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 004.7

В. А. Васенин[1], д-р физ.-мат. наук, проф., e-mail: vasenin@msu.ru,
В. Ю. Лёвин[2], канд физ.-мат. наук, инженер-программист, e-mail: levval@yandex.ru,
Ф. М. Пучков[1], канд физ.-мат. наук, стар. науч. сотр., e-mail: fedormex@gmail.com,
К. А. Шапченко[1], канд физ.-мат. наук, стар. науч. сотр., e-mail: shapchenko@iisi.msu.ru
[1]НИИ механики МГУ,
[2]Научно-технологический центр "Электронтех" РАН

К созданию защищенной наложенной сети связи нового поколения для передачи данных

Представлены результаты предпроектных исследований по созданию сети связи нового поколения, обеспечивающей повышенный уровень защищенности данных в распределенных автоматизированных системах, использующих сети связи общего пользования.

Ключевые слова: *информационная безопасность, распределенные системы, наложенная сеть связи, топология сети связи, децентрализация, скрытие факта передачи данных*

Задача создания защищенной среды передачи данных в условиях, когда используется объективно недоверенная коммуникационная интернет-среда, является одной из важнейших и востребованных в сфере информационной безопасности автоматизированных систем. Актуальной является защита как от нарушения конфиденциальности и целостности данных, передаваемых между отдельными узлами (элементами) распределенной автоматизированной системы, так и обеспечение надлежащего высокого уровня доступности сервисов, поддерживающих такую передачу [1—3]. Отметим, что под конфиденциальностью в контексте данной статьи будем понимать не только конфиденциальность самих данных, но и отсутствие возможности раскрыть, например, факт их передачи, объем переданных данных или другую информацию подобного рода.

Уровень гарантий защищенности и надежности передачи данных в широко используемых на практике традиционных средствах связи на основе стека протоколов TCP/IP недостаточно высок. Подобные средства, включая их протокольную базу, изначально не содержат механизмов поддержки таких гарантий [3, 4]. Этот недостаток относится как к решениям, ориентированным на отдельных пользователей, так и к решениям, используемым в крупных распределенных системах. Среди последних следует отметить корпоративные системы электронной почты и мгновенных сообщений, а также механизмы обмена данными в информационно-вычислительных системах на основе технологий *Grid Computing* и *Cloud Computing*. Вместе с тем, для тех из перечисленных классов систем, которые предназначены для автоматизации технологических процессов, протекающих в критически важных для государства объектах и инфраструктурах, решение рассматриваемой задачи является одной из главных целей в ходе их разработки и на этапе эксплуатации [1, 3].

Недостатки традиционных решений. Среди недостатков традиционных методов, технологий, средств и систем, которые используются в защищаемой среде передачи данных, выделим следующие:

- отсутствие доверия к поставщикам услуг связи;
- централизованная архитектура, нарушение работы корневых (центральных) элементов в которой приводит к отказу всей системы;
- отсутствие доверия к оконечным (терминальным) устройствам.

Рассмотрим перечисленные недостатки подробнее. В современных условиях информационного взаимодействия передача конфиденциальных данных, как правило, проводится с использованием (полным или частичным) сетей связи общего пользования (например, таких сетей в составе единой сети электросвязи РФ). Основу данных сетей в силу объективных причин составляют интернет-протоколы, которым, как отмечалось ранее, свойственен относительно низкий уровень защиты от злоумышленных действий. Эти действия могут быть направлены на перехват и

изменение (модификацию) информации, причем как содержательно-прикладной, так и служебной, а также на нарушение штатного режима функционирования автоматизированной системы, которая использует подобные сети. В числе последних имеются в виду атаки на отказ в обслуживании, например, с помощью исчерпания коммуникационных и вычислительных ресурсов, задействованных в передаче, приеме и обработке данных.

Отсутствие доверия к сетям связи общего пользования обусловлено как рядом их архитектурно-технологических особенностей (например, отсутствием гарантий на путь прохождения данных и уязвимостями систем установки связи "имя—адрес"), так и тем обстоятельством, что сам поставщик услуг (или представляющая его персона) может осуществлять злоумышленные действия. Кроме того, при использовании напрямую каналов связи общего пользования факт наличия некоторой "содержательной" коммуникации остается открытым и в случае, если само содержание злоумышленнику раскрыть не удается.

Отметим, что наличие только лишь одного поставщика услуг связи даже при использовании специализированных сетей является фактором, способствующим появлению единой точки отказа в силу отсутствия гарантий на качество обслуживания, имея в виду само сетевое соединение и его пропускную способность. Тем не менее, несмотря на перечисленные недостатки сети, связи общего пользования представляют собой на настоящее время объективно высоко востребованную и готовую к использованию коммуникационную инфраструктуру, полноценной и экономически целесообразной альтернативы которой пока не существует.

Другим аспектом традиционных решений для передачи данных, который снижает их надежность и защищенность, является централизация сетевой инфраструктуры и сервисов, задействованных в процессе передачи данных. Признаками централизованной иерархической структуры обладают следующие компоненты современных распределенных систем:

— механизмы маршрутизации трафика;

— сервисы системы имен;

— инфраструктура распределения криптографической ключевой информации;

— системы управления базами данных;

— системы электронной почты;

— системы обмена мгновенными сообщениями.

Такие компоненты являются едиными точками отказа в централизованной архитектуре распределенной системы. Этот факт означает, что при компрометации подобных компонентов система утрачивает свою функциональность, и, как следствие, снижается уровень доверия к ней. Отметим, что простое автоматическое дублирование данных и сервисов в полном объеме не решает задачу обнаружения и своевременного отключения (и последующей замены) скомпрометированных узлов, находящихся достаточно высоко в централизованной иерархии компонентов системы. В процессах распределения и совместной выработки криптографических ключей это обстоятельство означает наличие возможности для проведения атак вида "человек посередине" с последующим раскрытием всей конфиденциальной корреспонденции в рамках сеанса общения. В процессе передачи самих сообщений перехват служебных данных обычно приводит к обнаружению факта передачи.

Оконечные устройства, задействованные в традиционных системах передачи данных, как правило, не являются доверенными. Такие устройства обычно представляют собой персональные компьютеры и средства мобильной связи под управлением незащищенных (не обладающих должными механизмами защиты) операционных систем. На подобных устройствах может быть установлено вредоносное программное обеспечение либо они могут быть скомпрометированы иным образом. Пользовательские компоненты системы передачи конфиденциальных данных являются при этом всего лишь одними из множества программ на этом устройстве. В таком случае, если злоумышленник получает привилегии на таком устройстве не ниже привилегий пользователя системы передачи данных, он сможет перехватить и прочитать либо изменить конфиденциальную корреспонденцию, или иным образом нарушить штатный режим функционирования системы передачи данных. По той же причине, а именно в силу недоверенной реализации и отсутствия контроля за терминальным устройством, не представляется возможным реализовать достаточно сильную аутентификацию конечных пользователей.

В настоящее время активно, в том числе и в распределенных системах нового поколения (*Grid*, *Cloud Computing*), используется ряд решений для организации защищенной передачи данных, которые, однако, не могут полностью устранить перечисленные выше недостатки. Средства организации VPN и криптомаршрутизаторы, например, сами по себе не скрывают факта передачи данных. Средства организации защищенной электронной почты и защищенного мгновенного обмена сообщениями функционируют через сервисы передачи данных и сервисы распределения ключей, которые имеют централизованную архитектуру. Кроме того, такие средства, как правило, функционируют на недоверенных программно-аппаратных платформах, что еще больше повышает риск компрометации подобной системы передачи данных.

Получившие достаточно широкую популярность сети анонимизации, подобные Tor [5] и I2P [6], также обладают рядом недостатков. Во-первых, они, как правило, не скрывают факта передачи данных и размеров передаваемых пакетов. Известно, что подобные статистические скрытые каналы в некоторых ситуациях могут привести к раскрытию (может быть — частичному) содержательной части коммуникаций, в том числе в каналах с криптографическим кодированием передаваемых данных [7]. Во-вторых, использование пользовательских приложений в Tor-подобных сетях посредством протокола SOCKS proxy небезопасно. Причина в том, что такие приложения сами по себе

представляют угрозу деанонимизации пользователя (например, веб-браузер вследствие сохранения сессий и cookies) [8, 9]. В-третьих, Tor и подобные ему сети неизбежно сталкиваются с вопросом "выходного узла": последний узел в цепочке снимает последний слой шифрования в схеме каскадной маршрутизации и получает информацию о непосредственном получателе сообщения. Согласно известным исследованиям, для деанонимизации пользователя в Tor-сети с некоторой конечной вероятностью достаточно уже одного скомпрометированного узла [10, 11].

Таким образом, надежность и защищенность традиционно используемых на практике систем передачи данных представляется недостаточно высокой. Перечисленные выше недостатки систем передачи данных сложно устранить используя только традиционные методы информационной безопасности автоматизированных систем. По этой причине далее предлагается краткое описание новой системы передачи данных. Эта система, будучи реализованной на основе представляемой далее концепции и набора доверенных оконечных устройств, позволит получить более высокие уровни гарантий надежности и защищенности обмена конфиденциальными сообщениями.

Постановка задачи. Для полного или частичного устранения перечисленных недостатков предлагается создать систему передачи данных на основе "наложенной" ("оверлейной") сети, которая использует в качестве опорной (транспортной) сети нижележащего уровня совокупность традиционных IP-сетей (например, сетей в составе единой сети электросвязи РФ).

Задачей предлагаемой наложенной сети является реализация и поддержка служебных функций для защищенной передачи конфиденциальных данных. Такая сеть должна удовлетворять следующим требованиям:

- в качестве транспортной среды должны использоваться доступные ресурсы сетей связи общего пользования на основе стека протоколов TCP/IP;
- должна быть реализована децентрализованная структура для базовых компонентов наложенной сети, включая подсистемы маршрутизации, выработки и распределения криптографических ключей, определения связи "ключ—имя—текущий адрес";
- децентрализованные подсистемы наложенной сети должны продолжать работать в штатном режиме при компрометации части узлов систем, сохраняя при этом возможность управления системой передачи данных в целом;
- должна быть предусмотрена возможность динамически изменять состав узлов сети (добавлять узлы и отключать узлы) с сохранением выполнения перечисленных ранее требований;
- должна быть предусмотрена возможность использовать в качестве оконечных устройств выделенные специализированные программно-аппаратные решения с повышенным уровнем защищенности.

Основной функциональной возможностью, которая предоставляется системой передачи данных ее пользователям, является электронная почта с подтверждением получения, включая функции отправки, получения и хранения электронных сообщений и их атрибутов.

Система передачи данных должна обладать следующими режимами передачи:

- передача сообщений (электронной почты);
- передача потоковых данных (например, аудио- и видеопотоков, телеметрии);
- передача данных в рамках протокола SOCKS proxy (для взаимодействия с традиционными сервисами в TCP/IP-сетях).

Методы и технологии решения. Основная идея, положенная в основу архитектуры создаваемой наложенной сети, заключается в многократном дублировании тех компонентов, которые в традиционных системах, как правило, представлены в единственном числе. При этом используется не "слепая" репликация сервисов системы, а их дублирование с поддержкой голосования большинством при появлении спорных ситуаций. Кроме того, периодически проводится изменение множества продублированных ключевых компонентов ("ядра" системы).

При таком подходе сохраняется управление построенной децентрализованной системой за счет следующих ее особенностей:

- автоматическое поддержание корректного функционирования наложенной сети и системы передачи сообщений при условии, что скомпрометировано не более некоторого заданного числа узлов "ядра" (не более половины);
- автоматизированное обновление программного кода ("прошивки") устройств при условии корректного функционирования наложенной сети;
- автоматизированная повторная доверенная инициализация наложенной сети и системы передачи сообщений с использованием альтернативных каналов связи.

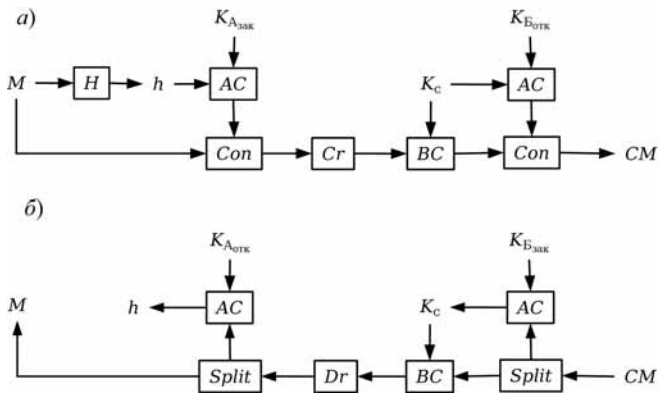
С точки зрения используемых технологических решений предлагаемая система должна включать следующие компоненты:

- наложенная сетевая инфраструктура с механизмами защищенной рандомизированной каскадной маршрутизации;
- система для распределения ключевой информации, устойчивая к полной компрометации значительной части узлов сетевой инфраструктуры;
- доверенная схема передачи данных с использованием зарекомендовавших себя методов асимметричной и симметричной криптографии;
- наличие нескольких видов терминальных устройств, в том числе устройств с высоким уровнем доверия, а именно — со своей аппаратной и программной частью, включая усиленные механизмы аутентификации.

Первые два компонента позволяют создать сетевую инфраструктуру, в которой удается скрыть факт передачи данных и организовать автоматическое дублирование маршрутов доставки сообщений для повы-

шения надежности передачи при условии использования нескольких поставщиков услуг связи. Второй и третий компоненты дают возможность организовать традиционную схему передачи данных, дополнив ее методами распределения ключей, более устойчивыми к компрометации части узлов сети передачи данных. С помощью последнего компонента — доверенных терминальных устройств — организуется возможность подключения к сети доверенной передачи данных в условиях разного уровня контроля пользователя над сетевой и компьютерной инфраструктурой. Спектр таких устройств может простираться от программных комплексов, предназначенных для запуска на защищенных операционных системах и системах общего назначения, до отдельных аппаратных решений.

Архитектура распределенной системы для защищенной передачи сообщений. Каждый узел, подключенный к системе защищенного обмена сообщениями, автоматически становится участником информационного обмена. Топология сети должна предусматривать автоматическое динамическое подключение и отключение узлов от сети. Основной единицей передачи данных системы защищенного обмена сообщениями является пакет. Размер пакета выбирается случайным образом в пределах некоторого интервала, исходя из требований к задержке и пропускной способности при передаче сообщения от одного узла к другому. Каждый узел в сети с определенной периодичностью обменивается пакетами с соседними в текущей топологии узлами. Частота рассылки пакетов для каждого отдельного узла выбирается в зависимости от качества



Используемые схемы шифрования сообщений:

а — схема посылки сообщения (отправляющая сторона А); *б* — схема получения сообщения (получающая сторона Б); *M* — исходное открытое сообщение; *H* — однонаправленная хеш-функция; *h* — хеш-значение; *AC* — асимметричная криптосистема; *Con* — модуль конкатенации; *Split* — модуль разделения данных на две части; *Cr* — модуль упаковки (сжатия) данных; *Dr* — модуль распаковки данных; *BC* — блочный шифр; *CM* — кодированное, готовое к передаче сообщение; $K_{Аотк}$ — открытый ключ стороны А; $K_{Азак}$ — закрытый (личный) ключ стороны А; $K_{Ботк}$ — открытый ключ стороны Б; $K_{Бзак}$ — закрытый (личный) ключ стороны Б; *KC* — сеансовый ключ блочного шифрования

его соединения с общей транспортной сетью (например, с сетью Интернет).

Пакет представляет собой подписанный отправителем и зашифрованный открытым ключом получателя запрос. При получении пакета узел проверяет его цифровую подпись, затем расшифровывает и обрабатывает содержащийся в нем запрос. Запросы могут быть двух видов: конечные и промежуточные. Промежуточный запрос представляет собой инструкцию вида "отправить данные соседнему узлу" с параметрами "данные" и "узел-получатель". Конечный запрос представляет собой запрос на получение информации об изменениях топологии сети, о соответствии имен, ключей и внутренних адресов. Промежуточные запросы генерируются, когда один пользователь инициирует отправку сообщения другому. При отправке сообщения узел-отправитель случайным образом выбирает в текущей топологии маршрут до узла-получателя. Регулярная структура топологии сети позволяет упростить эту задачу.

Конфиденциальность в реализации предложенной схемы обмена сообщениями реализуется путем использования зарекомендовавших себя на практике схем шифрования сообщений, передаваемых по сетям связи общего пользования, показанных на рисунке.

Опишем состав основных модулей. Модуль *H* по построению может состоять из произвольной односторонней криптографической функции (MAC, MDC [12, 13]), такой как MD4, MD5, RIPEMD, HAVAL, SHA1, SHA2. Однако имеющаяся тенденция к накоплению статического материала, развитие дифференциального и линейного криптоанализа, а также фиксированная структура построения хеш-функций могут позволить злоумышленнику найти способ построения коллизий. Для предотвращения таких ситуаций разработан мультиключевой алгоритм расчета хеш-значения, основанный на модифицированных стандартах блочного шифрования (ГОСТ 28147—89, DES), являющихся сетями Фейстеля [11]. Введение параметрического семейства латинских квадратов позволило ввести в *S*-блоки зависимость от дополнительного ключа. *S*-блоками будем называть таблицы перестановок битов в описании блочного шифра, по аналогии того, как *S*-блоки вводились в описании DES, ГОСТ 28147—89 [11]. В результате, без потери быстродействия число ключей соответствующей MAC-функции, основанной на сети Фейстеля, увеличено в несколько раз. Отмеченное обстоятельство позволило разработать мультиключевой алгоритм построения однонаправленных хеш-функций, в котором ключ может обновляться на каждом раунде, что в свою очередь позволило существенно увеличить порядок ключевого множества.

