ТС, учитывая (9)

$$x = R\cos(\theta);$$

$$y = R\sin(\theta)$$

при условии (5), получаем модифицированную модель разумного водителя, описывающую положение и скорость машины при прямолинейном движении и перестроении:

$$\begin{cases} \frac{dx}{dt} = -vs\sin(\theta), \\ \frac{dy}{dt} = vs\cos(\theta), \\ \frac{d\theta}{dt} = \frac{sv\sin(\varphi)}{L}, \\ \frac{d\varphi}{dt} = 0, \\ \frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s_0 + vT + v(v - \min(v_c^l, v_l^l, v_r^l))}{2\sqrt{ab}} \Big/ \left(y_l - y - l + c\left|x_{cm} - x\right|\right)\right)^2\right] \end{cases} \tag{10}$$

Параметрами этой модели в дополнении к параметрам соотношения (1) являются:

y_l — координата лидирующего ТС в текущей полосе движения при прямолинейном движении и координата ближайшего лидера на текущей или смежной полосе при совершении перестроения,

v_c^l — скорость лидирующего ТС в текущей полосе;

v_l^l — скорость лидирующего ТС в полосе, находящейся по правую сторону от текущей;

v_r^l — скорость лидирующего ТС в полосе, находящейся по левую сторону от текущей;

x_{cm} — координата, определяющая середину текущей полосы движения'

c — константа, характеризующая скорость перестроения (должна быть больше нуля);

s — константа, которая принимает значение -1 при левом повороте и 1 при правом повороте;

l — длины ТС.

Поясним принцип работы данной модели на примере перестроения ТС с левой полосы на правую. Допускается, что изменение угла поворота рулевого колеса происходит мгновенно.

Начальными условиями системы (10) при совершении перестроения слева направо будут:

$$\begin{aligned} &\varphi_0 = \frac{c}{y_l - y - l}, \\ &\theta_0 = \pi, \\ &x_0 = x_{t=0},\ y_0 = y_{t=0},\ v_0 = v_{t=0}, \\ &s = -1. \end{aligned} \tag{11}$$

Согласно (11), чем ближе ТС, собирающееся совершить маневр, находится к лидеру, тем больше будет угол его перестроения.

Далее, двигаясь по окружности, переместив большую часть кузова на полосу перестроения, водитель ТС меняет угол поворота руля на противоположный и стабилизирует свое положение в полосе. При смене угла поворота в качестве начальных условий используются текущие значения за исключением следующих

$$\theta_0 = \pi + \theta_{\text{текущее}},$$

$$s = 1.$$

В качестве скорости лидера, с которой согласовывает свою скорость водитель ТС, выбирается минимальная из скоростей лидирующих ТС на текущей и соседних полосах движения. Данная функциональность введена в модель в виду того, что условия (3) и (4) недостаточны для совершения перестроения. Численные эксперименты показали, что при моделировании, например, сужения на дороге, на среднегородских скоростях (60 км/ч) ТС, которые должны сменить полосу вследствие сужения, не могут сделать этого в силу большой разницы со скоростями ТС на смежной полосе движения. Благодаря введенной функции минимума скоростей, при проезде рядом с ТС, скорость которого много меньше собственной, происходит небольшое торможение, что объясняет и факт того, что водители ведут себя более настороженно при сильной разнице их собственной скорости со скоростями ТС на соседних полосах движения.

При совершении перестроения, член $c|x_{cm} - x|$ системы уравнений (10) обусловливает в модели следующий принцип: чем дальше ТС отъезжает от середины текущей полосы, тем меньше оно согласует свою скорость с лидером. Поэтому даже находясь вплотную к препятствию, ТС совершит маневр. При прямолинейном движении этот член обращается в ноль.

Вычислительный эксперимент

Для проверки предположения о провале интенсивности потока, выдвинутого в работе [19], моделировалось движение ТС в четыре полосы на протяженности одного км. Поскольку эмпирические данные, описанные в работе [19], снимались на участке третьего транспортного кольца от Автозаводской улицы до Варшавского шоссе без обоснования загруженности смежных компонентов УДС, влияющих на пропускную способность самого участка, в целях создания реалистичных условий моделирования на промежутке 750—900 м было введено препятствие в крайней правой полосе. Параметры модели представлены в таблице.

Моделирование проводилось в течение 20 ч модельного времени, при этом каждые 30 мин увеличивалась плотность ТС на каждой полосе. В качестве численного метода решения системы дифференциальных уравнений использовался метод Рунге-Кутта четвертого порядка с шагом интегрирования 0,05 с.

Результаты моделирования по четырем полосам сагрегированы в одну полосу и представлены в виде зависимости интенсивности ТП от его плотности (рис. 3).

На рис. 3 видно, что при значениях плотности 80—93 ТС/км происходит падение интенсивности. Далее при плотностях более 93 ТС/км интенсивность возрастает, но не достигает значения, как при плот-

Числовые значения параметров модели

Параметр	Значение
Длина ТС	5 м
Длина колесной базы	2,5 м
Начальная скорость	5 м/с
Комфортное ускорение	0,5 м/с^2
Пороговое ускорение ТС	0,2 м/с^2
Комфортное торможение	3 м/с^2
Максимально безопасное торможение	9 м/с^2
Фактор вежливости водителей	1
Минимальная дистанция до лидера при движении в заторе	3 м
Время реакции водителя	1,5 с
Желаемая скорость	17 м/с
Экспонента ускорения	4
Константа, характеризующая скорость перестроения	3

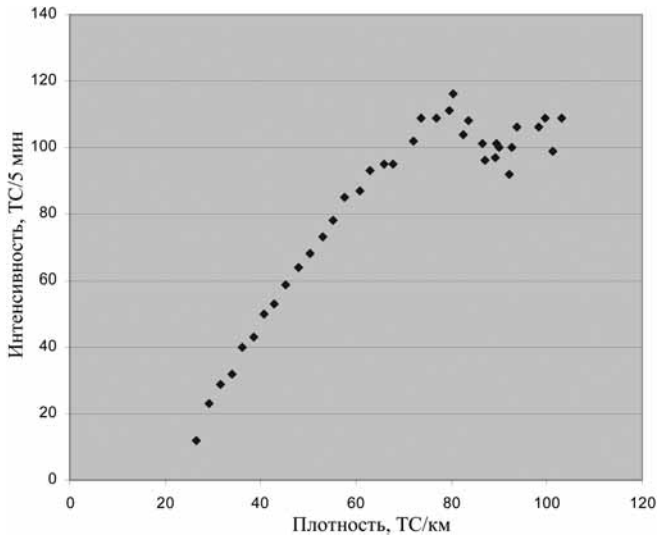


Рис. 3. Зависимость интенсивности ТП от его плотности

ности в 80 ТС/км. Таким образом, численные данные подтверждают предположение, выдвинутое в работе [19], о влиянии перестроений ТС на параметры ТП.

Заключение

В данной работе исследовалось влияние перестроений ТС на ТП. В качестве основы для моделирования движения ТС использовалась модель разумного водителя Трайбера. В ходе работы в нее были внесены изменения для описания процесса перестроения ТС. В качестве апробации модели проверялось предположение, основанное на эмпирических данных Центра Исследования Транспортной Инфраструктуры, о влиянии перестроений ТС на ТП при различных плотностях. Как показал численный эксперимент, разработанная модель подтверждает предположение о влиянии перестроений ТС на интенсивность ТП при различных плотностях и, следовательно, может быть использована на практике.

Список литературы

1. **Число** собственных легковых автомобилей на 1000 человек населения по субъектам Российской Федерации. Федеральная служба государственной статистики. URL: http://www.gks.ru/scripts/db_inet/dbinet.cgi?pl=1316016.

2. **Гольц Г. А.** О философии транспорта // Материалы XI международной (четырнадцатой екатеринбургской) научно-практической конференции. URL: http://waksman.ru/Russian/Systems/g2005.htm.

3. **Wardrope J. G.** Some theoretical aspects of road traffic research // Proc. of the Inst. of Civil. 1952. Part II. P. 325—378.

4. **Newell G. F.** Nonlinear effects in the dynamics of car — following // Operational Research. 1961. V. 9. P. 209—229.

5. **Gazis D. C.** Traffic science. N.Y.: Wiley, 1974.

6. **Gartner N. H., Messer C. J., Rathi A. K.** Traffic flow theory: A state-of-the-art report. Washington DC: Transportation Research Board, 2001.