Формально, пусть E_K — алгоритм блочного шифрования с ключом K, E_K: $\{0,1\}^n \to \{0, 1\}^n$ — функция шифрования сообщения, $n \in \{1, 2, 3, ...\}$ — длина сообщения в битах, $K \in \{0, 1\}^r$, $r \in \{1, 2, 3, ...\}$ — длина ключа в битах. Тогда хеш-значение $h(M)$, где h: $\{0, 1\}^* \to$ $\to \{0, 1\}^n$ — хеш-функция порядка n, $M \in \{0, 1\}^*$ — пер-

воначальное сообщение, вычисляется по правилу: $H_0 = 0$; $H_i = E_K(M_i + \mathrm{H}_{i-1})$, $i = 1, ..., N$, $h(M) = H_N$.

Представленный итерационный процесс введен в работе [12], где предполагается, что первоначальное сообщение M разбито на N блоков M_i одинаковой длины (при этом последний блок определенным образом достраивается до необходимой длины). Заметим, что представленный способ построения хеш-функций является режимом шифрования со сцеплением блоков с той лишь разницей, что в качестве результата берется не весь шифрованный текст, а только последнее значение H_N. Рассмотрим вместо статической системы с хеш-функцией h на основе алгоритма шифрования E_K, где K — первоначальный r-битный ключ блочного шифрования, динамическую систему (конечный автомат) с последовательностью хеш-функций h_t, $t = 1, 2, ...$, каждая из которых строится описанным выше образом на основе функции шифрования $E_{k(t)}$: $\{0, 1\}^n \to \{0, 1\}^n$ с ключом $k(t) = f(K, k(t-1))$, $t = 1, 2, ...$, где $k(0) = K$ и f:$\{0, 1\}^r \times \{0, 1\}^r \to \{0, 1\}^r$ — некоторая булева функция порядка r. При таком подходе на каждом этапе итерационного процесса происходит смена блоков перестановок (S-блоков), зависящих от ключа $k(t)$. Используя в мультиключевом алгоритме переменные S-блоки, удалось многократно снизить эффективность использования линейного и дифференциального анализа для компрометации системы. Действительно, основываясь на исследовании трудов Шамира и Мацуми [12], установлено, что если S-блоки не являются фиксированными, то соответствующие виды анализов малоэффективны. Разработанные схемы применены для введения дополнительных ключей в предложенные алгоритмы подсчета хеш-значения, которые позволяют разработчику создать их иерархию при использовании в различных аппаратных решениях. Важно отметить, что применение переменных блоков (мультиключевого подхода) в целях шифрования требует детального изучения вопросов, связанных с наличием слабых ключей и оптимальностью выбора блоков замены. В противном случае схема будет иметь существенные недостатки.

Модуль *AC* представляет собой произвольную асимметричную систему шифрования, например, RSA. Модуль *Con* осуществляет конкатенацию двух блоков данных с сохранением информации о длинах объединяемых частей, чтобы в дальнейшем можно было осуществить обратную операцию — разбиение данных на две части, которое реализуется в модуле *Split*. Модуль *Cr* является модулем архивирования (сжатия и упаковки данных в некоторый контейнер), например, алгоритмом zip. Модуль *Dr* осуществляет обратную по отношению к *Cr* операцию — распаковку данных. Модуль *BC* представляет собой реализацию отечественного стандарта блочного шифрования ГОСТ 28147—89.

При пересылке сообщения от одного узла к другому по цепочке, состоящей из нескольких промежуточных узлов, отправитель использует многослойное шифрование передаваемого сообщения. При этом каждый промежуточный узел в цепочке передачи снимает свой слой шифрования, после чего определяет следующего адресата. Таким образом:

— исходное сообщение может прочитать только конечный получатель;

— каждый промежуточный узел в цепочке знает только предыдущего и последующего участников;

— ни один промежуточный узел не знает конечного получателя и отправителя сообщения.

Такая схема передачи пакетов затрудняет идентификацию отправителя и получателя сообщения, обеспечивая тем самым конфиденциальность факта переписки. В силу того, что все пакеты имеют случайную длину и структуру с точки зрения внешнего наблюдателя, а также, поскольку частота их появления в сети постоянна, статистический профиль при пересылке информационного сообщения не отличается от профиля при пересылке любого другого системного сообщения. Тем самым ликвидируется скрытый статистический канал в сети, возникающий при активном общении двух пользователей, который характерен для других аналогичных сетей (например, Tor и I2P). Таким образом, обеспечивается сокрытие факта передачи сообщения.

Целостность передаваемых сообщений обеспечивается при использовании электронной цифровой подписи на основе эллиптической кривой.

Каждый пакет, передаваемый в сети, представляет собой зашифрованный и подписанный ключом отправителя запрос. Каждый узел при получении пакета расшифровывает его и проверяет то, что цифровая подпись действительно принадлежит отправителю. Таким образом обеспечивается защита от несанкционированной подмены пакетов в сети.

Высокий уровень доступности сервисов в системе защищенной передачи сообщений обеспечивается ее децентрализацией. Он достигается путем многократного дублирования информации о ключах пользователей и о текущей топологии сети. Дополнительная нагрузка на каждый отдельный узел и затраты на хранение данных на нем при этом, как правило, оказываются относительно небольшими.

Предварительная оценка работоспособности решения. При значительном числе узлов, подключенных к системе защищенной передачи данных, актуальным является вопрос оценки масштабируемости предлагаемого решения, включая оценки максимального и среднего времени задержки, а также пропускной способности канала при передаче сообщения от одного узла к другому. Эта задача становится тем более актуальной, что сообщения в системе передаются по цепочке, потенциально состоящей из большого числа промежуточных узлов с ограниченной пропускной способностью. Каждый такой узел вносит свою задержку в процесс обработки пакета (например, расшифрование внешнего контейнера пакета и проверка подписи). Топологию сети для защищенной передачи сообщений будем считать масштабируемой, если выполнены следующие условия:

Таблица 1

Результаты аналитического расчета коэффициентов изменения времени задержки и пропускной способности при использовании сетевой топологии многомерного тора

Топология многомерного тора	Размерность тора n	Длина стороны тора k	Число узлов $N = k^n$	Коэффициент увеличения времени задержки $c_L = n^2k$	Коэффициент изменения пропускной способности $c_{BW} = 1/(2n)$
10×10	2	10	100	40	1/4
10×10×10	3	10	1000	90	1/6
10×10×10×10	4	10	10 000	160	1/8
10×10×10×10×10	5	10	100 000	250	1/10
10×10×10×10×10×10	6	10	1 000 000	360	1/12

- максимальная задержка при передаче сообщения ограничена степенью логарифма от числа узлов в сети;
- сложность операции подключения/отключения узла ограничена степенью логарифма от числа узлов в сети.

Примером масштабируемой топологии сети является многомерный дискретный тор. В табл. 1 представлены аналитически полученные значения коэффициентов изменения максимальной задержки и минимальной пропускной способности канала передачи данных в зависимости от числа узлов для указанной топологии.

Децентрализованная архитектура системы защищенной передачи сообщений позволяет добиться устойчивости к полной компрометации части своих узлов (до любого их количества, строго меньшего 50 % от числа узлов, например, до 40 %). Это обеспечивается путем проведения автоматических голосований большинством по вопросам маршрутизации и управлением ключами. До тех пор, пока число скомпрометированных узлов в сети не превышает допустимую верхнюю границу, система будет функционировать в штатном режиме, обеспечивая доверенную маршрутизацию и безопасное управление ключами. Голосование может проводиться в следующих двух режимах.

Режим 1. В голосовании участвуют все узлы сети. Такой режим имеет смысл, когда число всех узлов невелико (меньше 1000). Он обеспечивает гарантированную достоверность результатов голосования при условии, что количество скомпрометированных узлов меньше 50 %.

Режим 2. В голосовании участвует только некоторое "ядро" — случайно выбранное подмножество узлов сети. Такой режим предназначен для использования в ситуации, когда число всех узлов велико (больше 1000). Вероятность события "не менее половины узлов в ядре скомпрометированы" при условии независимого равномерного распределения скомпрометированных узлов может быть вычислена по следующей формуле:

$$P(N, m, r) = \Sigma_{i = [(r + 1)/2], \ldots, m} C(i - 1, [(r + 1)/2] - 1) C(N - i, r - [(r + 1)/2] / C(N, r),$$

где N — общее число узлов, m — число скомпрометированных узлов; r — число узлов в ядре; $C(n, k)$ — число сочетаний из n элементов по k; $[n]$ — целая часть от n.

Достоверность результатов голосования обеспечивается с любой наперед заданной вероятностью в случае, когда количество скомпрометированных узлов в сети не превосходит 40 %. В табл. 2 для нескольких вариантов размера ядра приведены результаты расчета по представленной выше формуле вероятности того события, что не менее половины узлов в ядре скомпрометированы (при условии, что до 40 % узлов могут быть скомпрометированы).

Таким образом, вероятность "плохого голосования" убывает достаточно быстро с ростом числа узлов в ядре. Ядро, состоящее из 1000 узлов, можно считать абсолютно надежным для голосования большинством при увеличении общего размера сети до 1 000 000 узлов и количестве скомпрометированных узлов, не превосходящем 40 %.

Таблица 2

Результаты аналитического расчета вероятности компрометации не менее половины узлов ядра

Число узлов в ядре, r	Вероятность события «не менее половины узлов в ядре скомпрометированы», $P(1\,000\,000, 400\,000, r)$, %
100	≈2,11
200	≈0,09
300	≈0,0018
400	≈0,00001

Оконечные устройства. Каждое оконечное устройство позволит пользователю участвовать в нескольких режимах обмена сообщениями, предоставляя, например, возможность отказа от защиты факта передачи в пользу более высокой пропускной способности защищенного канала передачи данных.

Выделенные аппаратные оконечные устройства оснащаются интерфейсами для подключения к сетям связи и предоставляют устройства ввода-вывода для ввода и отображения передаваемых и получаемых сообщений, а также для проведения работ по обслуживанию устройства. Оконечное устройство такого вида может комплектоваться интерфейсами для подключения устройств хранения в целях организации передачи находящихся на них данных, а также интерфейсом для подключения к персональному компьютеру. С помощью последнего интерфейса может быть организован процесс передачи сообщений. В рамках этого процесса сообщение готовится на персональном компьютере, а непосредственно отправка осуществляется на выделенном устройстве с предварительным просмотром и подтверждением текста отправляемого сообщения.

Выводы. Предлагаемая к разработке система для защищенной передачи данных позволит устранить ряд недостатков, характерных для традиционных компьютерных систем подобного назначения. Предлагаемая система предоставляет возможности выполнять типовые задачи обмена электронными сообщениями, но в более защищенном исполнении по сравнению с традиционными решениями. Она позволяет действовать с использованием имеющихся коммуникационных ресурсов и предусматривает возможность выбора баланса между надежностью и скоростью передачи при условии вынужденных ограничений на каналы связи. С учетом представленных оценок масштабирования и устойчивости к возможной компрометации отдельных узлов, предлагаемое решение целесообразно использовать в территориально-распределенных информационных системах корпоративного масштаба и государственного уровня, в особенности если ставится требование сокрытия факта передачи данных.

Особенно актуально использование предлагаемого подхода для крупномасштабных, сложно организованных, распределенных критических систем нового поколения [1, 2], построенных по методологии *Grid Computing и Cloud Computing*. С одной стороны, поставленная выше задача для таких систем является основной, с другой стороны, на настоящее время она не имеет решения (по крайней мере — открытого) на традиционно используемых для этого методах, технологиях и инструментальных средствах.

Список литературы

1. **Критически** важные объекты и кибертерроризм. Часть 1. Системный подход к организации противодействия. / О. О. Андреев и др. Под ред. В. А. Васенина. М.: МЦНМО. 2008. 398 с.
2. **Критически** важные объекты и кибертерроризм. Часть 2. Аспекты программной реализации средств противодействия / О. О. Андреев и др. Под ред. В. А. Васенина. М.: МЦНМО. 2008. 607 с.
3. **Васенин В. А.** Проблемы безопасности информационных технологий: анализ "узких мест" // Научные основы национальной безопасности Российской Федерации. Материалы семинара. Комитет Совета Федерации по обороне и безопасности. 24 мая 2005 г. С. 70—74.
4. **Васенин В. А.** Проблемы математического, алгоритмического и программного обеспечения компьютерной безопасности в Интернет // Математика и безопасность информационных технологий. Материалы конференции в МГУ 23—24 октября 2003 г. М.: МЦНМО. 2004. С. 111—141.
5. **Tor Project:** Anonymity Online. The Tor Project, Inc. URL: https://www.torproject.org/
6. **I2P** Anonymizing Network. URL: http://www.i2p2.de/
7. **White A., Matthews A., Snow K., Monrose F.** Phonotactic Reconstruction of Encrypted VoIP conversations: Hookt on fon-iks // Proc. of IEEE Symposium on Security and Privacy. May, 2011. URL: http://www.cs.unc.edu/~amw/resources/hooktonfoniks.pdf
8. **Abbott T. G., Lai K. J., Lieberman M., Price E.** Browser-based attacks on Tor // Proc. of the 7th international conference on Privacy enhancing technologies (PET'07). Ottawa. Canada. 2007. Berlin: Springer-Verlag. 2007. P. 184-199.
9. **Perry M.** Securing the Tor Network // Black Hat USA 2007. Conference. Las Vegas, NV, USA, July 28 — August 2. 2007. URL: https://www.blackhat.com/presentations/bh-usa-07/Perry/Whitepaper/bh-usa-07-perry-WP.pdf
10. **Evans N. S., Dingledine R., Grothoff C.** A practical congestion attack on tor using long paths // Proceedings of the 18th conference on USENIX security symposium (SSYM'09). USENIX Association. Berkeley, CA, USA. 2009. P. 33—50.
11. **Fu X., Ling Zh.** One Cell is Enough to Break Tor's Anonymity // Black Hat DC 2009 Conference. Arlington, VA, USA, February 16—19. 2009. URL: https://www.blackhat.com/presentations/bh-dc-09/Fu/BlackHat-DC-09-Fu-Break-Tors-Anonymity.pdf
12. **Шнайер Б.** Прикладная криптография. 2-е издание. Протоколы, алгоритмы и исходные тексты на языке С. М.: Триумф, 2002. 595 с.
13. **Menezes A., Oorschot P., Vanstone S.** Handbook of Applied Cryptography. CRC Press, 1996.

УДК 004.056

В. А. Галатенко, д-р физ.-мат. наук., зав. сектором автоматизации программирования, НИИСИ РАН, г. Москва,

e-mail: galat@niisi.msk.ru

К автоматизации процессов анализа и обработки информационного наполнения безопасности: развитие протокола SCAP

Дано краткое описание того нового, что появилось в версии SCAP 1.2. (Протокол автоматизации процессов анализа и обработки информационного наполнения безопасности — Security Content Automation Protocol) [1, 2]. Информация о версии SCAP 1.1 была дана в работе [1].

Ключевые слова: *компьютерная безопасность, информационное наполнение безопасности, протокол автоматизации*

Структура спецификаций версии SCAP 1.2

Спецификации версии SCAP 1.2 включают в себя одиннадцать компонентов, разбитых на следующие пять групп.

- **Языки** (расширенная группа). В версии SCAP 1.2 языки предоставляют стандартные словари и соглашения для представления политики безопасности, механизмов технических проверок и результатов оценки. В эту группу входят три языка:
  - ♦ расширяемый формат описания конфигурационных контрольных перечней — *eXtensible Configuration Checklist Description Format* (XCCDF) 1.2 (язык для составления контрольных перечней и/или контрольных тестов и для генерации отчетов по результатам их применения);
  - ♦ открытый язык уязвимостей и оценки — *Open Vulnerability and Assessment Language* (OVAL) 5.10 (язык для представления информации о конфигурации систем, оценки состояния систем и для генерации отчетов по результатам оценки);
  - ♦ открытый интерактивный язык контрольных перечней — *Open Checklist Interactive Language* (OCIL) 2.0 (новый компонент — язык для представления проверок, собирающих информацию от системных администраторов, применяющих контрольные перечни, или из существующих хранилищ данных, построенных путем предшествующего сбора данных).
- **Форматы отчетов** (новая группа). Они обеспечивают создание конструкций, необходимых для представления собранной информации в стандартизованных форматах. В версии SCAP 1.2 в эту группу входят две спецификации:
  - ♦ формат отчетов о ресурсах — *Asset Reporting Format* (ARF) 1.1 (формат для представления транспортного вида информации о ресурсах и об отношениях между ресурсами и отчетами);
  - ♦ идентификация ресурсов — *Asset Identification* (AI) 1.1 (формат для уникальной идентификации ресурсов, основанный на известных идентификаторах и/или известной информации о ресурсах).
- **Перечисления**. Каждое перечисление в SCAP определяет стандартную номенклатуру (формат именования) и официальные словари или списки элементов, представленных с использованием этой номенклатуры. Имеется три спецификации перечисления:
  - ♦ общее платформное перечисление — *Common Platform Enumeration* (CPE) 2.3 (номенклатура и словарь аппаратуры, операционных систем и приложений);
  - ♦ общее конфигурационное перечисление — *Common Configuration Enumeration* (CCE) 5 (номенклатура и словарь конфигураций безопасности программного обеспечения);
  - ♦ общие уязвимости — *Common Vulnerabilities and Exposures* (CVE) (номенклатура и словарь уязвимостей).
- **Измерение и ранжирование уязвимостей** (расширенная группа). Содержит спецификации для измерения характеристик уязвимостей и генерации на этой основе количественной меры серьезности выяв-

ленных уязвимостей. В версии SCAP 1.2 таких спецификаций две:

- общая система оценки серьезности уязвимостей — *Common Vulnerability Scoring System* (CVSS) 2.0 (регламентирует измерение относительной серьезности (ранжирование) уязвимостей, вызванных программными дефектами);
- общая система оценки серьезности дефектов конфигурирования — *Common Configuration Scoring System* (CCSS) 1.0 (регламентирует измерение относительной серьезности (ранжирование) просчетов в конфигурации безопасности систем).

• **Целостность**. Помогает сохранить целостность информационного наполнения SCAP и результатов. В версии SCAP 1.2 в эту группу входит одна спецификация — модель доверия для данных автоматизации анализа и обработки информационного наполнения безопасности — *Trust Model for Security Automation Data* (TMSAD) 1.0 (спецификация использования цифровых подписей в общей модели доверия с учетом особенностей автоматизации анализа и обработки информационного наполнения безопасности).

Для определенных целей (функций), таких как контроль безопасности конфигураций, могут применяться комбинации перечисленных спецификаций. Совокупное информационное наполнение XML, используемое при этом, называется потоком данных SCAP. Под исходным потоком данных SCAP понимается входное информационное наполнение, а под результирующим потоком данных SCAP - выходное информационное наполнение. Основные элементы потоков данных называются компонентами потоков.

В SCAP используются эталонные данные из Национальной базы данных об уязвимостях (США) — *National Vulnerability Database* (NVD).

SCAP-совместимость

SCAP-совместимость необходима для продуктов и информационного наполнения безопасности.

SCAP-совместимые продукты подразделяются на два типа — поставщиков и потребителей информационного наполнения. Первые генерируют исходные потоки данных SCAP, в то время как вторые воспринимают существующие потоки данных SCAP, обрабатывают их и порождают результирующие потоки данных SCAP.

SCAP-совместимые продукты должны удовлетворять спецификациям на содержащиеся в них компоненты.

SCAP-совместимое информационное наполнение безопасности должно удовлетворять синтаксическим, структурным и другим требованиям спецификаций SCAP.

Отметим, что согласно спецификациям SCAP, потоки данных содержат не сами компоненты, а ссылки на них. Тем самым поддерживается повторное использование компонентов, допускаются внешние связи и их динамическое разрешение.

Модель доверия

Модель доверия [3] — необходимый элемент спецификаций SCAP. Эта модель служит основой для контроля целостности, аутентичности и прослеживаемости данных. Модель доверия может применяться также для управления доступом к потокам данных и для обеспечения прослеживаемости результатов, получения гарантий того, что результаты получены из доверенного источника.

Модель доверия в версии SCAP 1.2 строится путем конкретизации известных спецификаций, относящихся к XML и криптографии. При этом требования к поставщикам и потребителям информационного наполнения безопасности могут естественным образом различаться. Например, потребители должны нормально обрабатывать имитовставки в формате SHA-1, однако поставщикам рекомендуется использовать другие, более надежные разновидности этой криптографической схемы: SHA-256, SHA-384 или SHA-512.

Доверенное информационное наполнение должно быть подписано. В версии SCAP 1.2 допускается три типа подписей:

— отсоединенная;

— обернутая;

— обертывающая.

Отсоединенная электронная подпись хранится отдельно от подписанного информационного наполнения — например, в другом документе или в другой части того же документа. Отсоединенная электронная подпись полезна, когда нужно одной подписью заверить сразу несколько документов, или когда подпись предназначена лишь для внутреннего пользования и не должна передаваться вовне.

Обернутая электронная подпись помещается внутрь подписываемого документа (подписанное таким образом информационное наполнение включает в себя элемент, содержащий подпись). Обернутая электронная подпись обычно заверяет отдельные, автономные документы.

Обертывающая электронная подпись содержит внутри себя подписываемое информационное наполнение в качестве преемника узла <dsig:Object>. Чаще всего таким образом заверяется другая подпись.

Новое в общем платформном перечислении CPE версии 2.3

Общее платформное перечисление — это стандартизованный метод описания и идентификации аппаратно-программных ресурсов, таких как аппаратные устройства, операционные системы, программные приложения и т.п.

Обновленные спецификации общего платформного перечисления (CPE) версии 2.3 [4—7] реализуют идею разделения абстрактного и конкретного представления объектов.

Применительно к именованию (спецификация [4]), реализация отмеченной идеи означает введение понятия правильно сформированного имени — *Well Formed Name* (WFN) как логической конструкции, абстрактного способа именования и конкретных представлений имен, допускающих автоматическую обработку. В роли конкретных представлений могут выступать, например, универсальные идентификаторы ресурсов — *Universal Resource Identifier* (URI), что предусматривалось предыдущей версией спецификаций, а также отформатированные цепочки символов.

В общем случае под правильно сформированным именем понимается неупорядоченный набор пар (атрибут, значение). Спецификации CPE версии 2.3 предусматривают следующие атрибуты:

- часть;
- производитель;
- продукт;
- версия;
- выпуск;
- редакция;
- язык;
- программная редакция;
- целевое программное обеспечение;
- целевое аппаратное обеспечение.

Значения атрибутов могут быть логическими или представлять собой непустую цепочку символов.

Специфицированы два логических значения:

- ANY (любой);
- NA (не используется).

Для цепочек символов предусмотрена кодировка UTF-8, привычные метасимволы ("\", "*" и т. п.), а также допустимые значения для компонента "часть":

- "*a*" (программное приложение);
- "*o*" (операционная система);
- "*h*" (аппаратное устройство).

Введены два отображения: прямое, из абстрактного имени в конкретное (связывание) и обратное.

Кроме того, для правильно сформированных имен определены три функции:

- new ()— создать пустое (т. е. не содержащее ни одной пары (атрибут, значение)) имя;
- get (*w*, *a*) — для имени *w* выдать значение атрибута *a*;
- set (*w*, *a*, *v*) — установить для имени *w* значение атрибута *a* равным *v*.

В целом спецификации именования в CPE версии 2.3 можно рассматривать как далеко идущее (хотя и довольно очевидное) обобщение международного стандарта [8], регламентирующего именование программных сущностей.

Спецификация сопоставления имен CPE [5] определяет метод сравнения имен. Поскольку правильно сформированные имена представляют собой наборы, можно выяснить, выполняются ли для пары имени обычные теоретико-множественные отношения: равны ли два имени, является ли одно подмножеством другого и другие подобные им. Это полезно, например, для выяснения того, установлен ли на данной системе заданный программный продукт, удовлетворяет ли версия продукта определенным ограничением и для ответов на другие аналогичные вопросы.

В спецификации [5] сопоставляемые имена называются исходным и целевым, соответственно, а сам процесс сопоставления подразделяется на две фазы: сравнение атрибутов и сравнение имен.

На первой фазе в исходном и целевом именах сравниваются соответствующие пары (атрибут, значение). На второй фазе совокупность результатов попарных сравнений используется для определения общего результата сопоставления.

Спецификация [6] описывает стандартизованные методы для создания и администрирования CPE-словарей, содержащих CPE-имена и ассоциированные с ними метаданные. Каждое CPE-имя определяет единственный класс (а не множество классов) аппаратно-программных программных продуктов и представляет собой правильно сформированное имя в связанной форме, т. е. уникальный идентификатор ресурса или отформатированную цепочку символов. Это имя называется также идентифицирующим именем.

Вероятно, связанная форма CPE-имен в CPE-словарях объясняется соображениями обратной совместимости: понятие правильно сформированного имени появилось в спецификациях SCAP после того, как был создан официальный CPE-словарь [9].

Наличие официального CPE-словаря служит основой взаимной совместимости средств для работы с идентифицирующими именами. В первую очередь имена аппаратно-программных ресурсов отыскиваются в этом словаре. Организации могут создавать собственные, расширенные словари, например, для именования собственных аппаратно-программных продуктов, использующихся только в пределах определенной организации, или для новых продуктов, которые предназначены для широкого применения, но в силу своей новизны еще не попавших в официальный CPE-словарь.

Структура CPE-словаря определяется как XML-схема. Собственно CPE-словарь представляет собой список CPE-элементов. Каждый CPE-элемент включает идентифицирующее имя и ассоциированные с ним метаданные. Новым в CPE версии 2.3 стало поле источника (происхождения) элемента. В связи с тем, что аппаратно-программные продукты могут выходить из употребления или менять идентифицирующие имена, предусмотрена возможность делать элементы недействительными. В CPE-элементах предусмотрена также точка расширения.

Для CPE-словаря в целом и для отдельных CPE-элементов предусмотрены естественные операции (функции), такие как выборка и поиск, преобразование идентифицирующего имени к WFN-виду, выборка значений атрибутов и т. п. Можно предположить, что набор подобных функций, обязательных для реализации, будет расширяться.

Спецификация [7], называемая "Языком Применимости", определяет стандартизованный способ описания аппаратно-программных платформ путем формирования сложных логических выражений (с использованием логических операций "И", "ИЛИ", "НЕ" и скобок) как комбинации отдельных CPE-имен (например, имен операционной системы и/или программного приложения) и ссылок на проверки (например, на проверку того, что в операционной системе определенное аппаратное устройство активировано, или на проверки установок конфигурационных параметров: на OVAL-проверки, OCIL-проверки и т. д.). Эти логические выражения называются "утверждениями применимости", поскольку они обозначают применимые платформные руководства, политики и подобные им сущности. Утверждения применимости используются автоматизированными инструментальными средствами, чтобы определить, является ли целевая система частным случаем некоторой предварительно заданной платформы.

* * *

Спецификации, составляющие SCAP-протокол, активно развиваются. Они достигли состояния, допускающего практическое использование средствами информационной безопасности применительно к сложным, разнородным информационным системам. По мнению автора, пришло время внедрения SCAP-совместимых программных продуктов в практику крупных организаций для управления информационной безопасностью систем корпоративного уровня.

Список литературы

1. **Галатенко В. А.** К проблеме автоматизации анализа и обработки информационного наполнения безопасности // Программная инженерия. 2011. № 3. С. 45—47.
2. **Waltermire D., Quinn S., Scarfone K., Halbardier A.** The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.2 (Draft). Recommendations of the National Institute of Standards and Technology. U.S. Department of Commerce. National Institute of Standards and Technology. Special Publication 800-126 Revision 2 (Draft). NIST, July 2011.
3. **Booth H., Halbardier A.** Trust Model for Security Automation Data 1.0 (TMSAD) (Draft). U.S. Department of Commerce, National Institute of Standards and Technology, NIST Interagency Report 7802 (Draft). NIST, July 2011.
4. **Cheikes B. A., Waltermire D., Scarfone K.** Common Platform Enumeration: Naming Specification Version 2.3 (Draft). U.S. Department of Commerce, National Institute of Standards and Technology, NIST Interagency Report 7695 (Second Public Draft). NIST, April 2011.
5. **Parmelee M. C., Booth H., Waltermire D., Scarfone K.** Common Platform Enumeration: Name Matching Specification Version 2.3 (Draft). U.S. Department of Commerce, National Institute of Standards and Technology, NIST Interagency Report 7696 (Second Public Draft). NIST, April 2011.
6. **Cichonski P., Waltermire D., Scarfone K.** Common Platform Enumeration: Dictionary Specification Version 2.3 (Draft). — U.S. Department of Commerce, National Institute of Standards and Technology, NIST Interagency Report 7697 (Second Public Draft). — NIST, June 2011.
7. **Waltermire D., Cichonski P., Scarfone K.** Common Platform Enumeration: Applicability Language Specification Version 2.3 (Draft). U.S. Department of Commerce, National Institute of Standards and Technology, NIST Interagency Report 7698 (Initial Public Draft). NIST, June 2011.
8. **ISO/IEC 19770—2:2009**. Information technology — Software asset management — Part 2: Software identification tag.
9. **Официальный** CPE-словарь. URL: http://nvd.nist.gov/cpe.cfm

УДК 004.051

А. О. Удовиченко, аспирант, e-mail: aspudovichenko@mail.ru,
Н. Н. Пуцко, канд. техн. наук, доц.,
Московский государственный университет путей сообщения

Комплексная методика борьбы с эффектом "старения" ПО

Предлагается комплексная методика борьбы с эффектом "старения" ПО для информационной системы, построенной с применением технологии виртуальных машин. Рассматриваются основные компоненты методики и порядок ее работы. Приводятся результаты экспериментов, подтверждающие эффективность предложенной методики.

Ключевые слова: *программное обеспечение, "старение", восстановление, сервер, виртуальная машина, эффективность, методика*

Введение

В настоящее время задача обеспечения высокой эффективности функционирования информационных систем (ИС) приобретает все большую актуальность в результате широкого распространения информационных технологий. Достаточно часто от ИС, особенно в сфере электронного бизнеса, требуется доступность 24 часа в сутки, 365 дней в году.

Одной из распространенных причин нарушения работы ИС является эффект "старения" ПО (англ. *software aging*) [1], суть которого заключается в прогрессивной деградации работы приложений и увеличении числа их отказов с ростом времени непрерывной работы. Примерами данного эффекта являются утечка памяти, ошибки округления, утечка файловых дескрипторов. Борьба с эффектом "старения" ПО заключается в снижении его негативного воздействия на работу приложения на основе методики, получившей название "восстановление рабочего состояния ПО" [1—3]. Ее суть заключается в регулярном переводе приложения в рабочее состояние, близкое к первоначальному за счет проведения некоторой "очистки". "Очистка" может включать в себя сбор мусора, повторную инициализацию внутренних структур. Более радикальный метод предполагает полную перезагрузку операционной системы (ОС).

В качестве одной из ключевых технологий построения ИТ-инфраструктур специалистами рассматривается технология виртуальных машин. По результатам исследований аналитических агентств ожидается, что в ближайшие годы более половины всей серверной нагрузки будет обрабатываться виртуализированными серверами [4]. В настоящее время не существует законченного решения для борьбы с эффектом "старения" ПО в ИС, построенной на основе данной технологии. Применение существующих решений часто оказывается недопустимым в силу принципиального отличия новой технологии построения ИТ-инфраструктуры от классической. Нарушение работы управляющей виртуальной машины (УВМ), например, в результате восстановления на основе метода перезагрузки ОС УВМ, ведет к остановке всех серверов, размещенных на данном хосте.

Одной из главных трудностей борьбы с эффектом "старения" ПО является то, что существующие методы восстановления рабочего состояния приложения обладают высокими издержками, связанными обычно с нарушением работы пользователей и остановкой приложения в процессе восстановления. Таким образом, возникает необходимость планирования процессов восстановления приложения с учетом негативного воздействия на его работу как эффекта "старения" ПО, так и процессов его восстановления. В результате задача снижения негативного воздействия эффекта "старения" ПО на эффективность работы ИС разделяется на несколько подзадач:

- Восстановление рабочего состояния приложения. Цель восстановления заключается в переводе приложения в рабочее состояние, наиболее близкое к первоначальному, и минимизации влияния данного процесса на его работу.
- Определение времени запуска процесса восстановления. Основной целью является минимизация издержек, связанных с негативным воздействием на работу приложения эффекта "старения" ПО и процессов его восстановления.
- Согласование процессов восстановления. Решение данной задачи предполагает определение параметров процессов восстановления с учетом их взаимного влияния и текущего состояния ИС, например, доступного объема ресурсов.

Стоит отметить, что выделенные подзадачи должны рассматриваться комплексно и преследовать одну цель — снижение негативного воздействия эффекта "старения" ПО на эффективность работы ИС.

Методика борьбы с эффектом "старения" ПО

Решения по снижению негативного воздействия эффекта "старения" ПО на работу ИС представляют собой комбинации методов восстановления [2, 3] и методов определения времени восстановления [5, 6]. Первая группа методов обеспечивает перевод приложения в рабочее состояние. Вторая группа методов отвечает за определение времени восстановления в целях снижения негативного воздействия эффекта "старения" ПО и процессов восстановления на работу приложения. Существуют также и специализированные программные продукты по снижению негативного воздействия эффекта "старения" ПО на работу ИС [7], которые обычно включают в себя реализацию нескольких методов из каждой группы и инструменты, обеспечивающие удобство настройки и управления процессами восстановления. Соответственно, недостатки и достоинства этих продуктов определяются используемыми в них методами. Основными трудностями борьбы с эффектом "старения" ПО являются, во-первых, недостатки методов восстановления, основной из которых — остановка приложения в процессе восстановления, и, во-вторых, отсутствие эффективных методов управления процессами восстановления.

В данной работе предлагается методика борьбы с эффектом "старения" ПО для ИС, построенной на основе технологии виртуальных машин. Данная методика включает разработку новых методов определения времени восстановления приложения, методов восстановления и управления процессами восстановления.