7. **Kerner B. S.** Introduction to modern traffic flow theory and control. The long road to three — phase traffic theory. Springer, 2009.

8. **Швецов В. И.** Математическое моделирование транспортных потоков // Автоматика и телемеханика. 2003. № 11. С. 3—46.

9. **Helbing D.** Traffic and related self-driven many particle systems // Reviews of modern physics. 2001. V. 73. N 4. P. 1067—1141.

10. **Treiber M., Helbing D.** Explanation of observed features of selforganization in traffic flow // e-print. URL: http://arxiv.org/pdf/cond-mat/9901239v1.pdf.

11. **Kesting A., Treiber M., and Helbing D.** General Lane-Changing Model MOBIL for Car-Following Models // e-print. URL: http://akesting.de/download/MOBIL_TRR_2007.pdf.

12. **Laval J. A., Daganzo C. F.** Lane-Changing in Traffic Streams // Transportation Research B. 2006. Vol. 40. N 3. P. 251—264.

13. **Hidas P.** Modelling Vehicle Interactions in Microscopic Traffic Simulation of Merging and Weaving // Transportation Research C. 2005. Vol. 13. P. 37—62.

14. **Coifman B., Krishnamurthy S., and Wang X.** Lane-Changing Maneuvers Consuming Freeway Capacity // In Traffic and Granular Flow '03 / S. Hoogendoorn, S. Luding, P. Bovy, M. Schreckenberg, and D. Wolf, eds., Berlin: Springer, 2005. P. 3—14.

15. **Wei H., Lee J. J., Li Q., and Li C. J.** Observation-Based Lane—Vehicle Assignment Hierarchy: Microscopic Simulation on Urban Street Network // In Transportation Research Record: Journal of the Transportation Research Board 1710, TRB, National Research Council, Washington, D. C., 2000. P. 96—103.

16. **Brackstone M., McDonald M., and Wu J.** Lane Changing on the Motorway: Factors Affecting Its Occurrence, and Their Implications // In Proc. of 9th International Conference on Road Transportation Information and Control. Conference Publication 454. London. 1998. P. 160—164.

17. **Nagel K., Wolf D., Wagner P., and Simon P.** Two-Lane Traffic Rules for Cellular Automata: A Systematic Approach // Physical Review E. 1998. Vol. 58. P. 1425—1437.

18. **Laval J. A., Daganzo C. F.** Multi-Lane Hybrid Traffic Flow Model: Quantifying the Impacts Lane-Changing Maneuvers on Traffic Flow // Working Paper UCB-ITS-WP-2004-1. Institute of Transportation Studies. University of California, Berkeley, 2004.

19. **Гасников А. В., Кленов С. Л., Нурминский Е. А.** и др. Введение в математическое моделирование транспортных потоков: учеб. пособие / Под ред. А. В. Гасникова. М.: Изд. МФТИ, 2010. 362 с.

20. **Савельев И. В.** Курс общей физики. Т. 1. Механика, колебания и волны, молекулярная физика. М.: Наука, 1970. 508 с.

УДК 004.42

Т. Я. Кроль, канд. техн. наук, e-mail: tatyana.kroll@cit.ecm.ru,
М. А. Харин, аспирант, e-mail: maxim.kharin@cit.ecm.ru,
Ивановский государственный энергетический университет им. В. И. Ленина,
Ивановский центр информационных технологий —
филиал ОАО "Электроцентромонтаж"

Опыт построения и реализации электронного архива на базе системы сканирования и распознавания Flexi Capture

Описаны принципы построения и опыт реализации системы электронного архива на основе системы сканирования и распознавания ABBYY Flexi Capture. Приведена схема занесения документов в архив с указанием основных компонентов системы. Рассмотрены особенности доступа к данным и приведена схема взаимодействия компонентов архива с базой данных. Описана реализация процесса сканирования и распознавания документов с использованием дополнительно разработанных методов ускорения верификации. Также рассмотрен пример реализации системы конфигурирования архива и интернет-приложения для доступа к документам архива. Подобный опыт может быть использован разработчиками при создании собственных систем электронного архива.

Ключевые слова: *электронный архив, построение и опыт реализации, доступ к данным, сканирование, распознавание, верификация, интернет-приложение, ABBYY Flexi Capture*

Введение

Одним из самых ценных активов организации является корпоративная информация, представляющая собой определенный набор документов и материалов. Важной задачей является обеспечение хранения и конфиденциальности такой информации. Электронные архивы позволяют предприятиям надежно хранить информацию, предоставляя к ней оперативный доступ в случае необходимости. В соответствии со стандартом MoReq II [1] они имеют следующие особенности:

- не допускают изменения документа без создания отдельной версии;
- позволяют существовать единственной финальной версии документа;
- запрещают удаление документов, кроме некоторых строго контролируемых ситуаций;
- должны включать строгие правила хранения;
- должны включать строго упорядоченную структуру документов (классификационную схему), которая задается администратором;
- могут поддерживать ежедневную работу, но в первую очередь предназначены для обеспечения защищенного хранения бизнес-значимых документов.

В данной статье рассмотрены принципы построения системы электронного архива, а также пример реализации данных принципов в системе "ДокПрофи^{TM}" [2].

Схема занесения документов в архив

При занесении документов в архив используется схема, представленная на рис. 1. Документ проходит через этапы сканирования, распознавания, верификации и собственно занесения в архив [3].

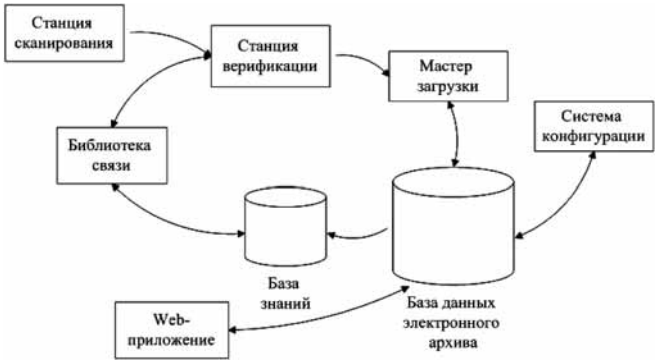


Рис. 1. Схема наполнения электронного архива документами

Особенности работы с базой данных архива

Один из ключевых компонентов электронного архива — база данных. В процессе работы к данным в базе обращаются система конфигурации, мастер загрузки документов, web-приложение, а также разрабатываемые компоненты для поиска знаний в архиве. База данных представляет собой физическую реализацию моделей данных электронного архива, описанных в статьях [4, 5]. Эти модели позволяют хранить документы различной структуры, определять структуру предприятия (как филиальную, так и иерархическую внутри филиала), реализовать права доступа к документам, осуществлять поиск документов по определенным признакам. База данных реализована с использованием СУБД "Oracle".

Доступ к данным для просмотра осуществляется с помощью виртуальных таблиц — представлений (*view*). Они дают следующие преимущества:

- добавляют уровень защиты данных;
- могут скрывать сложность данных, комбинируя нужную информацию из нескольких таблиц;
- могут скрывать настоящие имена столбцов, порой трудные для понимания, и показывать более простые имена.

Доступ к данным для добавления, редактирования и удаления осуществляется с помощью хранимых процедур, которые:

- повышают производительность: при вызове хранимой процедуры ее содержимое сразу же обрабатывается сервером, компилированная процедура выполняется быстрее, позволяют выполнять сложную транзакционную логику, избегая передачи большого количества команд;
- расширяют возможности программирования: обеспечивают модульность и стимулируют повторное использование кода;
- поддерживают функции безопасности данных: позволяют ограничить или полностью исключить непосредственный доступ к таблицам, а также снижают вероятность внедрения SQL-кода.

Обращение к базе данных осуществляет сервер электронного архива, принимающий запросы от клиентских приложений. При реализации сервера была использована ORM-технология NHibernate [6] (ORM — *object-relational mapping*, объектно-реляционное отображение). Эта технология позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов. Дополнительным преимуществом NHibernate является поддержка различных СУБД, соответственно появляется возможность осуществить перевод на другую систему, не затрагивая основного кода сервера.