Технология виртуальных машин обеспечивает функционирование нескольких серверов на одном компьютере с уровнем изоляции, с точки зрения совместимости приложений, близким к уровню изоляции отдельных физических компьютеров. Виртуальная машина (ВМ) представляет собой программный контейнер, в котором установлена ОС со всеми необходимыми службами и приложениями. Мониторинг и управление ВМ осуществляется специальными приложениями, размещенными внутри УВМ. Данные особенности технологии позволяют выделить объекты, наиболее уязвимые с точки зрения эффекта "старения" ПО, и решать задачу восстановления индивидуально для каждого из них с объединением полученных результатов в законченное решение. Наиболее критичными к воздействию эффекта "старения" ПО в ИС, построенной с использованием технологии виртуальных машин, являются:

- Логический сервер (объект 1). Под логическим сервером будем понимать некоторое приложение, размещенное внутри гостевой ВМ и предоставляющее услуги пользователям через сеть. К работе сервера обычно предъявляются высокие требования, часто доступность 24 часа в сутки, 365 дней в году.
- Управляющая виртуальная машина (объект 2). Данный объект отвечает за мониторинг и управление всеми ВМ на хосте, нарушение его работы может вести к нарушению работы этих ВМ.

Данные объекты существенно отличаются по своей природе, и для обеспечения наилучших результатов при борьбе с эффектом "старения" ПО были разработаны различные методы обеспечения процесса восстановления каждого из них. На рис. 1 приведена схема компонентов предлагаемой комплексной методики борьбы с эффектом "старения" ПО.

Разработанная методика использует три метода восстановления:

- Метод 1 — метод подмены виртуальной машины. Работа метода строится на основе подмены виртуальной машины с логическим сервером в процессе обслуживания пользователей [8]. Основными достоинствами данного метода являются сохранение доступности сервера в процессе восстановления и восстановление до уровня загрузки ОС.
- Метод 2 — метод перезапуска УВМ с перемещением ВМ и определением схемы их размещения. Для снижения негативного воздействия процесса восстановления на работу ВМ выполняется их перемещение на соседние хосты на основе технологии "горячей" миграции [9]. Такое решение обеспечивает восстановление УВМ до уровня загрузки ОС с сохранением активности логических серверов.
- Метод 3 — метод перезагрузки ОС. Хотя данный метод имеет существенный недостаток, связанный с временной остановкой приложения, он включен в данную методику, так как нетребователен к ресурсам, универсален и обеспечивает восстановления до уровня загрузки ОС.

Для восстановления объекта 1 используется метод 1, в случае недостатка ресурсов для его применения вместо него используется метод 3. Восстановление объекта 2 осуществляется на основе метода 2.

Для определения времени восстановления были разработаны два метода:

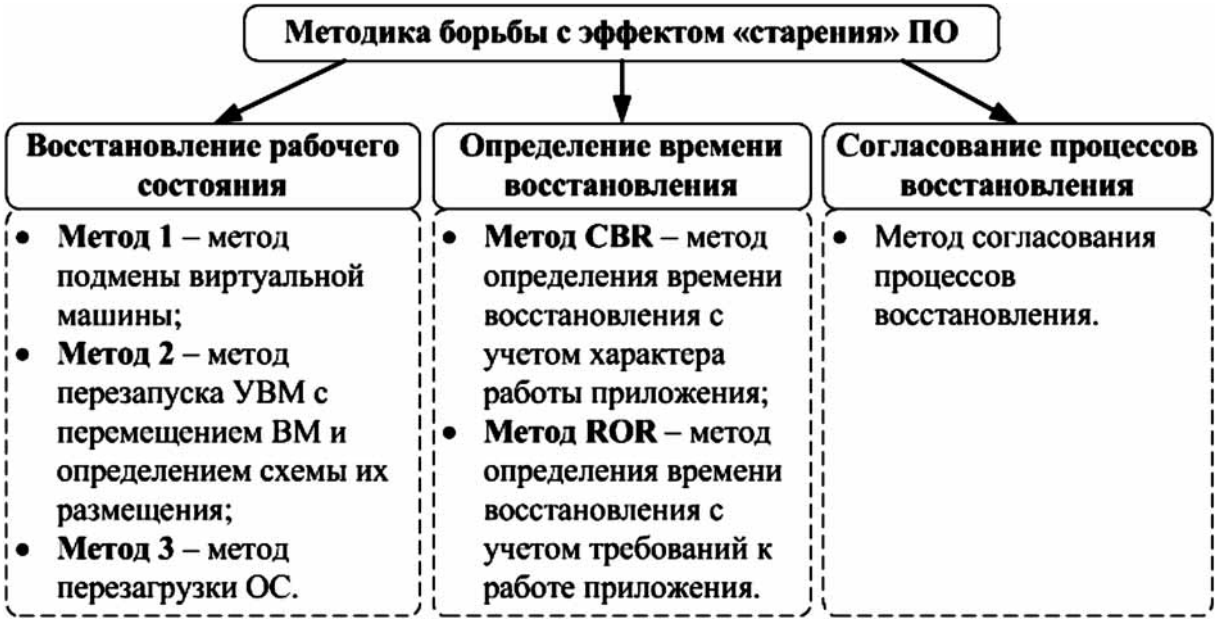


Рис. 1. Схема компонентов предлагаемой методики

• Метод определения времени восстановления по характеру работы приложения (*Condition-Based Rejuvenation*, CBR). Работа метода строится на основе мониторинга процесса "старения" приложения по одной из характеристик приложения с применением метода линейной регрессии. Такой подход обеспечивает учет изменения интенсивности "старения" приложения, что повышает его точность. Одной из ключевых особенностей метода CBR является возможность учета динамики изменения издержек процессов восстановления при работе приложения. Данное свойство метода реализуется на основе набора принципов, которые определяют характер изменения во времени негативного воздействия выбранного метода восстановления на приложение.

• Метод определения времени восстановления с учетом требований (*Requirement-Oriented Rejuvenation*, ROR). Особенностью данного метода является учет требований к работе сервера по нескольким показателям эффективности. При определении времени восстановления сервера учитывается негативное воздействие на работу приложения как эффекта "старения" ПО, так и процессов его восстановления по двум показателям эффективности работы сервера: времени обработки запросов и коэффициенту готовности. Работа метода направлена на выполнение заданных требований по этим показателям. В случае невозможности удовлетворения требований выполняется поиск решения для минимизации величин их нарушения.

Выбор метода определения времени зависит от типа объекта: для объекта 1 используется метод ROR, для объекта 2 — метод CBR.

Следующим компонентом предлагаемой методики является метод согласования процессов восстановления, цель которого заключается в определении параметров каждого процесса восстановления с учетом распределения ресурсов в ИС и взаимного влияния процессов. При согласовании процессов восстановления принимаются во внимание три показателя, характеризующие их воздействие на работу ИС: коэффициент готовности серверов, длительность и своевременность процессов восстановления. Работа метода строится на основе последовательного рассмотрения объектов восстановления в порядке возрастания расчетного времени начала восстановления и определения параметров процессов восстановления каждого из них.

Задача определения параметров восстановления объекта сведена к решению двух подзадач: первая — определение последовательности размещения виртуальных машин, вторая — планирование размещения виртуальной машины. В первой подзадаче выполняется формирование последовательности виртуальных машин в порядке возрастания их требований к ресурсам. Решение данной задачи строится на основе "жадного" алгоритма [10]. Объединение требований виртуальной машины по всему множеству ресурсов выполняется на основе метода линейной свертки векторного критерия в одну функцию — обобщенный (агрегированный) критерий. Во второй подзадаче выполняется планирование размещения каждой ВМ из подготовленной последовательности в целях минимизации длительности процесса восстановления и его своевременности. Решение данной подзадачи строится на основе метода динамического программирования.

При планировании размещения ВМ выполняется определение такого распределения ресурсов между хостами, которое обеспечивает размещение ВМ и является наилучшим с точки зрения длительности и своевременности выполнения процесса восстановления.

Одним из ключевых преимуществ данного метода согласования процессов восстановления является эффективное использование ресурсов ИС с точки зрения решаемой задачи.

Рассмотренные компоненты методики могут быть использованы независимо друг от друга. В рамках предлагаемой методики выполнено их объединение в виде законченного решения, именуемого далее система управления процессами восстановления. Порядок работы системы управления процессами восстановления включает два этапа:

- подготовительный этап;
- этап эксплуатации.

На подготовительном этапе выполняется определение множества объектов восстановления, настройка методов определения времени и методов восстановления. Для каждого объекта восстановления выполняется определение параметров методов определения времени восстановления на основе статистики работы объектов. Также на данном этапе могут быть заданы ограничения на размещение виртуальных машин, приоритеты восстановления и параметры процессов восстановления. Параметры, заданные на данном этапе, не являются строго фиксированными и при необходимости могут быть изменены на втором этапе в процессе работы системы управления.

На этапе эксплуатации в первую очередь выполняется формирование плана восстановления, включающее следующие шаги:

1. Определение времени восстановления объектов. На основе данных о текущем состоянии объектов восстановления выполняется определение времени восстановления каждого из них с помощью методов определения времени восстановления.

2. Согласование процессов восстановления. На основе рассчитанного времени восстановления, данных о состоянии ресурсов на хостах и требований виртуальных машин к ресурсам выполняется определение параметров процессов восстановления с помощью метода согласования процессов восстановления.

3. Корректировка параметров процессов восстановления. Особенностью используемых методов восстановления является возможность сохранения активности серверов на всем протяжении процесса восстановления при наличии необходимого объема ресурсов для их работы. В случае недостатка ресурсов вместо метода 1 для восстановления объекта 1 используется метод 3. Для объектов с методом восстановления 3 выполняется расчет времени восстановления на основе метода ROR с учетом временной остановки объекта 1, обусловленной особенностью используемого метода восстановления.

4. Сохранение плана восстановления. На основе набора параметров, полученного после согласования процессов восстановления и их корректировки, формируется план восстановления. План сохраняется в базе данных и используется для запуска процессов восстановления.

После того как план восстановления сформирован, система управления процессами восстановления функционирует в одном из двух следующих режимов.

Режим мониторинга. В данном режиме система отслеживает работу объектов восстановления, состояние хостов и виртуальных машин. В режиме мониторинга могут вноситься изменения в план восстановления в следующих случаях:

- если в план восстановления добавлен новый объект;
- если изменение одного из показателей состояния хостов и виртуальных машин превышает некоторое значение, определяющее допустимую погрешность его измерения;
- если изменение времени восстановления объекта превышает допустимую погрешность его измерения.

В каждом из этих случаев выполняется корректировка плана восстановления в соответствии с изменениями.

Режим восстановления. При достижении времени восстановления, указанного в плане, выполняется запуск процесса восстановления. На всем протяжении процесса восстановления выполняется блокировка мониторинга работы восстанавливаемого объекта и связанных с ним хостов и виртуальных машин. После завершения процесса восстановления система возвращается в режим мониторинга.

Управление процессами восстановления выполняется в циклическом режиме, после восстановления объекта выполняется его добавление в план восстановления.

Эксперименты

Для оценки предложенной методики была проведена серия экспериментов. Для этой цели был подготовлен тестовый стенд, который включал пять компьютеров с установленной платформой виртуализации, компьютер для хранения дисков виртуальных машин, четыре компьютера для запуска приложений, эмулирующих работу пользователей с сервером. Для проведения экспериментов была выбрана платформа виртуализации Citrix XenServer (URL: http://citrix.com), которая широко используется как в академических кругах для выполнения исследовательских работ, так и в промышленности, при построении виртуальной ИТ-инфраструктуры.

Для проведения тестов было задействовано восемь виртуальных машин со следующей конфигурацией: один процессор, 512 Мбайт оперативной памяти. Начальное размещение виртуальных машин на хостах варьировалось от одного до трех и выбиралось случайным обра-

зом в соответствии с равномерным распределением. Для экспериментов был выбран один из широко используемых серверов приложений — Apache Tomcat (URL: http:// tomcat.apache.org). В качестве источника "старения" ПО внутри УВМ была выбрана известная проблема утечки памяти в программах на языке Perl [11].

При тестировании был рассмотрен сложный случай, когда эффекту "старения" подвержены непосредственно сервер и УВМ. Для тестирования были сгенерированы два сценария работы ИС при воздействии эффекта "старения", отличающиеся интенсивностью "старения". Так как процесс "старения" может занимать достаточно продолжительное время, была использована техника его ускорения [12]. В обоих сценариях использовались характеристики web-нагрузки, полученные на основе реальных данных [13], но скорректированные в соответствии с техникой ускорения процесса "старения". Параметры первого сценария были следующие: время между сессиями рассчитывалось как сумма 280 мс и значения, выбранного на основе гамма-распределения с параметрами масштаба 2046,5634 и формы 0,6452; число запросов в сессии выбиралось в соответствии с распределением Вейбула с параметрами масштаба 10,6819 и формы 0,9124, но не более 30 запросов; время (единица измерения мс) между запросами выбиралось в соответствии с распределением Вейбула с параметрами масштаба 53,2432 и формы 0,8569; количество связных запросов выбиралось в соответствии с логнормальным распределением с математическим ожиданием 2,155 и среднеквадратическим отклонением 1,377, но не более 50 запросов, интервал времени вызова функции приложения, ответственной за "старение" УВМ, составлял 200 мс.

Второй сценарий отличался от первого временем между сессиями, которое было уменьшено на 10 % и рассчитывалось как сумма 252 мс и значения, выбранного на основе гамма-распределения с параметрами масштаба 1841,8764 и формы 0,6452, а также длительностью "старения" УВМ, которое также было уменьшено на 10 %.

Сравнение предложенной методики проводилось с широко используемыми решениями борьбы с эффектом "старения" ПО:

Решение 1. Комбинация метода перезагрузки ОС для объектов 1 и 2 и метода определения времени восстановления на основе мониторинга работы приложений [5].

Решение 2. Комбинация метода перезагрузки ОС для объектов 1 и 2 и метода определения времени восстановления без мониторинга работы приложения [6].

Решение 3. Комбинация метода перезапуска приложения для объекта 1 и метода перезагрузки ОС для объекта 2 с методом определения времени восстановления с мониторингом работы приложения [5].

Решение 4. Решение на основе быстрого перезапуска виртуальной машины для восстановления объектов 1 и 2 после обнаружения отказа [14].

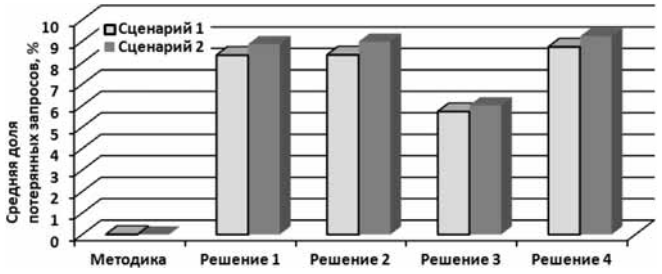


Рис. 2. Средняя доля потерянных запросов при различных сценариях для сравниваемых решений

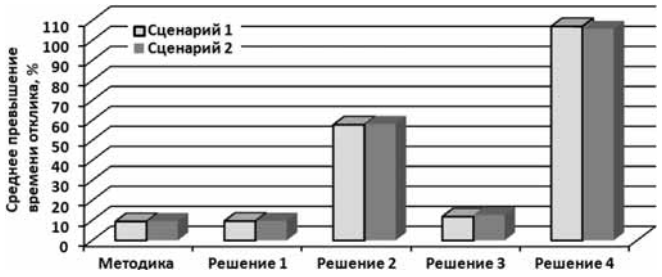


Рис. 3. Среднее превышение времени отклика при различных сценариях для сравниваемых решений

На рис. 2—3 приведены результаты экспериментов, демонстрирующие среднюю долю потерянных запросов и среднее превышение времени отклика серверов относительно среднего времени отклика серверов при отсутствии воздействия на них эффекта "старения" ПО и процессов восстановления.

Анализ результатов экспериментов показал, что разработанная методика обеспечивает наилучшие результаты среди сравниваемых решений. Методика позволила получить существенно лучшие результаты: исключить потерю запросов в процессе восстановления и обеспечить среднее время отклика на уровне лучших результатов среди сравниваемых решений. В процессе проведения экспериментов было обнаружено, что метод перезапуска приложения (Решение 3) не обеспечивает полного восстановления рабочего состояния сервера, а также длительность восстановления сервера данным методом увеличивается с ростом времени его работы под воздействием эффекта "старения" ПО. Кроме того, разработанная методика показывает более стабильные результаты при изменении интенсивности "старения" приложения среди сравниваемых решений. По результатам экспериментов можно сделать вывод, что предложенная методика лучше справляется с задачей снижения негативного воздействия эффекта "старения" ПО на эффективность работы ИС, построенной на основе технологии виртуальных машин, чем решения, с которыми проводилось сравнение.

Заключение

В данной работе была предложена комплексная методика борьбы с эффектом "старения" ПО для ИС, построенной на основе технологии виртуальных машин. Особенностью предложенной методики является комплексное рассмотрение задачи борьбы с эффектом "старения" ПО, которое включает в себя решение подзадач определения времени восстановления, восстановление приложения и согласование процессов восстановления.

Проведенные эксперименты показывают, что предложенная методика превосходит по выбранным показателям эффективности распространенные решения и лучше справляется с задачей снижения негативного воздействия эффекта "старения" ПО на эффективность работы ИС. Полученные в ходе проведения экспериментов результаты демонстрируют, что предложенная методика позволяет сохранить запросы пользователей при борьбе с эффектом "старения" ПО и обеспечить время отклика на уровне лучших результатов среди сравниваемых решений.

Стоит отметить, что для обеспечения наилучших результатов применения предложенной методики необходимо наличие достаточного объема свободных ресурсов. Это позволяет избежать нарушения доступности серверов для пользователей в процессе восстановления. Несмотря на то, что метод согласования процессов восстановления стремится максимизировать средний коэффициент готовности серверов, наличие достаточного объема свободных ресурсов на момент восстановления объекта является той ценой, которую приходится платить за высокие результаты.

Список литературы

1. **Huang Y., Kintala C., Kolettis N., Fulton N.** Software rejuvenation: Analysis, module and applications // The Proc.s of Fault-Tolerant Computing Symposium. 1995. Vol. 25. P. 381—390.

2. **Candea G.** [и др.]. Microreboot — A Technique for Cheap Recovery // Proc. 6th Symp. on Operating Systems Design and Implementation. 2004. Vol. 6. P. 31—34.

3. **Kourai K., Chiba S.** A fast rejuvenation technique for server consolidation with virtual machines // Proc. of International Conference on Dependable Systems and Networks. 2007. Vol. 37. P. 245—255.

4. **Орлов С.** Виртуализация "от и до" // Журнал сетевых решений / LAN. 2010. № 2.

5. **Li L., Vaidyanathan K., Trivedi K.** An Approach for Estimation of Software Aging in a Web Server // International Symposium on Empirical Software Engineering. 2002. Vol. 7. P. 91—100.

6. **Dohi T., Goseva-Popstojanova K., Trivedi K.** Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule // Proc. of 2000 Pacific Rim International Symposium on Dependable Computing. 2000. P. 77—84.

7. **Castelli V.** etc. Proactive Management of Software Aging // IBM Journal of Research & Development. 2001. Vol. 45. N 2. P. 311—332.

8. **Удовиченко А. О.** Разработка метода восстановления рабочего состояния серверного ПО на основе технологии виртуальных машин // 7-я международная научно-практическая конференция "Trans-Mech-Art-Chem". 2010. С. 368—370.

9. **Clark C.** etc. Live Migration of Virtual Machines // 2nd Symposium on Networked Systems Design and Implementation. 2005. Vol. 2. P. 273—286.

10. **Коган Д. И.** Задачи и методы конечномерной оптимизации. Часть 3. Динамическое программирование и дискретная многокритериальная оптимизация. Нижний Новгород: Изд-во НТУ, 2004. 157 с.

11. **Мичурин А.** Утечки памяти в программах на Perl // Системный администратор. 2004. № 5 (18). URL: http://samag.ru/archive/article/286

12. **Matias R., Barbetta P., Trivedi K., Filho P.** Accelerated Degradation Tests Applied to Software Aging Experiments // IEEE Transactions on Reliability. 2010. Vol. 59. N 1. P. 102—114.

13. **Walters L. O.** A web browsing workload modeling for simulation: master of science thesis. Cape Town: UCT, 2004. 177 p.

14. **High** availability and disaster recovery with Microsoft, Citrix and HP. URL: http://citrixandmicrosoft.com/Docs/WhitePapers/CTX_HP_MS_HA_DR_WhitePaper_final.pdf .

ИНФОРМАЦИЯ

УДК 004.42

Е. А. Кольчугина, канд. техн. наук, доц.,
Пензенский государственный университет,
e-mail: kea-sci@list.ru

Применение методов теории нумераций для представления эволюции кода программных агентов в темпоральных базах данных

Рассматриваются вопросы представления кода эволюционирующих программных агентов во временных базах данных. Рассматриваются способы выполнения сверток на основе нумерации Кантора. Для уменьшения роста значения функции свертки предложено использовать схему сдваивания.

***Ключевые слова**: теория нумераций, программные агенты, темпоральные базы данных, свертка и восстановление последовательностей*

Введение

Современные программы обладают высокой структурной и функциональной сложностью, которую можно понимать как большое число элементов, связей между ними, а также большое количество внешних событий, на которые должна реагировать программа, находясь в вычислительном окружении и взаимодействуя с аппаратным обеспечением и другими программами.

Чем сложнее программы, тем сложнее их проектировать и обеспечивать корректную работу в процессе эксплуатации. Поэтому желательно, чтобы программы обладали способностью самостоятельно адаптироваться к среде выполнения, т. е. программа или программная система должны уметь перестраивать и перепроектировать самих себя в течение всего жизненного цикла. Подобного рода методы адаптации, основанные на применении генетических алгоритмов, используются в теории искусственной жизни.

В настоящей работе рассматриваются вопросы, связанные с созданием эволюционирующего программного обеспечения в виде коллектива программных агентов. Каждый программный агент описывается цифровой ДНК в виде числовой последовательности, для кодирования и оперирования которой можно применить методы теории чисел и теории нумераций.

Кодирование программного агента с помощью числовой последовательности

Основная идея теории нумераций состоит в установлении соответствий между объектами и целочисленными номерами с последующей заменой операций над самими объектами операциями над нумерующими их целыми числами.

В работе [1] с использованием результатов теории нумераций [2—4] была предложена концепция построения самоорганизующейся программной системы в рамках парадигмы неравновесного программирования. Основой построения таких самоорганизующихся систем является коллектив эволюционирующих программных агентов. При этом в процессе функционирования самоорганизующейся программной системы могут изменяться как значения переменных состояния агентов, так и поведение агентов. Это возможно благодаря тому, что агент кодируется с помощью генотипа — линейной последовательности целых чисел. Числа представляют собой номера возможных значений переменных состояния и программных реализаций отдельных поведенческих блоков базового алгоритма агента. Таким образом, отдельные множества как значений переменных, так и программных реализаций можно рассматривать как домены, на которых заданы нумерации [3]. Сам агент представляет собой результат сборки, выполненной на основе доменов значений переменных и программных реализаций

в соответствии с кодирующей числовой последовательностью.

Непрерывная эволюция программной системы ставит задачу исследования хода эволюции в целях управления ею. Поэтому важное значение приобретает представление смены состояний и алгоритмов поведения агентов в базах данных. Наиболее подходящей концепцией баз данных для решения подобных задач является концепция темпоральных баз данных (в отечественной литературе иногда используются синонимы — временны́е, динамические базы данных).

Темпоральные базы данных

Концепция темпоральных или временных баз данных — это концепция, в которой центральное место занимает задача отражения категории времени и ее свойств, а как следствие, динамики предметной области. Отличительным свойством темпоральных баз данных являются:

- привязка истинности или ложности утверждения к моменту времени, относительно которого делается утверждение (поддержка временной логики);
- обязательное представление текущего момента времени (как правило, в виде системной переменной);
- отсутствие необходимости удаления данных — при потере актуальности данных в таблицы просто добавляются новые записи с актуальными данными;
- неограниченный рост объема базы данных, являющийся следствием отсутствия удалений данных.

Наибольшую известность получили исследования в области темпоральных баз данных, проводимые группой TimeCenter во главе с Р.Т. Снодграссом [5]. Работы этой группы посвящены главным образом представлению семантики категории времени в языке SQL, в результате исследований был предложен темпоральный диалект SQL — TSQL. В этих работах моменты времени (временные метки, *timestamps*) представляются традиционными средствами SQL, а интервалы задаются как наборы временных меток.

В работах отечественных авторов [6, 7] развивался другой подход, абстрагированный от SQL или каких-либо языковых средств. Основное внимание уделялось представлению в базах данных, традиционно работающих с дискретной информацией, непрерывных функций. При этом моменты времени (или эпохи [7]) задавались числами, и их можно было рассматривать как номера событий. Интервалы времени задавались парой чисел, обозначающих координаты начала и конца интервала, которые также нумеруют события.

В работе [6] предложено представлять непрерывные функции, например, описывающие реальные физические процессы, в виде множества аппроксимирующих их полиномов, соответствующих различным интервалам времени. Значение функции в конкретный момент времени получается в результате подстановки координаты момента времени в соответствующий полином (в терминологии работы [6] — генерирующую функцию).

В работе [7] была предложена модель данных, являющаяся проблемно-ориентированным вариантом реляционной модели, в которой поддерживалась работа как с полиномами (генерирующими функциями), так и с отдельными результатами измерений, и допускались их взаимные преобразования, такие как аппроксимация и выборка. Однако основное внимание было уделено представлению категории времени путем поддержки представления в базах данных временных шкал — гомоморфных отображений множества событий на множество моментов времени. Время, при условии нетождественности и неодновременности событий, можно рассматривать как функцию нумерации, упорядочивающую события. Применение чисел для нумерации событий удобно тем, что это позволяет использовать различные единицы измерения времени и оперировать временными рядами как с A — так и с B — рядами [8], т. е. вести отсчет как от зафиксированного момента времени, так и от постоянно изменяющегося текущего момента.

Используя подобный подход, генотип числового организма может быть представлен в темпоральной базе данных двумя способами. В случае дискретного способа представления времени высказывание $\langle t, \langle n, x_1, ..., x_n\rangle\rangle$ означает, что в момент t программный агент кодировался последовательностью неотрицательных целых чисел $x_1, ..., x_n$ длины n. В случае интервального представления высказывание $\langle\langle t_b, t_e\rangle, \langle n, x_1, ..., x_n\rangle\rangle$ означает утверждение, что на протяжении временного интервала $\langle t_b, t_e\rangle$ программный агент кодировался последовательностью неотрицательных целых чисел $x_1, ..., x_n$ длины n и эта последовательность оставалась неизменной.

С точки зрения хранения числовых последовательностей в таблицах, неудобство состоит в том, что с течением времени могут изменяться не только элементы кодирующих последовательностей, но и длины последовательностей. Поэтому было бы предпочтительнее заменять каждую последовательность одним числом — результатом вычисления функции свертки, а при необходимости выполнять восстановление последовательности.

Алгоритмы свертки и восстановления числовых последовательностей

Нумерацией [2—4] будем называть отображение множества чисел, задающих номера, на множество нумеруемых объектов:

$$\nu_{Obj}: N_1 \to Obj.$$

В дальнейшем, если не будет оговорено другое, будем считать, что $N_1 \subseteq \mathbf{N}_{0+}$, где $\mathbf{N}_{0+}$ — множество целых неотрицательных чисел, а отображение ν_{Obj} является взаимнооднозначным.

В работе [2] вводится понятие функции свертки, или n-местной функции, осуществляющей отображение $\mathbf{N}_{0+}^n$ в $\mathbf{N}_{0+}$, которая может строиться на основе

функции свертки пары целых неотрицательных чисел $s(x, y)$ [2]:

$$s^1(x) = x,$$
$$s^2(x, y) = s(x, y),$$
$$\ldots$$
$$s^n(x_0, \ldots, x_n) = s(s^{n-1}(x_0, \ldots, x_{n-1}), x_n).$$

Обратная операция, или восстановление последовательности по ее номеру, выполняется с помощью n-развертки, или набора r_n одноместных функций [2]:

$$s^n(r_{n,1}(x), \ldots, r_{n,n}(x)) = x,\ r_{n,1}(s^n(x_0, \ldots, x_n)) = x_i.$$

В дальнейшем нумерация последовательностей будет конструироваться на основе нумерации пар. К таким нумерациям относится нумерация Кантора [2—4]:

$$f(x, y) = s = \frac{(x+y)(x+y+1)}{2} + x,$$

$$x = L(s) = s \dot{-} \frac{1}{2}\left[\frac{[\sqrt{8s+1}]+1}{2}\right]\left[\frac{[\sqrt{8s+1}] \dot{-} 1}{2}\right],$$

$$y = R(s) = \left[\frac{[\sqrt{8s+1}]+1}{2}\right] -$$
$$- s \dot{-} \frac{1}{2}\left[\frac{[\sqrt{8s+1}]+1}{2}\right]\left[\frac{[\sqrt{8s+1}] \dot{-} 1}{2}\right] - 1.$$

Здесь $[a]$ означает округление до ближайшего меньшего целого, символ $\dot{-}$ означает усеченную разность [4]:

$$a \dot{-} b = \begin{cases} a - b, & a \geqslant b, \\ 0, & a < b. \end{cases}$$

Используя введенные понятия свертки и функции нумерации пар, можно заменить кортежи темпоральных отношений с дискретным представлением времени $\langle t, \langle n, x_1, \ldots, x_n\rangle\rangle$ парами чисел $\langle y_1, y_2\rangle = \langle t, s^{n+1}(x_1, \ldots, x_n, n)\rangle$, а кортежи отношений с интервальным представлением времени вида $\langle\langle t_b, t_e\rangle, \langle n, x_1, \ldots, x_n\rangle\rangle$ — парами чисел $\langle y_1, y_2\rangle = \langle s^2(t_b, t_e), s^{n+1}(x_1, \ldots, x_n, n)\rangle$. Преимущество такого представления состоит в том, что с течением времени длина цифровой ДНК агента может изменяться (удлиняться или укорачиваться), но на способе представления цифровой ДНК в базе данных это никак не скажется.

Эффективность применения свертки определяется свойствами функции нумерации пар $f(x, y)$. Функция $f(x, y) = \frac{(x+y)(x+y+1)}{2} + x$ растет пропорционально среднему x и y, но при этом рост x оказывает более существенное влияние на рост значений функции, чем рост y. Для простоты будем считать, что $x = y$. Тогда $s^2(x, x) = 2x^2 + 2x$. Для последовательности длины n полученный результат будет использован еще $n - 2$ раза при вычислениях последующих сверток. Таким образом, грубая заниженная оценка роста значения функции свертки по методу Кантора может быть получена по первому члену итогового полинома как $2^{n-1}x^{2^{n-1}}$.

На практике это приводит к тому, что для последовательности $w = \langle 4, 4, 2, 1, 0, 1, 3\rangle$ значение свертки при использовании метода Кантора равно 12448864443827436089573933594197096117912157 (1,24489E + 43).

Однако рост значений функции свертки можно снизить. Для этого предлагается изменить порядок вычисления сверток с линейного на известную в параллельном программировании схему сдваивания. В случае схемы сдваивания уменьшается и уравнивается число сумм, в которых участвует каждый из членов исходной последовательности, благодаря этому функция свертки растет не так быстро, как в классическом линейном случае. На заключительной фазе может быть выполнен еще один дополнительный шаг, позволяющий добавить к свертке исходной последовательности число составлявших ее элементов, что важно для выполнения обратной операции восстановления последовательности.

$$sv^m(x_1, \ldots, x_m) = f(l_m, r_m),\ 2 < m \leqslant n,\ k = 2^{[\log_2 m]},$$

$$l_m = \begin{cases} sv^k(x_1, \ldots, x_k), & m > k, \\ sv^{[m/2]}(x_1, \ldots, x_{[m/2]}), & m = k, \end{cases}$$

$$r_m = \begin{cases} sv^{m-k}(x_{k+1}, \ldots, x_m), & m > k, \\ sv^{[m/2]}(x_{[m/2]+1}, \ldots, x_m), & m = k, \end{cases}$$

$$sv^2(x_i, x_{i+1}) = f(x_i, x_{i+1}),\ sv^1(x_i) = x_i,$$

$$sv^{n+1}(x_1, \ldots, x_n, n) = f(sv^n(x_1, \ldots, x_n), n).$$

Модифицированная операция свертки позволяет определить операцию восстановления исходной последовательности:

$$rzv^{n+1}(sv^{n+1}(x_1, \ldots, x_n), n)) = \langle rzv^n(sv^n(x_1, \ldots, x_n)), n\rangle,$$

$$rzv^n(sv^n(x_1, \ldots, x_n)) = \langle x_1, \ldots, x_n\rangle,$$

$$rzv^m(sv^m(x_1, \ldots, x_m)) = \langle rzv^{k_1}(l_m), rzv^{k_2}(r_m)\rangle,$$
$$2 < m \leqslant n,\ k = 2^{[\log_2 m]},$$

$$k_1 = \begin{cases} k, & m > k, \\ [m/2], & k = m, \end{cases}$$

$$k_2 = \begin{cases} m\text{-}k, & m > k, \\ [m/2], & k = m, \end{cases}$$

$$l_m = L(sv^m(x_1, \ldots, x_m)),$$

$$r_m = R(sv^m(x_1, \ldots, x_m)),$$

$$l_m = \begin{cases} sv^k(x_1, \ldots, x_k), & m > k, \\ sv^{[m/2]}(x_1, \ldots, x_{[m/2]}), & m = k, \end{cases}$$

$$r_m = \begin{cases} sv^{m-k}(x_{k+1}, \ldots, x_m), & m > k, \\ sv^{[m/2]}(x_{[m/2]+1}, \ldots, x_m), & m = k, \end{cases}$$

$$rzv^2(s^2(x_i, x_{i+1})) = \langle L(s^2(x_i, x_{i+1})),$$
$$R(s^2(x_i, x_{i+1}))\rangle = \langle x_i, x_{i+1}\rangle,\ rzv^1(s^1(x_i)) = x_i.$$

Грубое оценивание роста значений функции свертки при использовании схемы сдваивания и при условии, что все элементы исходной последовательности равны x, дает $2^{[\log_2 n]-1}\, x^{2^{[\log_2 n]-1}}$.

Для приведенной выше последовательности $w = \langle 4, 4, 2, 1, 0, 1, 3\rangle$ результат свертки по схеме сдваивания составил 754594 без участия в формировании свертки числа элементов последовательности, и 284712466495 в случае учета при вычислении свертки числа элементов исходной последовательности.