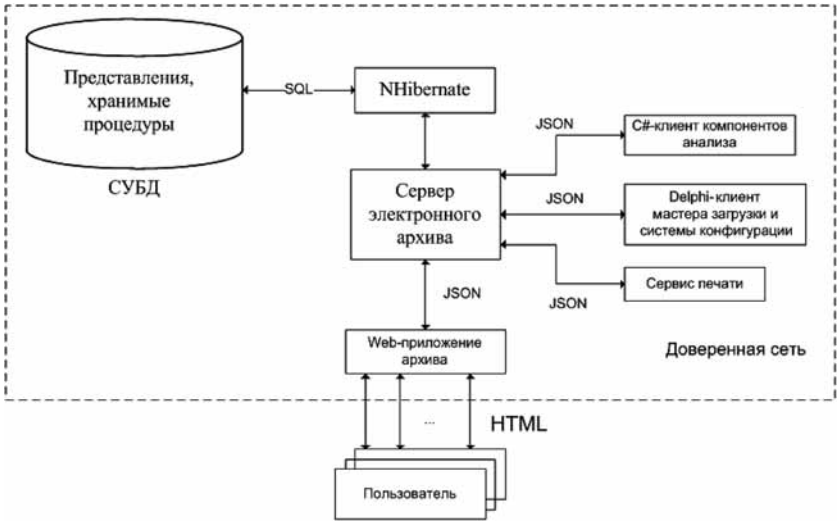


Рис. 2. Схема взаимодействия компонентов архива с базой данных

В целом взаимодействие компонента архива с базой осуществляется по схеме, приведенной на рис. 2.

Для обмена данными с сервером электронного архива используется технология JSON. Это позволяет реализовать клиентскую часть приложений на разных языках программирования (например, в системе используются языки Delphi, C#, Java) за счет унификации обращений.

Итак, применяемая архитектура дает такие преимущества, как повышение скорости и надежности работы, распределенность и простота доступа к сервису архива, масштабируемость, многоуровневая защита данных. Каждый компонент системы может быть изменен независимо от других: есть возможность сменить платформу СУБД, использовать нужные языки программирования для реализации клиентских приложений, добавлять и уда-

лять новые компоненты архива, например, компоненты интеллектуального анализа данных. Отметим, однако, что при смене СУБД необходимо соответствующим образом переписать код хранимых процедур. Но при этом сохраняются их интерфейсы, поэтому такое изменение кода не затронет других компонентов. Например, авторами была реализована версия архива, работающая на СУБД Microsoft SQL Server. Для этого с помощью инструмента создания моделей баз данных Power Designer был сгенерирован скрипт для создания базы, переписаны сохраненные процедуры и изменены настройки NHibernate. Таким образом, переход был осуществлен достаточно быстро.

Реализация процесса сканирования и распознавания

Для сканирования используется станция сканирования Flexi Capture 10 (рис. 3).

Станция предоставляет возможность либо получения документов со сканера, либо загрузки из файла. Полученные страницы объединяются в документы, документы — в пакеты, например, по дате или типу. В системе Flexi Capture создан проект, содержащий набор шаблонов для всех документов (счет-фактура, накладная, рабочая документация, акт и др., всего около 50 шаблонов). Шаблон документа (в новых версиях продукта "Определение документа") определяет расположение элементов документа, указывает, откуда будут извлечены данные. При создании шаблона документа определяются свойства полей, их область значений. Наложение шаблонов на сканированные документы осуществляется в автоматическом или ручном режиме.

Далее запускается процесс распознавания. Полученный текст записывается в соответствующие поля документа (рис. 4).

Неуверенно распознанные символы в полях выделяются цветом. Задача верификатора — проверить качество распознавания и заполнить поля документа. Для некоторых шаблонов, например, счета-фактуры, настроены скрипты для связи с библиотекой, работающей со справочниками и последовательностями.

Справочник представляет собой набор правил вида: *"Если* $A_1 = s_1$*, то* $A_2 = s_2$ *с вероятностью* x*"*. Здесь A_1 и A_2 — некоторые атрибуты, $A_1 \in TA$, $A_2 \in TA$, TA — множество всех доступных атрибутов типа, s_1 и s_2 — значения атрибутов, x — численное значение вероятности. Более подробно справочники и их модель данных описаны в статьях [4, 7].

Последовательность — это закономерность вида: *"Если значение атрибута* A_1 *документа* $Д_1$ *типа* T_1 *равно значению атрибута* A_1 *документа* $Д_2$ *типа* T_2*, то значение атрибута* A_2 *документа* $Д_2$ *равно значению атрибута* A_2 *документа* $Д_1$ *с вероятностью* x*"*. Здесь T_1 и T_2 — определенные в архиве типы документов, A_1 и A_2 — определенные в архиве атрибуты документов, $Д_1$ и $Д_2$ — некоторые документы архива, x — численное значение вероятности. Последовательности и их модель более подробно описаны в статьях [4, 8].

Библиотека связи реализована в виде dll-файла, предоставляющего определенные методы работы.

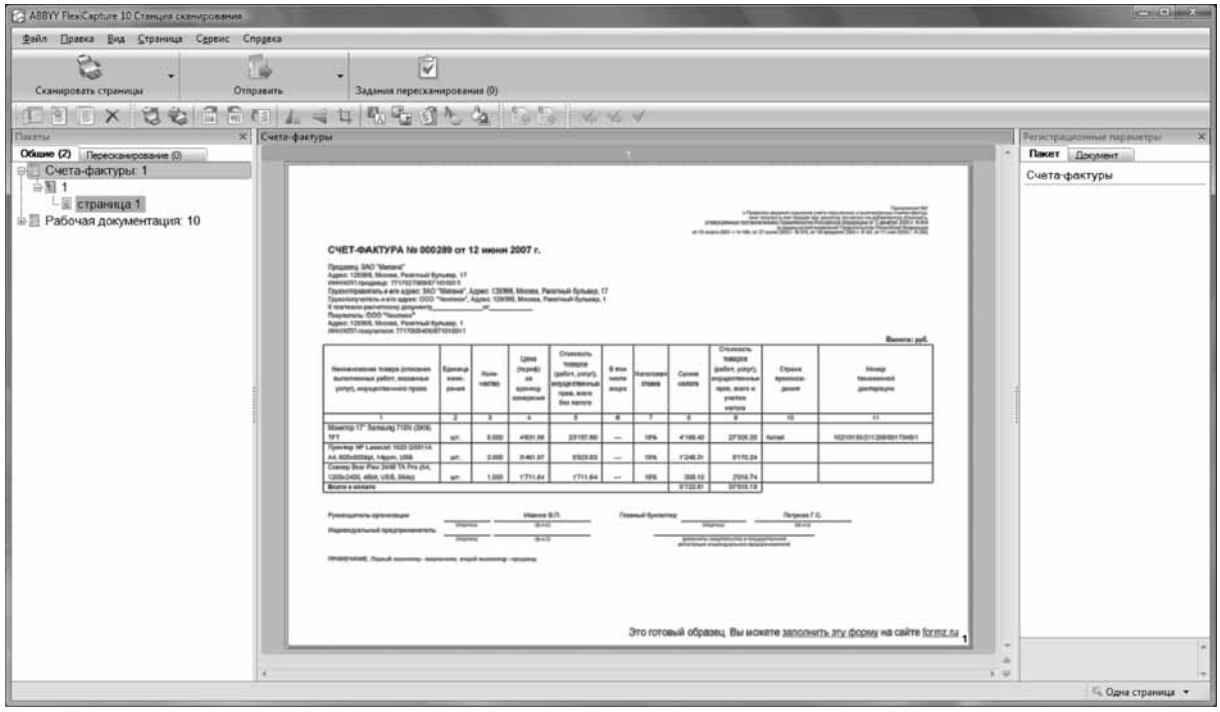


Рис. 3. Интерфейс станции сканирования Flexi Capture 10

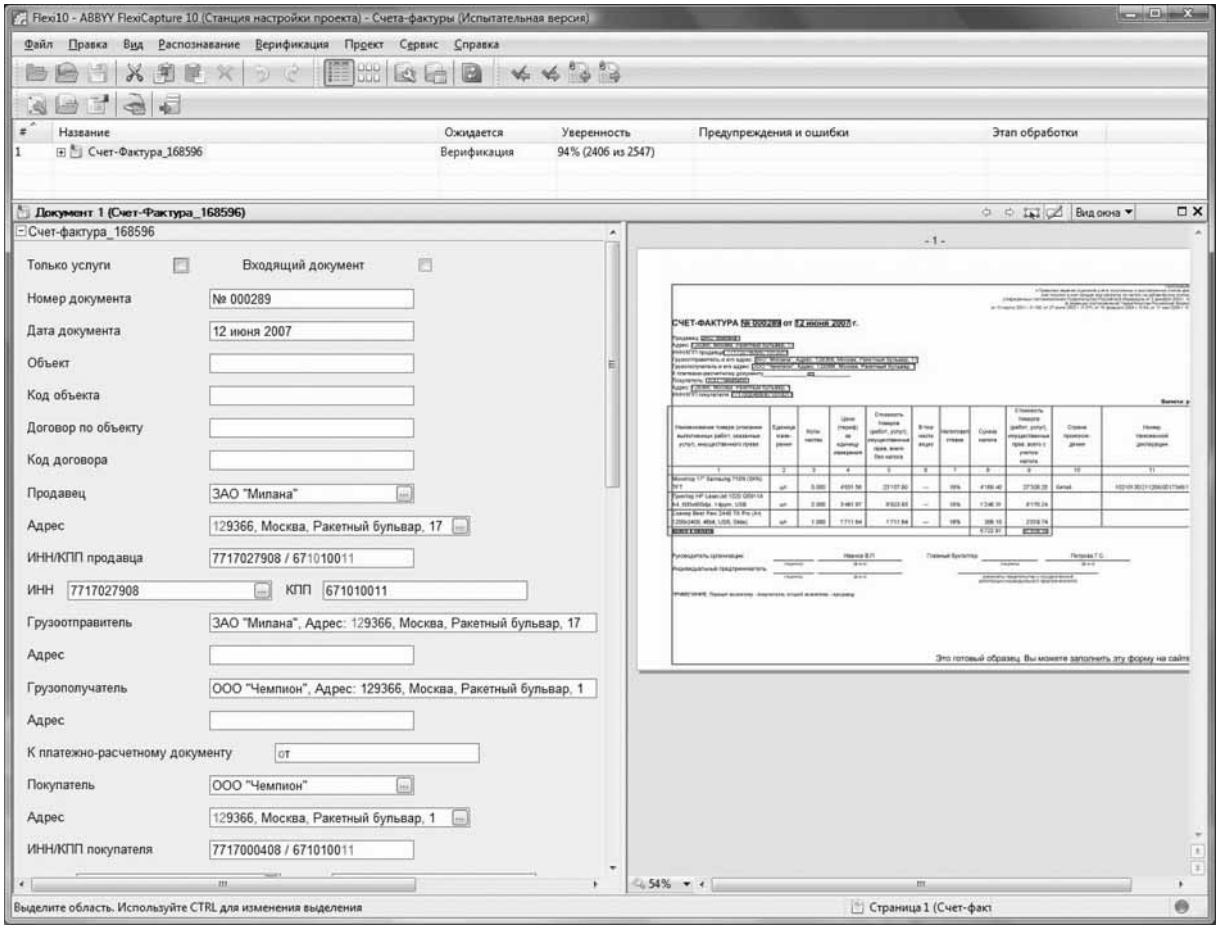


Рис. 4. Распознанный документ в системе Flexi Capture 10

При нажатии на кнопку "..." у поля (рис. 4) в библиотеку передается контекст документа: его поля, выделенное поле и другая информация. Библиотека обращается к базе данных, ищет подходящие правила, находит значения и возвращает в программу верификации. Соответственно, если какое-либо поле распознается уверенно (таким полем часто бывает, например, ИНН организации), то часть других полей можно взять из базы (например, наименование и адрес организации).

Справочники и последовательности формируются на основе документов, уже загруженных в архив. Для формирования написана специальная утилита, реализующая методы, описанные в работах [7] и [8], и использующая модель, описанную в работе [4]. В частности, реализованы методы складывающихся столбцов и полного вероятностного справочника и метод поиска последовательностей. Указываются типы и атрибуты для поиска правил и вызываются соответствующие алгоритмы. Запуск алгоритмов может выполняться с некоторой периодичностью для поддержания актуальности правил.

Когда документ отверифицирован, он экспортируется в XML-файл с сохранением изображения в PDF-файле. Экспорт осуществляется с помощью скрипта, настройки которого можно изменить, используя специальную утилиту настроек (рис. 5). В XML-файле сохраняется следующая информация:

- тип шаблона;
- список полей и значений;
- ссылка на файл изображения;
- информация о состоянии верификации (сохраняется опционально), т. е. маска, показывающая уверенность или неуверенность распознавания символа.

Затем полученные файлы обрабатываются отдельной программой загрузки.

Таким образом, сканирование, распознавание и верификация документов осуществляются с помощью программного продукта Flexi Capture 10 с использованием возможности подключения пользовательских скриптов и библиотек. Данные возможности позволяют повысить скорость верификации документов человеком, что становится особенно актуально при увеличении потока документов, внедрении системы в новых организациях и филиалах.

Рассмотрим подробнее систему конфигурации, мастер загрузки (обработки) документов и web-приложение для просмотра и поиска документов.

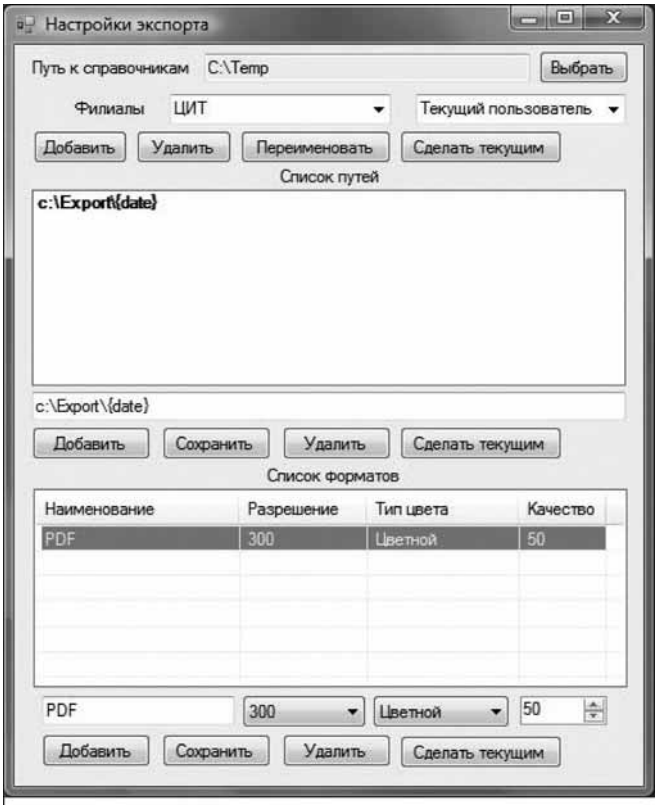


Рис. 5. Утилита настроек экспорта

Реализация системы конфигурации, мастера загрузки и web-приложения

Система конфигурации предназначена для настройки системы электронного архива. При запуске выполняется авторизация в виде формы ввода имени и пароля, их проверка осуществляется сервером "Электронного архива". К работе с данной подсистемой могут быть допущены только пользователи, входящие в системную группу "Администраторы". Общий интерфейс системы настройки приведен на рис. 6.

Интерфейс представляет собой форму с вкладками для настроек и панелью инструментов. Вкладка "Типы документов" содержит элементы управления для следующих действий:

- добавление типов документов;
- изменение и удаление типов документов, для которых еще не созданы экземпляры документов;
- добавление и изменение имен атрибутов типов документов;
- удаление имен атрибутов, не задействованных ни в одном типе документов;
- формирование атрибутов типов документов, для которых еще не созданы экземпляры документов;
- задание последовательности отображения атрибутов документов и типа сортировки по атрибутам документа.