Необходимо отметить, однако, что для вычисления сверток, разверток и восстановления последовательностей как по традиционной схеме, так и по схеме сдваивания, следует применять методы "длинной" арифметики, а в целях повышения производительности необходимо использовать параллельные вычисления, например с использованием пакетов MPI или CUDA NVIDIA.

Заключение

Применение методов теории нумераций позволяет не только универсальным образом закодировать реализацию программного агента, но и обеспечить ее хранение в темпоральной базе данных реляционного типа. При этом проявляется свойство взаимной двойственности программ и данных: с одной стороны, программы кодируются последовательностями данных, которые могут храниться в таблицах; с другой стороны, последовательности чисел из таблиц могут кодировать программы, которые выполняют операции над таблицами.

Недостатком применения операций свертки над последовательностями номеров является быстрый, превышающий экспоненциальный рост значений результатов свертки. Однако применение схемы сдваивания способно замедлить этот рост. Следует ожидать, что с совершенствованием методов и алгоритмов вычисления сверток, развитием технологий параллельного программирования для многоядерных архитектур, методы теории нумераций и операции свертки будут применяться все чаще в различных областях.

Список литературы

1. **Кольчугина Е. А.** Неравновесное программирование// Известия высших учебных заведений. Поволжский регион. Технические науки. 2009. № 3 (11). С. 25—31.

2. **Ершов Ю. Л.** Теория нумераций. М.: Наука, 1977. 416 с.

3. **Линьков В. М.** Нумерационные методы в проектировании систем управления данными: Монография. Пенза: Изд-во Пенз. гос. техн. ун-та, 1994. 156 с.

4. **Мальцев А. И.** Алгоритмы и рекурсивные функции. 2-е изд. М.: Наука, 1986. 368 с.

5. **TimeCenter Publications**. [Электронный документ]. URL: http://timecenter.cs.aau.dk/pub.htm.

6. **Гуляев А. И.** Временные ряды в динамических базах данных. М.: Радио и связь, 1989. 128 с.

7. **Кольчугина Е. А.** Системы управления временными базами данных для решения задач обработки информации в автоматизированных системах управления распределенными объектами (Спец. 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей): Дисс. ... канд. техн. наук. Пенза, 1998. 156 с.

8. **Караваев Э. Ф.** Основание временной логики. Ленинград: Изд-во Ленинградского университета, 1983. 176 с.

ИНФОРМАЦИЯ

Международный конгресс по интеллектуальным системам и информационным технологиям

IS&IT'12

2—9 сентября 2012 г.,
Россия, Черноморское побережье, Геленджик-Дивноморское.

Официальный сайт конгресса *http://icai.tsure.ru*

УДК 004.436.2

А. В. Жарковский, канд. техн. наук, зам. проректора по науч. работе,
А. А. Лямкин, канд. техн. наук, доц.,
Н. П. Микуленко, канд. техн. наук, стар. науч. сотр.,
Санкт-Петербургский государственный электротехнический университет "ЛЭТИ",
e-mail: av.jarkov@yandex.ru

Структурограммы на основе объектно-признакового языка

Предлагается улучшенная форма структурограмм для представления (описания) процессов функционирования систем на основе объектно-признакового языка.

Ключевые слова: *структурограмма, модель, объектно-признаковый язык, алгоритм*

Цели функционирования любой сложной технической системы достигаются только благодаря управлению. Правила управления поведением системы реализуются в виде функционального (боевого, бортового) программного обеспечения (ФПО) комплекса управления. От качества ФПО зависит эффективность системы, а также сроки и стоимость ее создания. На ранней стадии создания ФПО (стадии концептуального проектирования), когда разрабатываются модели поведения системы и алгоритмы управления ею, важно использовать формализованные способы описания модели будущего ФПО, которые были бы одинаково понятны специалистам заказчика и исполнителя. Это помогло бы и на последней стадии создания ФПО при его отладке и испытаниях.

Модель (точнее, алгоритмическая модель) процесса функционирования системы или ее компонента, в отличие от вычислительного алгоритма, предполагает многократное воспроизведение во времени одной и той же последовательности действий (по-существу, одного и того же алгоритма) при меняющихся исходных данных, которые вводятся автоматически как результат работы внешних устройств или воспроизведения других алгоритмов.

Широко распространенное представление алгоритмов в виде блок-схем получается хоть и наглядным, но подчас труднообозримым, а допустимость вербальных формулировок и отсутствие способа управления уровнем детализации блок-схем делает переход от алгоритма к программе затруднительным и неоднозначным. Более удобной формой представления алгоритмов являются структурограммы (схемы Нэсси—Шнайдермана) [1]. По компактности (обозримости) и формализованности записи они в целом превосходят блок-схемное представление алгоритмов, не уступая ему в наглядности, и могут использоваться для описания моделей функционирования.

Однако компактному (на одном листе) и наглядному представлению моделей в виде структурограмм мешают отсутствие в них символов обращения к другим алгоритмам и передачи управления при окончании выполнения данного алгоритма, длинные косые линии и необходимость указания "Да" и "Нет" в блоках "Решение", запись в каждом блоке только одного действия или одного условия, использование служебных слов из подмножества английского языка и т. п. Это препятствует широкому использованию структурограмм для представления моделей функционирования систем и их компонентов.

В общем случае алгоритм функционирования или алгоритмическую модель ($A.X$) любого объекта X можно рассматривать как "черный ящик", совершающий преобразование входной информации в выходную на основе типа объекта:

$$A.X = F.XU \xrightarrow{T.X} F.XI,$$

где $F.XU$ и $F.XI$ — формуляры обмена, содержащие соответственно входную и выходную информацию, а $T.X$ — тип объекта (система, ее компонент или алгоритм). Для описания различных объектов используется объектно-признаковый язык [2], являющийся промежуточным между вербальным языком и языками

программирования высокого уровня. Этот язык, в отличие от псевдокода [1], не содержит служебных слов.

Коротко о языке. Можно считать, что каждый конкретный объект *X* принадлежит к определенному множеству (типу) объектов (*T.X*). Любой объект этого множества характеризуется номером (именем конкретного экземпляра) и целым набором признаков, которые перечисляются через запятую в квадратных скобках после имени объекта (*T.X* [...]). Различают объекты двух видов: объекты *T.X* [...] (система, блок), признаки которых являются техническими характеристиками, неизменными в процессе функционирования, и объекты *F.X* [...] (информационные связи), признаки которых являются переменными во времени величинами. Объекты и их признаки обозначаются на латинице начальными буквами терминов языка предметной области и в виде общепринятых обозначений физических величин.

Существуют признаки числовые (например, *D* — дистанция, *V* — скорость, *U* — угол) и нечисловые (например, *TC* — тип цели, *PH* — признак высоты и др.). Если признак объекта характеризуется только одним членом некоторого множества, то все члены множества перечисляется в фигурных скобках (например, *TC*:{*C, W, R*}, где *C* — самолет, *W* — вертолет, *R* — ракета). Если признак объекта характеризуют все члены множества, то они задаются кортежем в ломаных скобках (например, *GR*: <*LX, LY, LZ*>, где *GR* — геометрические размеры объекта вдоль осей его симметрии).

Структурограмма *A.X* (рис. 1) как средство описания модели функционирования объекта *X* состоит из трех соединенных между собой блоков: входные данные, выходные данные и тело алгоритма (модели). Далее, не обсуждая известные общие правила построения структурограмм, отметим лишь предлагаемые в них изменения, которые либо связаны с представлением моделей на основе объектно-признакового языка, либо ведут к более компактному представлению самих структурограмм.

В блоке исходных данных указывается входной формуляр *F.XU* [...] (не обязательно один) и тип объекта *T.X* [...] и/или только некоторые его константы (*C* [...]), а в блоке выходных данных — выходной формуляр *F.XI* [...]. Это не ввод и не вывод данных, не действие, а некая совокупность данных, которая соответственно используется в модели или получается в результате выполнения алгоритма. Признаки объекта *T.X* [...] неизменны и задаются однократно вручную, а информация, содержащаяся во входном формуляре *F.XU* [...], является переменной и поступает от некоторого внешнего по отношению к данному объекта (алгоритма или устройства).

Тело алгоритма содержит графические символы (блоки) трех типов [1]: Обработка, Решение и Цикл. Блок обработки включает в себя действия, выполняемые операторами присваивания значений, выполнения общематематических вычислений и передачи управления. В одном блоке обработки могут записываться несколько операторов, разделяемых через точку с запятой и составляющих смысловую группу. Все операторы читаются и выполняются по порядку слева направо и сверху вниз.

Первым в теле алгоритма указывается блок инициализации, состоящий из операторов присваивания (:=) начальных значений некоторым переменным, используемым в алгоритме. В качестве примера приведены (см. рис.1) присваивание нулевого значения счетчику циклов *N* (*N*:= 0) и присваивание максимального значения *L* ограничителю числа циклов *K* (*K*:= *L*). Передача управления внутри алгоритма изображается с помощью вложенных структур, а для обозначения передачи управления к другому алгоритму можно использовать графический символ в виде стрелки (→), за которой следует имя алгоритма (например, алгоритм *B*), а затем в круглых скобках через запятую его входной *F.BU* [...] и выходной *F.BI* [...] формуляры. Естественно, что перед этим необходимо записать в *F.BU* [...] информацию, результат действия над которой будет указан в *F.BI* [...]. Отдельный символ вывода информации отсутствует. Передача (запись) информации проводится в любом месте тела алгоритма по мере готовности данных и обозначается как оператор присваивания значения какой-либо пере-

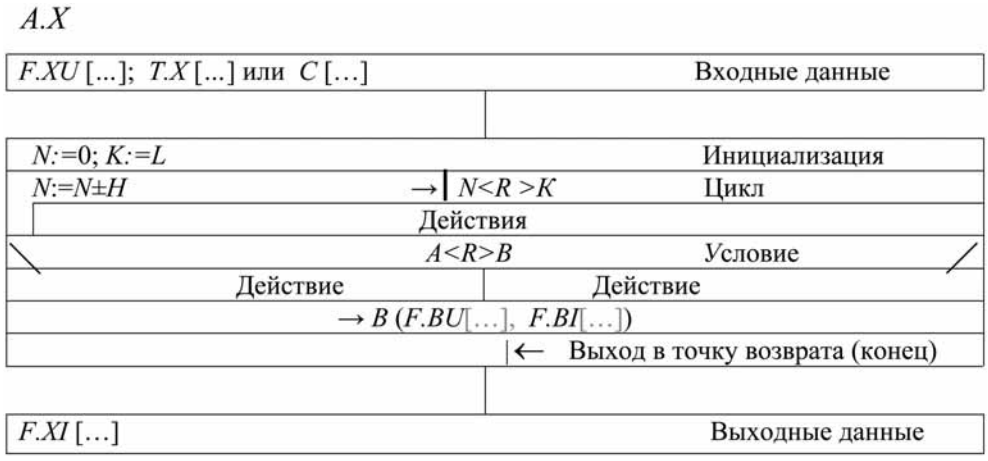
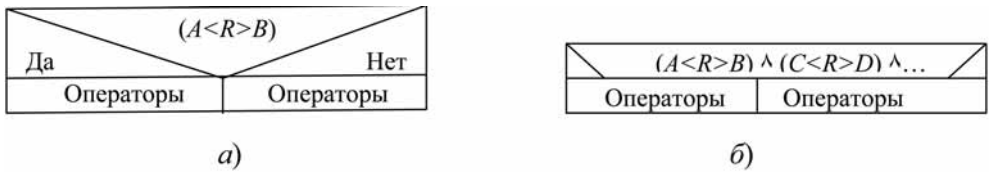


Рис. 1.



а) *б*)

Рис. 2.

менной выходного формуляра данного алгоритма. Окончание алгоритма обозначается обратной стрелкой с ограничением (|←). В случае модели это означает "возврат в точку выхода", т. е. возврат в ту точку более сложной модели, из которой было обращение к данной.

Символ Решение (рис. 2, *а*) применяется для обозначения конструкций "Если..., то...". Условие (вопрос) располагается в верхнем треугольнике, варианты ответов Да и Нет — по его сторонам, а операторы процесса обработки — в блоках, лежащих под вариантами ответов.

Если несколько изменить верхний блок (рис. 2, *б*), ограничив косые линии, то вместо одного условия можно записать несколько, соединив их логическими функциями "И" (∧) и "ИЛИ" (∨). Например,

$$(A < R > B) \wedge (C < R > D) \wedge \ldots,$$

где <*R*> — отношение между двумя какими-либо величинами (*A* и *B* или *C* и *D*), принимающее значения ⩽, = , > и др. А если принять, что слева всегда будут располагаться операторы, соответствующие выполнению условия, а справа — его невыполнению (это легко выполнить, изменив отношение), то можно отказаться и от прямого указания ответов Да и Нет.

Блок Цикл (рис. 3) содержит счетчик числа циклов *N* с добавлением и вычитанием шага *H*, который может принимать любое целое значение (в частности, $H = 1$) и используется для обозначения многократного выполнения ряда операторов, изображаемых во внутреннем прямоугольнике. Номер счетчика — это номер устройства, номер записи в массиве данных и т. п.

Процесс выполнения операторов повторяется некоторое число раз либо до тех пор (→|), пока выполняется некоторое условие ($N < R > K$), либо до тех пор пока оно не выполнится. Этому соответствует место проверки условия завершения цикла — в его начале (для цикла с предусловием) или в его конце (для цикла с постусловием). При достижении условия цикл заканчивается и осуществляется переход к следующему оператору.

Рассмотрим в качестве примера упрощенную модель функционирования стационарной наземной радиолокационной станции (РЛС) кругового обзора, служащей для обнаружения воздушных целей. Такие РЛС являются примером наиболее распространенных обнаружителей сложных технических систем.

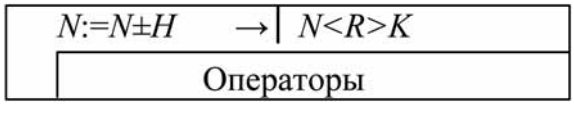


Рис. 3.

В общем виде модель функционирования РЛС (*M.Z*) можно представить как преобразование входного формуляра *F.ZU* в выходной формуляр *F.ZI* на основе типа РЛС (*T.Z*) и ее пространственного размещения (*T.RZ*)

$$M.Z = F.ZU \xrightarrow{T.Z\&T.RZ} F.ZI.$$

Входной формуляр, поступающий из модели окружающей среды, содержит информацию о положении воздушных объектов (*S*) и его можно записать как

$$F.ZU\ [LS,\ N\_S,\ PR:\{BR,\ MR\},\ X.S.,\ Y.S,\ Z.S],$$

где *LS* — число объектов в окружающей среде; *N_S* — порядковый номер объекта; *PR*:{*BR*, *MR*} — признак размера объекта со значениями *BR* — объект большеразмерный, *MR* — объект малоразмерный; *X.S, Y.S* и *Z.S* — координаты пространственного положения объекта в декартовой системе координат, принятой для описания общей модели функционирования системы.

Обнаруженные объекты принято называть целями. Выходной формуляр РЛС содержит координаты целей в сферической системе координат, связанной с обнаружителем. Его можно записать в виде

$$F.ZI\ [LC,\ N\_C,\ K,\ M,\ D,\ TD,\ TE,\ PR:\{BR,\ MR\},\ PH:\{H,\ N\}],$$

где *LC* — число обнаруженных целей; *N_C* — номер цели; *K* — курсовой угол; *M* — угол места цели; *D* — дальность до цели; *TD* и *TE* — соответственно точность измерения дистанций и углов; *PR*:{*BR*, *MR*} — признак размера цели, совпадающий с признаком размера объекта среды; *PH* :{*H*, *N*} — признак высоты цели со значениями *H* — цель высоколетящая и *N* — цель низколетящая.

В упрощенном виде тип обнаружителя может быть описан как

$$T.Z\ [N\_T.Z,\ DM,\ DH,\ PC: < PR,\ PH >,\ TD,\ TE],$$

где *N_T.Z* — имя (номер типа) обнаружителя; *DM* и *DH* — соответственно минимальная и максимальная дальность обнаружения; *PC* — вырабатываемые признаки цели (признак размера и признак высоты); *TD* и *TE* — соответственно точность измерения дистанций и углов.

Пространственное размещение РЛС

$$T.RZ\ [N\_T.RZ,\ X.Z,\ Z.Z]$$

характеризуется двумя декартовыми координатами *X.Z* и *Z.Z*.