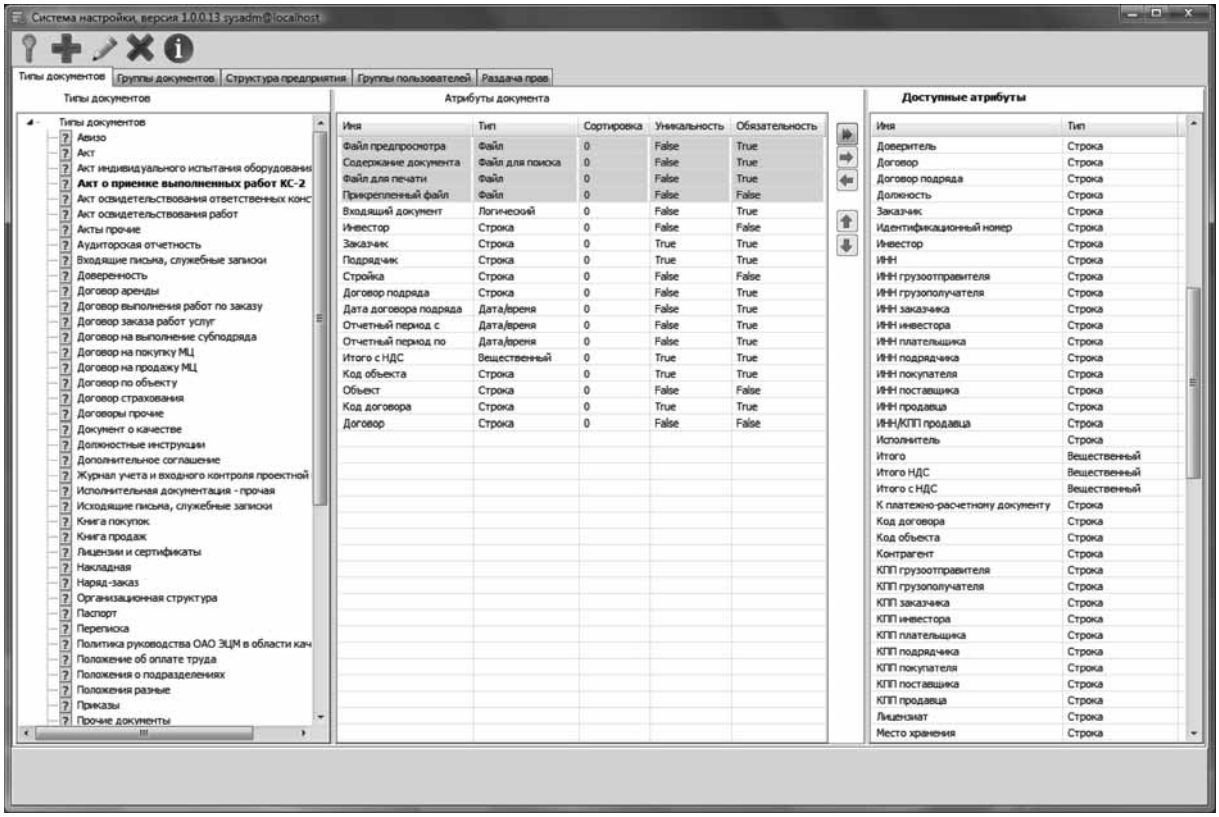


Рис. 6. Интерфейс системы настройки

Элементы на вкладке "Группы документов" позволяют выполнять следующие действия:

- добавление и изменение групп документов;
- удаление пустых групп документов;
- распределение типов документов по группам.

Элементы на вкладке "Структура предприятия" позволяют выполнять следующие действия:

- добавление и изменение филиалов организации;
- удаление пустых филиалов, на которые нет ссылок в экземплярах документов;
- добавление и изменение подразделений;
- удаление пустых подразделений;
- добавление и изменение пользователей системы;
- удаление пользователей системы, о которых нет записей в журнале архива.

Элементы вкладки "Группы пользователей" аналогичны элементам вкладки "Группы документов" и позволяют выполнять следующие действия:

- добавление и изменение групп пользователей;
- удаление пустых групп пользователей;
- распределение пользователей по группам.

Наконец, элементы последней вкладки "Раздача прав" позволяют выполнять назначение прав на документы группам пользователей. Возможные права: чтение, изменение, удаление, полный доступ.

Рассмотрим мастер обработки документов. Он обеспечивает возможность автоматизированного ввода документов в электронное хранилище, уменьшая вероятность ошибок, повышая скорость ввода и исключая многократный ввод документов. Основной интерфейс программы приведен на рис. 7.

Интерфейс представляет собой форму с вкладками, на которых можно выполнять следующие действия:

- подготовка документов к загрузке в архив;
- распределение загруженных документов по пакетам;
- интеграция с внешними бухгалтерскими системами (например, 1С, КИС "Флагман");
- управление файлами: удаление либо откат для повторной загрузки уже загруженных файлов.

Как отмечалось ранее, после обработки документа в системе Flexi Capture (сканирование, распознавание, верификация) он экспортируется в XML-файл с сохранением изображения в файле PDF. Данное изображение используется затем в качестве файла для печати. Однако также необходим файл предпросмотра, поэтому при загрузке документа в мастер происходит автоматическое формирование предпросмотра в формате PNG из первой страницы файла для печати. Сама загрузка выполняется либо вручную, либо автоматически через указанный интервал времени. При этом в мастер попадают все незагруженные документы из папки, указанной в настройках программы.

Также предоставляется возможность повторной проверки и корректировки полей документа и прикрепления к нему дополнительных файлов. Работа с мастером обычно проводится верификаторами сразу после экспорта документов из Flexi Capture.

Далее рассмотрим web-интерфейс, позволяющий пользователям искать, просматривать и печатать необходимые документы. Обычным пользователям доступна только эта подсистема, с ней можно ознако-

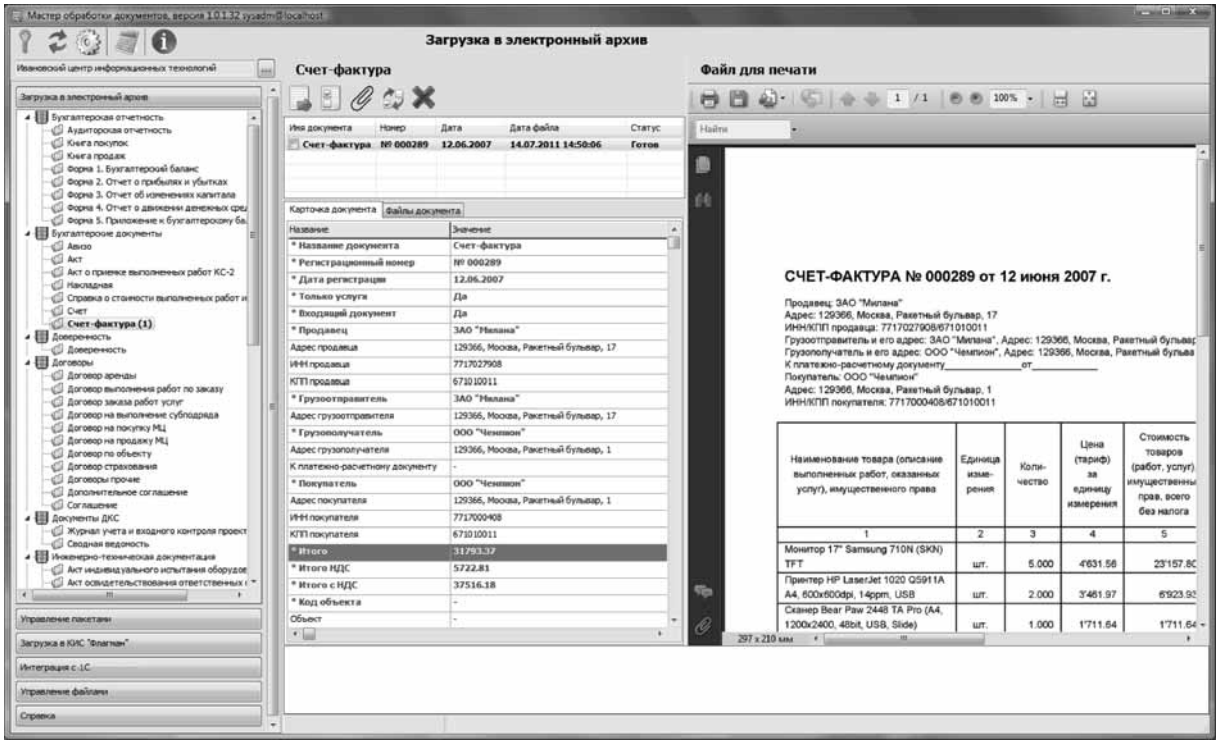


Рис. 7. Интерфейс мастера обработки документов

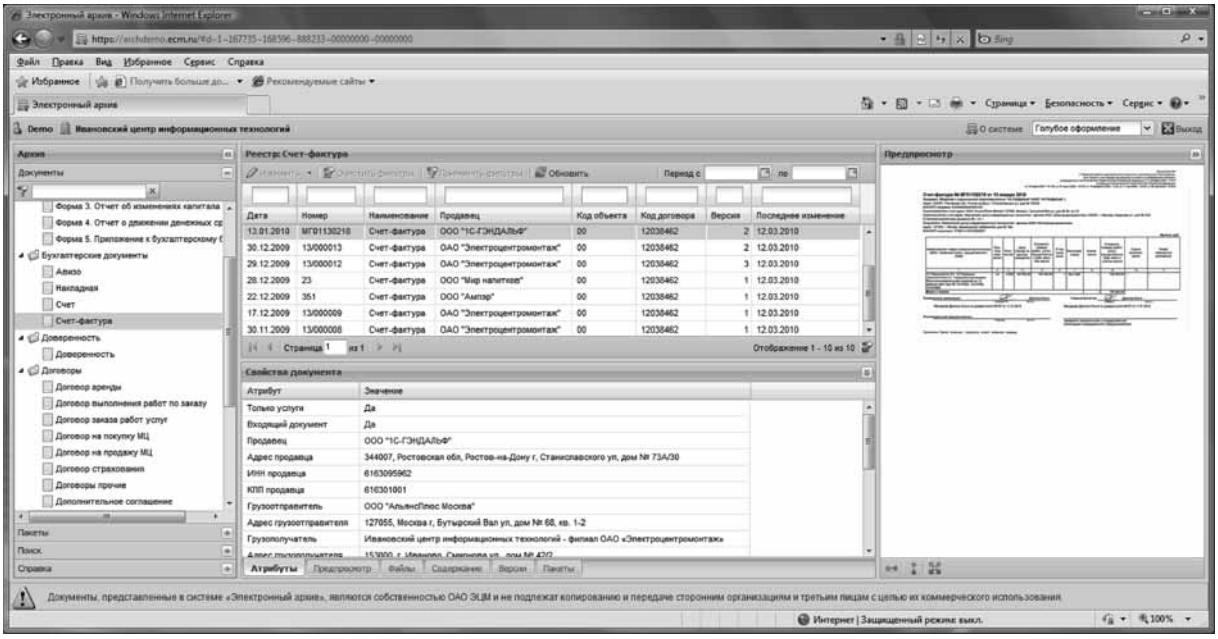


Рис. 8. Web-интерфейс электронного архива

миться на сайте [9], используя имя пользователя Demo и пароль Demo. Данный интерфейс приведен на рис. 8.

Документы рассортированы по группам и типам документов. Для быстрого поиска документов определенного типа можно использовать фильтры в верхней части таблицы. У выбранного документа можно просмотреть набор его атрибутов, файл предпросмотра, остальные файлы документа, а также все предыдущие версии. При наличии соответствующих прав пользователь может изменить значения атрибутов документа, при этом создается новая версия документа.

Внешний вид вкладки "Пакеты" аналогичен вкладке "Документы", за исключением того, что документы рассортированы не по типам, а по иерархической структуре пакетов. Отметим, что документ может одновременно входить в несколько пакетов. При загрузке документа в архив он автоматически попадает в пакет, соответствующий дате загрузки. Эти пакеты имеют структуру "Год" -> "Месяц" -> "День". Отметим, что пакеты документов представляют собой только логические группы, документы всегда остаются в архиве, как бы их ни группировали.

На вкладке "Поиск" можно найти нужный документ по запросу. Для этого задаются следующие параметры.

- Тип документа. Можно задать один или несколько типов для поиска.
- Дата. Задается либо конкретное значение даты, либо диапазон дат.
- Номер и наименование. Эти параметры представляют собой строковые значения. К различным значениям можно применить логические условия:
  - ИЛИ — значение параметра поиска будет равно одному из выбранных значений;
  - ПОДОБНО — значение параметра поиска будет содержать в себе хотя бы одно из выбранных значений;
  - КРОМЕ — в найденных документах значение параметра поиска не будет равно ни одному из выбранных значений;
  - НЕ ПОДОБНО — в найденных документах значение параметра поиска не будет содержать в себе ни одно из выбранных значений.
- Текст документа. Задаются слова для поиска и следующие параметры: искать как словосочетание или в пределах нескольких слов, искать, включая синонимы.
- Значения конкретных атрибутов документов. Отметим, что когда выбран тип документов, то набор доступных атрибутов ограничивается этим типом. Если выбрано несколько типов, то ищется пересечение множеств атрибутов этих типов.

Выводимые результаты поиска ограничены 500 документами. Если искомого документа нет среди найденных 500, следует уточнить условия поиска.

Для печати документов разработан специальный сервис. Имеется возможность печати отдельного документа при выборе его в реестре, либо полного пакета (с подпакетами) документов с вкладки "Пакеты". Пакетную печать удобно использовать, если пакеты организованы, например, по контрагентам, договорам, соответствию документов (например, когда в пакете содержится справка о стоимости выполненных работ и затрат по форме КС-3 и соответствующие ей акты о приемке выполненных работ по форме КС-2). Тогда при подготовке отчетов достаточно найти нужный пакет и выполнить его печать.

Весь процесс занесения документов в архив отражен в презентации [10].

Отметим, что реализованная система соответствует требованиям стандартам MoReq, в частности, при изменении документа создается новая версия, при этом последняя версия становится актуальной; удаление документов средствами архива невозможно, удаление документа из какого-либо пакета не приводит к удалению из архива; структура документов и назначение прав доступа к ним регулируются администраторами в единой системе конфигурации.

Заключение

Приведенные в данной статье принципы построения и реализации позволяют построить эффективную систему электронного архива. Более подробная информация по моделям данных и методам обработки данных может быть найдена в работах [4, 5, 7, 8]. Подобная система предназначена для решения проблем размещения, хранения и быстрого поиска большого количества документов, что особенно важно для крупных предприятий со сложной филиальной и организационной структурой. Примером реализации приведенных идей является система "ДокПрофи^{TM}", однако, они могут успешно применяться и в других системах электронных архивов.

Список литературы

1. **Типовые** требования к управлению электронными официальными документами. Спецификация MoReq2. URL: http:// ec.europa.eu/transparency/archival_policy/, свободный.

2. **Свидетельство** о государственной регистрации программы для ЭВМ № 2011610409 ДокПрофи.

3. **Кроль Т. Я., Харин М. А., Евдокимов П. В.** Схема наполнения электронного архива документами // Материалы первой международной конференции "Автоматизация управления и интеллектуальные системы и среды". Терскол, 20—27 декабря. 2010. Т. IV. С. 53—56.

4. **Кроль Т. Я., Харин М. А.** Расширение модели документа электронного архива с целью извлечения и использования накопленных знаний // Материалы конференции "Наука в информационном пространстве — 2011" 29—30.09.2011 г. URL: http://www.confcontact.com/20110929/tn_hrol.htm.

5. **Кроль Т. Я.** и др. Модели данных для реализации поиска и прав доступа к документам // Молодой ученый. 2011. № 11. С. 79—84.

6. **ORM**-система NHibernate. URL: http://nhibernate.org/, свободный.

7. **Кроль Т. Я., Харин М. А., Евдокимов П. В.** Методы создания справочника на основе электронного архива // Известия КБНЦ РАН. 2011. № 1. С. 154—158.

8. **Кроль Т. Я., Харин М. А.** Использование последовательностей при занесении документов в электронный архив // Материалы конференции "Спецпроект: анализ научных исследований". 30—31.05.2011 г. URL: http://www.confcontact.com/20110531/ tn8_krol.htm.

9. **Демо-версия** системы электронного архива "ДокПрофи^{TM}". URL: https://archdemo.ecm.ru.

10. **Презентация** "Электронный архив. Процесс сканирования, распознавания, верификации и загрузки". URL: http://www.cit.ecm.ru/sites/all/files/cit/doc/docprofi-electron-archive.pps.

ИНФОРМАЦИЯ

17—20 октября 2012 г. в г. Санкт-Петербурге будет проходить Шестая Международная конференция

«Математические методы, модели и архитектуры для защиты компьютерных сетей» (MMM-ACNS-2012)

Конференция будет проходить совместно со Вторым международным семинаром

«Научный анализ и поддержка политик безопасности в киберпространстве» (SA&PS4CS-2012)

Основными целями конференции MMM-ACNS-2012 являются обсуждение современного состояния в области математических методов и моделей защиты компьютерных сетей и обеспечения информационной безопасности для содействия лучшему пониманию последних достижений и тенденций в области безопасности компьютерных сетей, а также использование последних достижений в области современных информационных технологий для защиты компьютерных сетей и обеспечения информационной безопасности.

http://comsec.spb.ru/mmm-acns12/
http://www.comsec.spb.ru/saps4cs12/

УДК 004.4'423

П. М. Карпов, аспирант, e-mail: reoser@mail.ru,
Вычислительный центр им. А. А. Дородницына РАН, г. Москва

Быстрый интерпретатор языка Brainfuck

Статья посвящена созданию эффективного в плане производительности интерпретатора языка Brainfuck и рассмотрению различных методов его оптимизации. В основу создания быстрой виртуальной машины легла техника объединения инструкций, подкрепленная статистическим анализом Brainfuck-программ. Приводится вклад различных макроинструкций в производительность и сравнение с другими интерпретаторами.