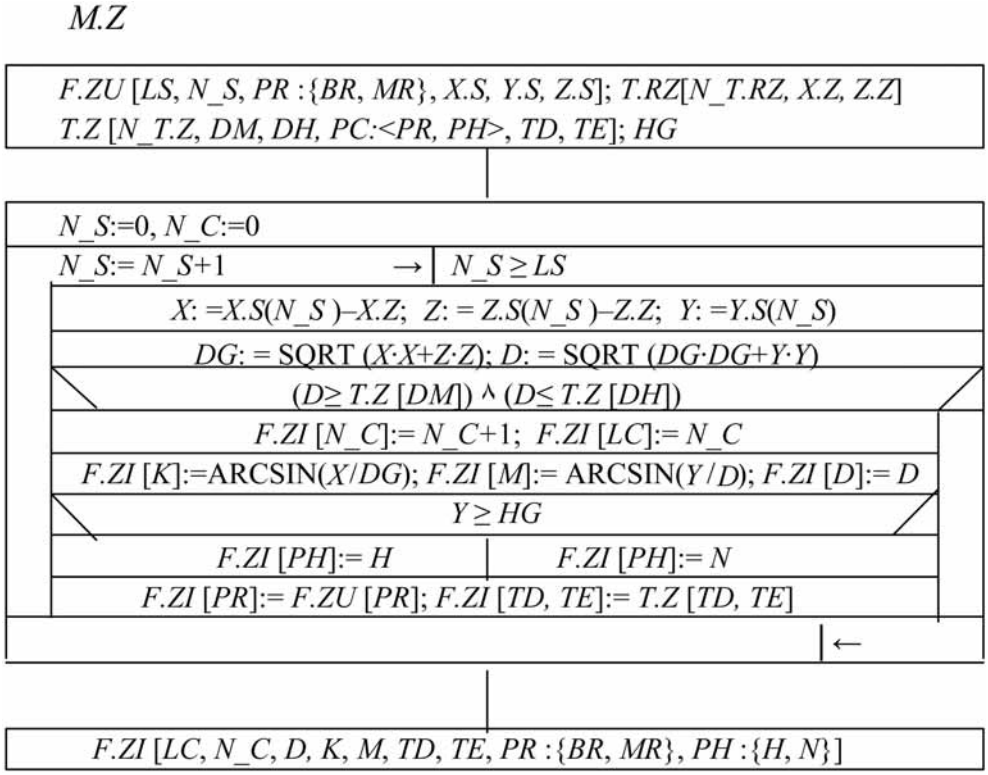


Рис. 4.

Упрощенная модель РЛС *M.Z*, как часть более общей модели, например модели системы противовоздушной обороны, может быть представлена в виде, приведенном на рис. 4.

Сначала в этой модели устанавливаются начальные нулевые значения счетчиков объектов ($N\_S: = 0$) и целей ($N\_C: = 0$), а затем организуется цикл по перебору всех объектов среды, начиная с первого. В цикле сначала определяются координаты каждого объекта в связанной с РЛС декартовой системе координат, а затем вычисляются горизонтальная (*DG*) и наклонная (*D*) дальности до объекта. После этого определяется, может ли быть обнаружен объект, исходя из присущих данной РЛС характеристик по дальности обнаружения целей. Если объект не может быть обнаружен (его дальность не попадает в диапазон дистанций обнаружения), то переходят к рассмотрению следующего объекта. Если объект обнаруживается РЛС, ему как цели присваивается очередной номер, начиная с первой. Число обнаруженных целей совпадает с номером последней обнаруженной цели (для простоты можно считать, что их число не ограничено). Число целей и номер очередной из них записываются в выходной формуляр. Далее определяются сферические координаты цели и признак высоты ее полета. Если высота цели (*Y*) превышает заданное граничное значение (*HG*), признаку высоты присваивается значение *H* (цель высоколетящая) или присваивается значение *N* (цель низколетящая) в противном случае. Наконец, признаку размера цели присваивается признак размера объекта, принимаемого за цель. Все вычисленные значения параметров каждой цели, а также *TD* и *TE* передаются в выходной формуляр. Цикл перебора объектов продолжается до тех пор, пока не выполнится условие ($N\_S \geqslant LS$), после чего осуществляется переход в точку возврата, т. е. в точку общей модели функционирования системы, из которой было обращение к рассмотренной модели РЛС.

Таким образом, улучшенная форма структурограмм на основе объектно-признакового языка позволяет:

- представлять модели функционирования систем и их компонентов в более компактном виде;
- участникам разработки ФПО (заказчику, программисту и пользователю) однозначно и полностью понимать, что должно быть сделано и что делается в функциональной части программно реализуемого проекта;
- сделать алгоритмы управления инвариантными к используемым языкам программирования, что открывает возможность формирования и поддержания библиотек алгоритмов, записанных строго формально.

Список литературы

1. **Современные** методы программирования на языках С и С++: Учеб. пособие / Л. Б. Бузюков, О. Б. Петрова. СПб.: ЛИНК, 2008. 288 с.

2. **Жарковский А. В., Лямкин А. А., Тревгода Т. Ф.** Объектно-признаковый язык описания сложных технических систем // Программная инженерия, 2012. № 2. С. 18—21.

УДК 004.7.056

В. Н. Тарасов, д-р техн. наук, проф., зав. каф.,
Е. М. Мезенцева, аспирант, ассистент,
Поволжский государственный университет телекоммуникаций и информатики, г. Самара,
e-mail: katya-mem@psati.ru, vt@ist.psati.ru

Защита компьютерных сетей. Веб-программирование многомодульного спам-фильтра

Описаны алгоритмы построения спам-фильтра на сайтах по двум методам: Байеса и Фишера. Приведены код программы на языке веб-программирования PHP и результаты тестирования обученного фильтра.

***Ключевые слова**: сообщения, спам, классификация Байеса, априорные знания, метод Фишера, распределение хи-квадрат*

Введение

Для идентификации поступающих на сайт сообщений необходимо построить классификатор, который будет определять степень принадлежности сообщения к одной из трех категорий (спам, не спам, не определенное). В связи с этим необходимо выделить все признаки (слова) в сообщении для анализа, вычислить статистические вероятности для отдельных признаков и объединить все вероятности в одно значение для всего сообщения. В большинстве случаев вероятность принадлежности сообщения к одной из категорий намного выше, чем к остальным, тогда сообщение будет отнесено к этой категории. В научной литературе известны методы Байеса и Фишера, реализованные в фильтрах электронной почты и написанные на таких языках как Lisp и Python [1]. На наш взгляд, также актуальным является вопрос защиты сайтов организаций (в комментариях, форумах и т. д.). При этом возникает задача организации спам-фильтра на языках веб-программирования.

Вычисление объединенных вероятностей признаков

Перед вычислением объединенных вероятностей документов необходимо вычислить вероятность того, что отдельное слово документа принадлежит некоторой категории. Для этого можно разделить найденное число сообщений с признаком i в данной категории, на общее число сообщений в той же категории, но можно воспользоваться другим методом, описанным ниже.

Пусть F_{ai} — число сообщений с признаком i в группе спама; F_{bi} — число сообщений с признаком i в группе не спама.

Тогда статистическая вероятность появления признака i в спам-сообщении

$$p_{ai} = \frac{F_{ai}}{F_{ai} + F_{bi}}, \quad (1)$$

а вероятность появления признака i в не спам сообщении

$$p_{bi} = \frac{F_{bi}}{F_{ai} + F_{bi}}. \quad (2)$$

Таким образом, число сообщений с признаком i в одной из категорий делим на общее число сообщений с данным признаком i.

При использовании формул (1) и (2) учитывается тот факт, что со временем может накопиться одинаковое число сообщений в обеих категориях, т. е. эти формулы не зависят от того, сколько сообщений накоплено в той или иной категории.

Заметим, что приведенные выше формулы дают точный результат только для тех признаков, которые фильтр уже встречал в обеих категориях. Это делает спам-фильтр слишком чувствительным на ранних этапах обучения в отношении к редко встречающимся словам. Чтобы справиться с данной проблемой, вычислим новую вероятность, начиная с предполагаемой априорной вероятности $P_{\text{пр}}$ и веса w, приданного этой вероятности, а затем добавим рассчитанные по формулам (1) и (2) вероятности.

Если вероятность $P_{\text{пр}} = 0{,}5$ и $w = 1$ — вес предполагаемой вероятности равен одному слову, то определяем средневзвешенные вероятности, используя выражения (1), (2):

$$\overline{p_{ai}} = \frac{wP_{\text{пр}} + p_{ai}(F_{ai} + F_{bi})}{w + F_{ai} + F_{bi}},$$

$$\overline{p_{bi}} = \frac{wP_{\text{пр}} + p_{bi}(F_{ai} + F_{bi})}{w + F_{ai} + F_{bi}}.$$

Такой подход позволяет избегать деления на ноль в формуле Байеса, а также учитывать редко встречающиеся слова.

Для получения объединенных вероятностей всего документа (сообщения), будем исходить из словаря, полученного на этапе обучения спам-фильтра. Введем следующие обозначения событий: A — документ относится к спаму; B — документ не спам-сообщение. Предположим, что вероятности независимы, поэтому возможно их перемножение:

$$P(A) = \overline{p_{a1}} \times \overline{p_{a2}} \times \ldots \times \overline{p_{an}} \qquad (3)$$

для вероятности совместного появления слов в спаме;

$$P(B) = \overline{p_{b1}} \times \overline{p_{b2}} \times \ldots \times \overline{p_{bn}} \qquad (4)$$

для вероятности совместного появления слов в не спаме, где n — число слов в документе.

Исходный код реализации представлен в виде функции categorize($message, $method = 'hi'), блок 1 (см. приложение).

Определение спама с помощью формулы Байеса и априорного знания

Для вычисления вероятности того, что сообщение принадлежит одной из двух категорий (спам/не спам) рассмотрим два метода классификации. В этом разделе применим формулы Байеса с использованием априорного знания.

Для любого сообщения введем две гипотезы: H_A — сообщение относится к спаму, H_B — сообщение относится к не спаму.

Введем обозначения:

F_a — общее число сообщений спама;

F_b — общее число не спам-сообщений;

$p_a = \frac{F_a}{F_a + F_b}$ — априорная вероятность, что сообщение окажется спамом;

$p_b = \frac{F_b}{F_a + F_b}$ — априорная вероятность, что сообщение окажется не спамом;

$O_a = \frac{P_a}{1 - P_a}$ — априорные шансы, что сообщение окажется спамом;

$O_b = \frac{P_b}{1 - P_b}$ — априорные шансы, что сообщение окажется не спамом.

Тогда на основе теоремы Байеса с применением априорного знания получим:

$P(H_A) = \frac{P(A)O_a}{P(A)O_a + P(B)O_b}$ — апостериорная вероятность, что это спам-сообщение;

$P(H_B) = \frac{P(B)O_b}{P(A)O_a + P(B)O_b}$ — апостериорная вероятность, что это не спам-сообщение [2]. Здесь вероятности $P(A)$ и $P(B)$ вычисляются по формулам (3) и (4).

Исходный код реализован в виде функции bayes($Pa, $A, $B), блок 5 (см. приложение).

Классификация по методу хи-квадрат Фишера (R. A. Fisher)

Этот метод классификации приведен как альтернатива теореме Байеса, так как он дает существенно более точную оценку вероятности, что может оказаться крайне полезным в отчетах или при выставлении порогов принятия решения.

По методу Фишера все вероятности перемножаются аналогично методу Байеса, но затем от произведения берется натуральный логарифм, и результат умножается на −2. Для этого введем переменную *hisqv*, которая будет определена выражениями, приведенными ниже:

$$hisqv = -2\ln(P(A)) \text{ или } hisqv = -2\ln(P(B)),$$

где вероятности $P(A)$ и $P(B)$ вычисляются по формулам (3) и (4). Исходный код вычисления *hisqv* реализован в блоке 2 (см. приложение).

Фишером доказано, что если есть набор независимых и случайных вероятностей (3) или (4), то ве-

личина $-2\ln(P(A))$ подчиняется распределению χ^2 с $2n$ степенями свободы (n — число слов в документе):

$$F(x) = \int_0^x \frac{t^{n-1}e^{-t/2}}{2^n\Gamma(n)}dt, \qquad (5)$$

где $\Gamma(n)$ — гамма-функция.

С учетом сказанного выше и представления гамма-функции от четного аргумента перепишем интеграл (5) в виде (6)*:

$$F(hisqv) = \frac{1}{2^n(n-1)!q} \int_0^{hisqv} x^{n-1}e^{-x/2}dx. \qquad (6)$$

В программе вычисление вероятности по выражению (6), т. е. обратной функции распределения хи-квадрат, выполняется с помощью квадратурной формулы Гаусса с 15 узлами:

$$\int_a^b f(t)dt \approx \frac{b-a}{2} \sum_{i=1}^{n} A_i f(t_i),$$

где $t_i = (b + a)/2 + (b - a)x_i/2$, а x_i — узлы квадратурной формулы Гаусса; A_i — гауссовы коэффициенты, ($i = 1, 2, ..., 15$) [3]. Для нашего случая $a = 0$, $b = hisqv$.

Число, возвращаемое функцией $F(hisqv)$, будет малым в случае, если в тексте много признаков спама. Для верной классификации сообщения необходим обратный результат. Для этого от единицы вычтем значение $F(hisqv)$. Соответственно, вычитая из единицы значение функции $F(hisqv)$ для большого количества не спам-признаков, получим вероятность того, что сообщение является не спамом. Исходный код реализован в виде функции hiSquare($b, $n) в блоке 4 (см. приложение).

Однако метод Фишера не является симметричным. Значит, необходимо скомбинировать вероятности спама и не спама, путем объединения вероятностей в одно число, которое даст нам значение спам/не спам от 0 до 1.

Для этого воспользуемся индикатором Фишера:

$$I = \frac{1 + P\left(H'_A\right) - P\left(H'_B\right)}{2},$$

где $P\left(H'_A\right) = 1 - F(-2\ln(P(A))$ — вероятность принадлежности документа к спаму; $P\left(H'_B\right) = 1 - F(-2\ln(P(B))$ — вероятность принадлежности документа к не спаму.

Если значение I близко к 1, то сообщение будет отнесено к спаму, а если близко к нулю, то — к не спаму. Исходный код реализован в блоке 3 (см. приложение).

Тестирование модуля распределения хи-квадрат

При тестировании работы функции хи-квадрат на вход подаются значения χ^2 и число степеней свободы n, на выходе получаем вероятности. Сравнение выходных данных с табличными значениями вероятностей, приведенными в таблице, подтверждает их полную

Тестирование вычисления распределения хи-квадрат

Данные на входе функции распределения хи-квадрат hiSquare (χ^2, n)	Выходные данные	Табличные значения
3,9403, 10	0,05	0,05
9,3418, 10	0,50	0,50
18,3070, 10	0,95	0,95
10,8508, 20	0,05	0,05
19,3374, 20	0,50	0,50
31,4104, 20	0,95	0,95
18,4927, 30	0,05	0,05
29,3360, 30	0,50	0,50
43,7730, 30	0,95	0,95
26,5093, 40	0,05	0,05
39,3353, 40	0,50	0,50
55,7585, 40	0,95	0,95
34,7643,50	0,05	0,05
49,3349, 50	0,50	0,50
67,5048, 50	0,95	0,95

* Значение $(n - 1)!$ в отдельности и подинтегральной функции (6) в целом могут вызвать ошибку переполнения. В связи с этим, их вычисление в программе реализовано по реккурентной формуле.

идентичность. При этом табличные значения вероятностей в свою очередь рассчитаны с помощью функции chi2inv пакета MATLAB.

Пороги принятия решений

В методах классификации Байеса и Фишера необходимо задавать начальные значения нижнего и верхнего порогов для окончательного принятия решений. Пусть T и L величины, измеряющиеся в процентах и определяющие соответственно верхний и нижний пороги принятия решений.

Тогда будем считать, что документ принадлежит группе H, если $P(H) \geqslant T$; документ не принадлежит группе H, если $P(H) \leqslant L$; если же выполняется $T \geqslant P(H) \leqslant L$, то нельзя принять никакого решения.

Экспериментальное исследование степени корректности фильтрации сообщений

Рассмотренные в данной статье методы способны обучаться тому, как надо классифицировать сообщения. Классификатору необходимо предоставить примеры правильных ответов. Чем больше сообщений и верных способов классификации мы предоставим приведенным алгоритмам, тем точнее получится итоговая классификация документов.

Поэтому для тестирования работы рассмотренных выше алгоритмов было проведено обучение нашей системы фильтрации спама на 300 сообщениях спама и 500 сообщениях не спама.

Тестирование работы фильтра при различных порогах принятия решения показало, что самыми оптимальными являются:

- верхняя граница $T = 0{,}95$;
- нижняя граница $L = 0{,}4$.

Мы установили жесткие рамки по спаму и обычные для не спама. Это было сделано для того, чтобы как можно меньше нормальных сообщений проходило в спам, т. е. для избежания ложных срабатываний.

После определения порогов принятия решения переходим к тестированию работы выбранных методов фильтрации спама.

Подаем на вход фильтра 400 спам-сообщений — методы Байеса и Фишера классифицировали все сообщения как спам (100 % вероятность).

Тестируем на 200 не спам-сообщений.

Метод Байеса:

- спам-сообщения — 11 сообщений (5,5 %);
- не спам-сообщения — 171 сообщение (85,5 %);
- неопределенные — 18 сообщений (9 %).

Метод хи-квадрат Фишера:

- спам-сообщения — 2 сообщения (1 %);
- не спам-сообщения — 157 сообщений (78,5 %);
- неопределенные — 41 сообщение (20,5 %).

Проверяем работу методов на 200 новых перемешанных сообщениях (135 спам/65 не спам).

Метод Байеса:

- спам-сообщения — 137 сообщений;
- не спам-сообщения — 57 сообщений;
- неопределенные — 6 сообщений.

Возникли два ложных срабатывания — нужные сообщения были оценены как спам, все 135 спам-сообщений определены верно.

Метод хи-квадрат Фишера:

- спам-сообщение — 134 сообщения;
- не спам-сообщения — 54 сообщения;
- неопределенные — 12 сообщений.

В результате тестирования ложных срабатываний и отнесения нормальных сообщений к спаму не было. Одно спам-сообщение отмечено как неопределенное.