Ключевые слова: *эзотерические языки программирования, интерпретатор, виртуальная машина*

Введение

Brainfuck (в дальнейшем — BF) — эзотерический язык программирования, известный своим минимализмом. Машина, управляемая BF-программой, представляет собой массив ячеек, изначально инициализированный нулями, указатель, изначально указывающий на самую левую ячейку, а также потоки ввода и вывода. Сам язык состоит из восьми представленных далее (табл. 1) команд. Программа на языке BF состоит из последовательности этих команд, выполняющихся одна за другой и прочих символов, которые игнорируются при выполнении.

Таблица 1

Команды языка BF

Команда	Описание
>	Передвинуть указатель на одну ячейку вправо
<	Передвинуть указатель на одну ячейку влево
+	Прибавить 1 к текущей ячейке
–	Отнять 1 от текущей ячейки
[	Если текущая ячейка равна нулю, перейти к команде после соответствующей]
]	Если текущая ячейка не равна нулю, перейти к команде после соответствующей [
.	Вывод значения из текущей ячейки
,	Ввод значения в текущую ячейку
Примечание: команды [и] всегда имеют совпадающие парные команды и в этом отношении ведут себя так же, как скобки в арифметических выражениях.	

Конкретные характеристики виртуальной BF-машины, такие как размер массива, разрядность ячеек, а также обработка ситуаций арифметического переполнения и выхода указателя за границы массива не стандартизированы и зависят от конкретной реализации языка. Полезный набор правил создания BF-машин описан в работе [1]. В данной статье используются следующие соглашения:

- ячейки являются беззнаковыми числами размером 1 байт;
- операции "+" и "–" выполняются по модулю 256.

Язык BF Тьюринг-полон [2] и, следовательно, в принципе не уступает высокоуровневым языкам по своим возможностям. Ввиду своей простоты, BF является хорошим языком для тестирования систем генетического программирования. Желание добиться высокой скорости вычислений в подобных задачах и явилось основным мотивом выполнения этой работы.

Выполнение BF-программ

Существует большое число как интерпретаторов, так и компиляторов языка BF. Компиляция потенциально способна дать большую скорость выполнения, однако обладает и рядом недостатков. В случае небольших программ накладные расходы на компиляцию могут превысить время выполнения самой программы. При необходимости отслеживать параметры,

такие как число выполненных команд и ошибки выхода указателя за границы массива, компиляция усложняется. По этой причине ограничимся рассмотрением интерпретации. Техники создания и оптимизации виртуальной BF-машины представлены ниже.

- **Наивное выполнение.** Оно представляет собой поочередную интерпретацию элементарных команд.
- **Предвычисление адресов переходов.** Так как адреса ячеек, на которые возможны условные переходы в результате выполнения команд [и] не меняются со временем, возможно их предварительное вычисление, что избавляет от необходимости поиска парных операторов во время интерпретации.
- **Сжатие одинаковых команд.** Типичные BF-программы содержат большое число одинаковых идущих подряд команд, что неудивительно в силу природы языка. Несложной, но эффективной оптимизацией является объединение таких команд в одну, которая прибавляет к ячейке эквивалентное значение или перемещает указатель на эквивалентное число позиций.
- **Макрокоманды.** Более общей версией предыдущего способа оптимизации является распознавание часто встречающихся последовательностей инструкций и их замена специальными макрокомандами. Примеры таких идиом — обнуление ячейки ([−] или [+]) и поиск ненулевого значения ([<] или [>]). Эта техника часто используется при создании эффективных виртуальных машин. Крупные команды выгодны тем, что меньше времени тратится на пересылку управления.
- **Простые циклы.** Это циклы, удовлетворяющие следующим условиям: содержат лишь операции +− <>, содержат одинаковое количество < и >, изменяют значение проверяемой ячейки на ±1. Число итераций в таких случаях зависит только от значения текущей ячейки, что соответствует циклам типа *for* в высокоуровневых языках.

Хотя возможные техники оптимизации этим не исчерпываются (более полный список можно найти на сайте [3]), перечисленных приемов вполне достаточно для написания быстрого интерпретатора. Более сложные техники могут быть полезны при создании компилятора.

Таблица 2

Характеристики часто встречающихся циклов

Тип цикла	Инструкции	Действие	Частота встречаемости, %	
Очистка	`[-]` `[+]`	`a[p] = 0`	19	
Поиск	`[N<]` `[N>]`	`a[p'] = 0`	14	

Деструктивное сложение/вычитание	`[-N>+N<]`
	`[N>+N<-]`
	`[-N<+N>]`
	`[N<+N>-]`
		`a'[p] = 0, a'[p + N] = a[p + N] + a[p]`
		`a'[p] = 0, a'[p - N] = a[p - N] + a[p]`
			12	13

	`[-N>-N<]`
	`[N>-N<-]`
	`[-N<-N>]`
	`[N<-N>-]`
		`a'[p] = 0, a'[p + N] = a[p + N] - a[p]`
		`a'[p] = 0, a'[p - N] = a[p - N] - a[p]`
			1	

Деструктивное умножение	`[>N+<-]`
	`[->N+<]`
	`[<N+>-]`
	`[-<N+>]`
		`a'[p] = 0,`
		`a'[p + 1] = a[p + 1] + a[p] * N;`
		`a'[p] = 0,`
		`a'[p - 1] = a[p - 1] + a[p] * N;`
			10	12

	`[>N-<-]`
	`[->N-<]`
	`[<N->-]`
	`[-<N->]`
		`a'[p] = 0,`
		`a'[p + 1] = a[p + 1] - a[p] * N;`
		`a'[p] = 0,`
		`a'[p - 1] = a[p - 1] - a[p] * N;`
			2	

Деструктивное сложение с близкими		`a'[p] = 0,`
		`a'[p + dp1] = a[p + dp1] + a[p],`
		`a'[p + dp2] = a[p + dp2] + a[p],`
		`...`
			2	

Условный инкремент/декремент	`[[-]N>-N<]`
	`[N>-N<[-]]`
	`[[-]N>+N<]`
	`[N>+N<[-]]`
		`если a[p] <> 0 то`
		`a'[p + N] = a[p + N] - 1, a'[p] = 0`
		`если a[p] <> 0 то`
		`a'[p + N] = a[p + N] + 1, a'[p] = 0`
			2	

Прочие			38	

Примечание: `NX` обозначает `N`-кратное повторение инструкции `X`; `a` — массив ячеек; `p` — указатель ячейки; переменные со штрихом соответствуют состоянию после выполнения цикла.

Статистика макрокоманд

При реализации тех или иных методов оптимизации важно, оправдывается ли прирост скорости увеличением сложности. Бесполезно, например, объединять редко используемые наборы инструкций в макрокоманды. Существует техника динамической генерации макрокоманд [4], однако она значительно сложнее статического варианта. По этой причине для выяснения потенциально наиболее выгодных макрокоманд был проведен статистический анализ множества программ на языке BF, состоявший в подсчете частот встречаемости различных циклов в исходных кодах и их классификации. Результаты этого исследования представлены в табл. 2.

Основной вывод, который можно сделать по результатам статистического анализа заключается в том, что в исследованных BF-программах 58 % циклов представляют собой одну из следующих операций: очистка, поиск, деструктивное сложение/вычитание, деструктивное умножение. Целесообразно выделение этих операций в макрокоманды. Доли остальных типов циклов составляют 2 % и менее. Необходимо отметить, что такое решение полностью не устраняет необходимости выбора оптимальных макроинструкций. Причина в том, что некоторые часто встречающиеся операции не состоят из одного лишь цикла, например копирование ячейки:

```
[>+>+<<-]>>[<<+>>-]<< a'[p + 1] = a[p].
```

Это однако не является большой проблемой — подобные операции обычно включают в себя указанные циклы, выполняемые при помощи макрокоманд.

Реализация интерпретатора

Интерпретатор, применяющий некоторые из рассмотренных техник оптимизации, был реализован в среде Delphi 7. Использовались макроинструкции, представленные в табл. 3.

В отличие от макроинструкции `Add`, которую можно использовать как для сложения, так и для вычитания, перемещение указателя может быть выгодно реализовать при помощи двух команд — `Left` и `Right`. В случае если необходимы проверки выхода за границы массива, выполнение этих инструкций требует проверки выхода лишь за одну из границ, вместо двух для операции общего вида.

Исходя из анализа BF-программ целесообразным выглядит введение макроинструкций, сочетающих сложение и перемещение. В таком случае тела многих циклов могут быть записаны небольшим числом таких инструкций. Неудобство заключается в том, что циклы могут начинаться как с команды сложения, так и перемещения. По этой причине, чтобы добиться искомого минимума, необходимо ввести две инструкции — `AddMove` и `MoveAdd`.

Для преобразования исходного BF-кода в последовательность макрокоманд (BF-байткод) удобно использовать схему наподобие Shift-Reduce-парсера. На каждом шаге осуществляется поиск совпадений комбинации следующей команды исходного BF-кода и нескольких уже обработанных макроинструкций с одним из существующих правил, а также проводится замена на соответствующую макроинструкцию. Для рассматриваемого интерпретатора правила представлены в табл. 4.

Таблица 3

Использованные макроинструкции

Макроинструкция	Аргументы	Действие
`Add`	`X`	`a'[p] = a[p] + X`
`Move`	`X`	`p' = p + X`
`AddMove`	`X, Y`	`a'[p] = a[p] + X, p' = p + Y`
`MoveAdd`	`X, Y`	`p' = p + X, a'[p'] = a[p'] + Y`
`Clear`		`a'[p] = 0`
`Seek`	`X`	`p' = p + NX, a[p'] = 0`
`DAdd`	`X`	`a'[p] = 0, a'[p + X] = a[p + X] + a[p]`
`DSub`	`X`	`a'[p] = 0, a'[p + X] = a[p + X] - a[p]`

Таблица 4

Правила построения байткода

Макроинструкции	Следующая инструкция	Результат
`Move X`	`+/-`	`MoveAdd X, +/-1`
`MoveAdd X, Y`	`+/-`	`MoveAdd X, Y+/-1`
`Add X`	`+/-`	`Add X+/-1`
`?`	`+/-`	`Add +/-1`
`Add X`	`>/<`	`AddMove X, +/-1`
`AddMove X, Y`	`>/<`	`AddMove X, Y+/-1`
`Move X`	`>/<`	`Move X+/-1`
`?`	`>/<`	`Move +/-1`
`[+/-`	`]`	`Clear`
`[Move X`	`]`	`Seek X`
`[AddMove -1, X; AddMove 1, -X`	`]`	`DAdd X`
`[MoveAdd X, 1; MoveAdd -X, -1`	`]`	`DAdd X`
`[AddMove -1, X; AddMove -1, -X`	`]`	`DSub X`
`[MoveAdd X, -1; MoveAdd -X, -1`	`]`	`DSub X`

Примечание: знак вопроса обозначает правило по умолчанию, выполняемое если совпадение среди других правил не было найдено.

Для проверки быстродействия была использована программа [5], выводящая множество Мандельброта. Тестирование проводилось на компьютере на базе процессора Intel Core i7 920 @ 2.66 Ghz. Его результаты представлены в табл. 5.

Из результатов тестирования следует, что предложенные техники оптимизации дают значительный прирост производительности. Использование макрокоманд сложения-перемещения и деструктивного сложения дает почти двукратный выигрыш. Рассмотренный интерпретатор является одним из самых быстрых среди существующих виртуальных BF-машин. Об этом свидетельствуют результаты измерений, представленные в табл. 6.

Один из путей дальнейшего увеличения быстродействия состоит в добавлении новых макроинструкций. Однако дополнительная сложность реализации может не оправдаться приростом скорости. Другой путь состоит в реализации более эффективного метода передачи управления. Использование оператора *case* для выполнения очередной инструкции является относительно неэффективным. Существует более быстрая альтернатива — техника, известная как *direct threading* [9], способная обеспечить ускорение вплоть до двукратного. Однако она не может быть реализована на языке Pascal (как и на ANSI C).

Таблица 5

Влияние набора макроинструкций на производительность

Набор инструкций	Время выполнения программы mandelbrot.b, с	Относительная скорость, %	Прирост скорости, %
Наивное выполнение с предвычислением адресов	53,5	29	
`Add, Move`	15,6	100	
`Clear`	15,1	103	3
`Seek`	12,8	122	19
`AddMove, MoveAdd`	10,2	153	31
`DAdd, DSub`	7,3	213	60
Примечание: каждая строка соответствует набору, включающему все предыдущие инструкции плюс указанные.			

Таблица 6

Сравнение интерпретатора с существующими виртуальными BF-машинами

Интерпретатор	Время выполнения программы mandelbrot.b, с
Рассмотренный интерпретатор	7,3
Moderately-optimizing Brainf*ck language interpreter [6]	7,7
bff4 [7]	9,8
bff4-lnr [7]	10,8
Brainfuck compiler/optimizer/VM [8]	13,5
Примечание: компиляция исходных кодов проводилась при помощи компилятора GCC с максимальной оптимизацией.	

Заключение

При создании оптимизирующего интерпретатора языка Brainfuck весьма полезной оказывается техника объединения часто встречающихся последовательностей элементарных команд в макрокоманды, которые, в свою очередь, выполняются виртуальной машиной. Перевод исходного кода в такой байткод удобно реализуется при помощи набора правил, ставящих новую макроинструкцию в соответствие очередной инструкции исходного кода и нескольким предыдущим макроинструкциям.

В проведенных тестах использование простейшей оптимизации (сжатие последовательных команд +−<>) было в 3,4 раза быстрее, чем наивное выполнение. Использование более сложных макроинструкций позволило увеличить вычислительную производительность еще более чем в два раза, что превосходит другие существующие интерпретаторы. Большой вклад внесли инструкции объединенного сложения-перемещения и деструктивного сложения. Этот факт — результат уменьшения затрат на передачу управления в первом случае и уменьшения числа операций во втором.

Статистический анализ BF-программ показал, что большая часть циклов представляют собой одну из следующих операций: очистка, поиск, деструктивное сложение/вычитание, деструктивное умножение. Следовательно, расширение набора макроинструкций за счет каких-либо других операций вряд ли может принести значительный выигрыш. Дальнейшее ускорение возможно за счет реализации более эффективной техники передачи управления (*direct threading* или *call threading*) и оптимизаций на уровне ассемблера.

Список литературы

1. **Raiter B.** Portable Brainfuck. URL: http://www.muppetlabs.com/~breadbox/bf/standards.html.
2. **Faase F.** BF is Turing-complete. URL: http://www.iwriteiam.nl/Ha_bf_Turing.html.
3. **Esotope** Brainfuck compiler. URL: http://code.google.com/p/esotope-bfc/wiki/Comparison.
4. **Piumarta I., Riccardi F.** Optimizing direct threaded code by selective inlining // Proc. of the ACM SIGPLAN 1998 conference on Programming language design and implementation. ACM Press, 1998. P. 291—300.
5. **Bosman E.** A mandelbrot set fractal viewer in brainf***. URL: http://esoteric.sange.fi/brainfuck/bf-source/prog/mandelbrot.b.
6. **Pankratov A.** Moderately-optimizing Brainf*ck language interpreter. URL: http://www.swapped.cc/bff/.
7. **Oleg Mazonka's** personal web page. URL: http://mazonka.com/brainf/index.html.
8. **Brainfuck** compiler / optimizer / VM. URL: http://remcobloemen.nl/2010/brainfuck-vm.html
9. **Ertl M. A., Gregg D.** The Behavior of Efficient Virtual Machine Interpreters on Modern Architectures // Proc. of the 7th European Conference on Parallel Computing (Europar 2001). Manchester, August 2001. P. 403—412.

CONTENTS

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т.Н. Погорелова*. Технический редактор *Е.М. Патрушева*. Корректор *Е. В. Комиссарова*

Сдано в набор 14.06.2012 г. Подписано в печать 25.07.2012 г. Формат 60×88 1/8. Заказ PI612
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.