Оба метода верно оценили два действительно неопределенных сообщения, которые были не информативны.

Заключение

По результатам исследования мы выявили, что метод хи-квадрат Фишера дает более точные результаты даже при обучении фильтра на небольшой выборке сообщений и сводит к минимуму возникновение ложных срабатываний.

Процент ложных срабатываний (ошибочная классификация нужных сообщений как спам) и доля пропуска спама являются основными критериями при оценке работы алгоритмов фильтрации спама. Полностью избавиться от них невозможно, но необходимо максимально уменьшить их число. Этого мы и добились, применяя алгоритмы фильтрации спама по методам Байеса и Фишера. Даже при обучении на небольшой выборке сообщений процент ложных срабатываний не велик, а чувствительность этих методов (доля выявленных спам-сообщений) приближается к 100 %.

Хоть алгоритм хи-квадрат Фишера и показал результаты тестирования лучшие, чем метод Байеса, отказываться от последнего не следует, так как при дальнейшем обучении спам-фильтра он может показать иной результат. Поэтому необходимо делать окончательный вывод, посредством "коллективного решения", скомбинировав два этих алгоритма.

Список литературы

1. **Seibel P.** Practical Common Lisp. New York: Apress, 2005. 528 с.

2. **Джарратано Д., Райли Г.** Экспертные системы. Принципы разработки и программирование, 4-е издание. М: Вильямс, 2007. 1147 с.

3. **Никольский С. М.** Квадратурные формулы. М.: Наука, 1974. 224 с.

Приложение

Код программы на языке PHP

```
/**
    * Классификация сообщения $message
    * @param string $message Текст сообщения
    * @param string $method Метод классификации, по
      умолчанию хи-квадрат
    * @return array
    */
    function categorize($message, $method = 'hi')
    {
      // Вес и начальная вероятность признака
      $weight = 1;
      $ap = 0.5;
      // Получение всех признаков документа
      $tokenizer = new Tokenizer();
      $tokens = $tokenizer->tokenize($message, 'UTF-8');
      if (!count($tokens))
      {
        // Документ не содержит признаков
        return array(0.5, 0.5);
        }
       // Исключение общеупотребительных слов
      $tokens=$tokenizer->excludeTokens($tokens,'UTF-8');
      if (!count($tokens))
      {
        // Документ не содержит признаков после про-
         цедуры исключения
        return array(0.5, 0.5);
      }
      // Применение алгоритма OSB
      $osb = $tokenizer->createOSB($tokens);
      $tokens = array_merge($tokens, $osb);
     $storage = new Storage();
      //Априорная вероятность
      $apriory = $storage->getProbabilities();
      if (!$apriory[0]['tokens_count'] || !$apriory[1]['tokens_
     count'])
      {
     // Априорная вероятность неизвестна, нет сооб-
     //щений в одной из категорий
        return array(0.5, 0.5);
      }
      // Считываем частоты признаков из базы данных
      $tokens = $storage->getTokensCollection($tokens);
      $countTokens = count($tokens);
      if (!$countTokens)
      {
       // В базе данных нет ни одного признака документа
        return array(0.5, 0.5);
      }
    // ---------------------------------------------- 1
    // ----Вычисление взвешенных вероятностей каж-
    // дого признака
    for ($i = 0; $i < $countTokens; $i + +)
    {
      $totals = $tokens[$i]['spam_cnt'] + $tokens[$i]
['nospam_cnt'];
      $sbasicprob = $tokens[$i]['spam_cnt'] / $totals;
      $nsbasicprob = $tokens[$i]['nospam_cnt'] / $totals;
      $tokens[$i]['s_wprob'] = (($weight * $ap) + ($totals *
$sbasicprob)) / ($weight + $totals);
      $tokens[$i]['ns_wprob'] = (($weight * $ap) + ($totals *
$nsbasicprob)) / ($weight + $totals);
    }
      $A = 1;
      $B = 1;
      // Вычисление вероятностей совместного появ-
      // ления признаков в обеих категориях
      foreach ($tokens as $t)
    {
      $A * = $t['s_wprob'];
      $B * = $t['ns_wprob'];
    }
    // Исключение деления на ноль в будущих вычис
    // лениях
    $A = $this->fakeNull($A);
    $B = $this->fakeNull($B);
    // Классификация по выбранному методу
    if ($method = = 'bayes')
    {
      $Pa = $apriory[0]['tokens_count'] / ($apriory[0]
['tokens_count'] + $apriory[1]['tokens_count']);
      $PHa = $this->bayes($Pa, $A, $B);
      $PHb = 1 - $PHa;
    } else if ($method = = 'hi')
    {
    // ------------------------------------------------ 2
       $fscoreA = -2 * log($A);
       $fscoreB = -2 * log($B);
    // ------------------------------------------------3
       $PHa = $this->hiSquare($fscoreA, $countTokens * 2);
       $PHb = $this->hiSquare($fscoreB, $countTokens * 2);
       // Индикатор Фишера
       $I = (1 + $PHa - $PHb) / 2;
       $PHa = $I;
       $PHb = 1 - $PHa;
    } else
    {
      // Выбран несуществующий метод классифика-
      // ции
      return array(0.5, 0.5);
    }
    return array($PHa, $PHb);
    }
    /**
```

```
    * "Системный ноль". Функция возвращает 1.0E-
  10, если $value = 0
    * @param mixed $value число
    * @return float
    */
    function fakeNull($value)
  {
  $fakeNull = 1 * pow(10, -10);
  $value = = 0 ? $value = $fakeNull : null;
  return $value;
  }
  // ------------------------------------------------4
  /**
    * Классификация документа с помощью распре-
деления хи-квадрат Фишера
    * @param float $b Верхний предел интегрирования
    * @param int $n Количество степеней свободы
    * @return float
    */
  function hiSquare($b, $n)
  {
    // Делим количество степеней свободы на 2
    $n = $n / 2;
    // Нижний предел интегрирования равен нулю
    $a = 0;
    // Гауссовы коэффициенты
  $ag = array(0.0307532420, 0.0703660475,
    0.1071592205, 0.1395706779,
    0.1662692058, 0.1861610000,
    0.1984314853, 0.2025782419,
    0.1984314853, 0.1861610000,
    0.1662692058, 0.1395706779,
    0.1071592205, 0.0703660475, 0.0307532420);
  // Узлы квадратурной формулы Гаусса
  $xg = array(-0.9879925180, -0.9372733924,
    -0.8482065834, -0.7244177314,
    -0.5709721726, -0.3941513471,
    -0.2011940940, 0.0,
    0.2011940940, 0.3941513471,
    0.5709721726, 0.7244177314,
    0.8482065834, 0.9372733924, 0.9879925180);
  // Численное интегрирование с использованием
  // квадратурной формулы Гаусса
  $a1 = ($b + $a) / 2;
  $a2 = ($b - $a) / 2;
  $hi = 0;
  for ($i = 0; $i < 15; $i + +)
  {
    $x = $a1 + $a2 * $xg[$i];
    // $gauss + = $ag[$i] * (1 / (pow(2, $n) * $n2) *
    // pow($x, $n - 1) * exp(-$x / 2));
    // Используем для вычисления факториала и
    // возведения в степень рекуррентную формулу
    // во избежание переполнения
    $m = exp(-$x / 2) / 2;
    for ($j = 1; $j < = $n - 1; $j ++)
    {
      $m * = $x / ($j * 2);
    }
      $hi + = $ag[$i] * $m;
  }
  $hi * = $a2;
  // конец интегрирования
  // 1 - хи-квадрат
  $hi = min($hi, 1);
  $hi = abs(1 - $hi);
  return $hi;
  }
// ------------------------------------------------ 5
  /**
    * Классификация документа по формулам Байеса
с учетом априорных знаний
    * @param float $Pa Априорная вероятность попа-
дания сообщения в категорию "спам"
    * @param float $A Вероятность совместного появ-
ления признаков сообщения в категории "спам"
    * @param float $B Вероятность совместного появ-
ления признаков сообщения в категории "нужные"
    * @return float
    */
  function bayes($Pa, $A, $B)
  {
    $Pb = 1 - $Pa;
    $Oa = $Pa / (1 - $Pa);
    $Ob = $Pb / (1 - $Pb);
    $PHa = $A * $Oa / ($A * $Oa + $B * $Ob);
    return $PHa;
  }
```

УДК 004.41, 004.42, 511, 519.6

Д. А. Орлов, канд. техн. наук, ассистент,
Московский энергетический институт (технический университет),
e-mail: orlovdmal@gmail.com

Реализация арифметики повышенной разрядности на графических процессорах*

Показана возможность применения графических процессоров (GPU) для работы с числами большой разрядности. Предложены два варианта алгоритмов выполнения арифметических операций над числами повышенной разрядности на GPU. Указана область их возможного применения.

***Ключевые слова**: графические процессоры, параллельные вычисления, машинная арифметика, CUDA, GP GPU, числа большой разрядности*

Введение

Задача работы с числами большой разрядности является важной для многих приложений. Например, для криптографии и для высокоточных вычислений. Высокоточные вычисления применяются в случаях, если задача является чувствительной к вычислительным ошибкам. Прежде всего, это характерно для алгоритмов вычислительной геометрии, использующихся в САПР [1]. Поскольку современные графические процессоры (GPU) широко доступны, их пиковая производительность зачастую превышает пиковую производительность центральных процессоров [2], а направление использования графических процессоров для решения вычислительных задач (*general purpose computations on GPU*, GP GPU) активно развивается, использование графических процессоров для высокоточных вычислений представляется актуальной и интересной задачей.

В этой статье будут предложены варианты реализации работы с целыми числами большой разрядности на GPU архитектуры CUDA, разработанной компанией NVIDIA.

* Работа выполнена при поддержке РФФИ (грант № 11-07-00751-а) и грантом Президента РФ для молодых кандидатов наук: МК-65190.2010.9.

Архитектура NVIDIA CUDA

В данном исследовании рассматриваются графические процессоры, имеющие архитектуру CUDA как наиболее распространенные. Рассмотрим эту архитектуру подробнее. CUDA (*Compute Unified Device Architecture*, унифицированная архитектура вычислительных устройств) является разработкой фирмы NVIDIA. Вычислительное устройство CUDA представлено либо видеокартой, либо специализированным вычислителем (например, NVIDIA Tesla). Вычислительное устройство CUDA имеет несколько потоковых мультипроцессоров (*SM*) и собственную оперативную память (GPU RAM). Упрощенная структурная схема вычислительного устройства CUDA [3] представлена на рис. 1.

Каждый потоковый мультипроцессор (*SM* 1, ..., *SM N*) имеет несколько скалярных процессоров (*SP* 1, ..., *SP M*). Каждый скалярный процессор имеет свою собственную регистровую память (*Registers*). Скалярные процессоры одного и того же потокового мультипроцессора могут одновременно выполнять одну и ту же команду, поскольку они имеют общий блок выборки инструкций (*Instruction Unit*). Для ускорения выборки команд из памяти в скалярном мультипроцессоре имеется кэш команд (*Instruction Cache*). Каждый потоковый мультипроцессор имеет блок разделяемой памяти (*Shared Memory*), доступной всем скалярным

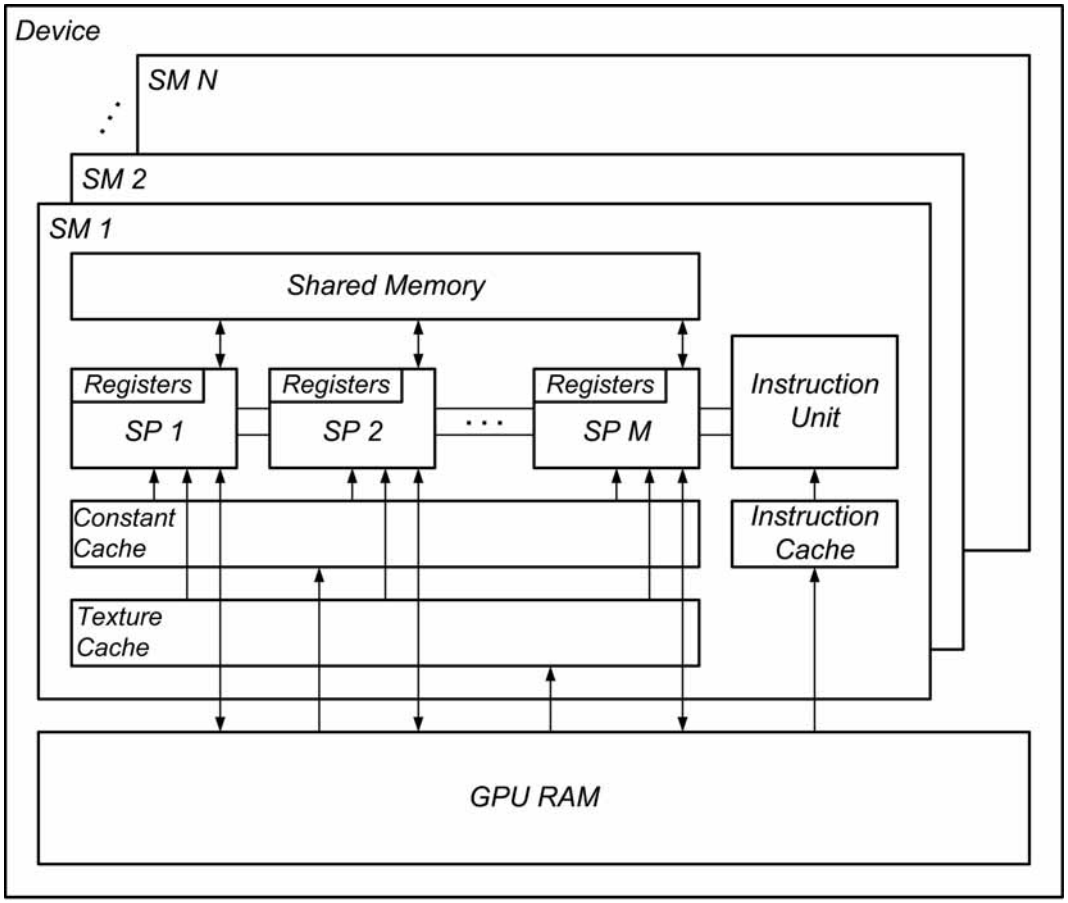


Рис. 1. Упрощенная структурная схема вычислительного устройства CUDA

процессорам для чтения и записи. Все скалярные процессоры имеют возможность считывания и записи из GPU RAM. Отметим, что быстродействие GPU RAM значительно ниже, чем быстродействие регистровой или разделяемой памяти, кроме того, латентность GPU RAM значительно выше, чем у регистровой или разделяемой памяти. Несмотря на это, доступ к GPU RAM не кэшируется. Кроме того, существует кэш констант (*Constant Cache*) и кэш текстур (*Texture Cache*).

На каждом мультипроцессоре может выполняться большое число потоков. Потоки выполняются в режиме вытесняющей многозадачности. Отличие от реализации переключения потоков на центральных процессорах состоит в том, что на вычислительном устройстве CUDA имеется аппаратный планировщик, работающий на уровне отдельных команд. Кроме того, ресурсы между потоками распределяются на этапе компиляции программы, поэтому при переключении потоков практически отсутствует необходимость переключения контекста [3]. Таким образом, переключение потоков почти не требует накладных расходов. Для полноценной загрузки вычислительного устройства CUDA необходим запуск большого числа потоков, благодаря чему маскируются задержки доступа к GPU RAM.

Программная модель CUDA рассматривает вычислительное устройство CUDA (в терминологии документации называемое *Device*) как вычислительное устройство, имеющее возможность исполнять достаточно большое число потоков параллельно [3]. Вычислительное устройство CUDA работает в качестве сопроцессора центрального процессора (CPU). CPU в терминологии документации CUDA называется хостом (*host*). Хост и вычислительное устройство CUDA имеют каждый свою собственную оперативную память. Между оперативной памятью вычислительного устройства CUDA и оперативной памятью хоста возможен обмен данными, для чего предусмотрены вызовы программного интерфейса CUDA. Инициатором обмена данными между хостом и вычислительным устройством CUDA может быть только хост.

Программа, исполняющаяся на вычислительном устройстве CUDA, называется ядром (*kernel*). Выполняющиеся потоки ядра организованы в решетку (*grid*), состоящую из блоков (*block*). Блок — это множество потоков, которые могут взаимодействовать посредством быстродействующей разделяемой памяти (*shared*

memory) и синхронизации. Поддерживается барьерная синхронизация потоков блока. Следует отметить, что все потоки одного блока выполняются на одном и том же потоковом мультипроцессоре. Потоки разных блоков могут общаться между собой только через GPU RAM. Отметим, что все потоки ядра выполняют один и тот же код, а все блоки ядра имеют один и тот же размер. Каждый блок в решетке имеет свой одно- или двухкомпонентный индекс, а каждый поток (*thread*) в блоке — одно-, двух- или трехкомпонентный индекс.

Схема организации программы, использующей CUDA, показана на рис. 2.

Для вычислительного устройства CUDA определена следующая модель памяти. Поток, выполняющийся на устройстве, имеет доступ только к видам памяти, перечисленным в табл. 1 (физически расположенным либо непосредственно на чипе GPU, либо в DRAM устройства). В табл. 1 также приведены свойства всех типов памяти, использующихся в CUDA.

Регистровая память расположена на чипе, поэтому обеспечивает наименьшее время доступа. В версиях архитектуры CUDA 1.0-1.1 на один мультипро-

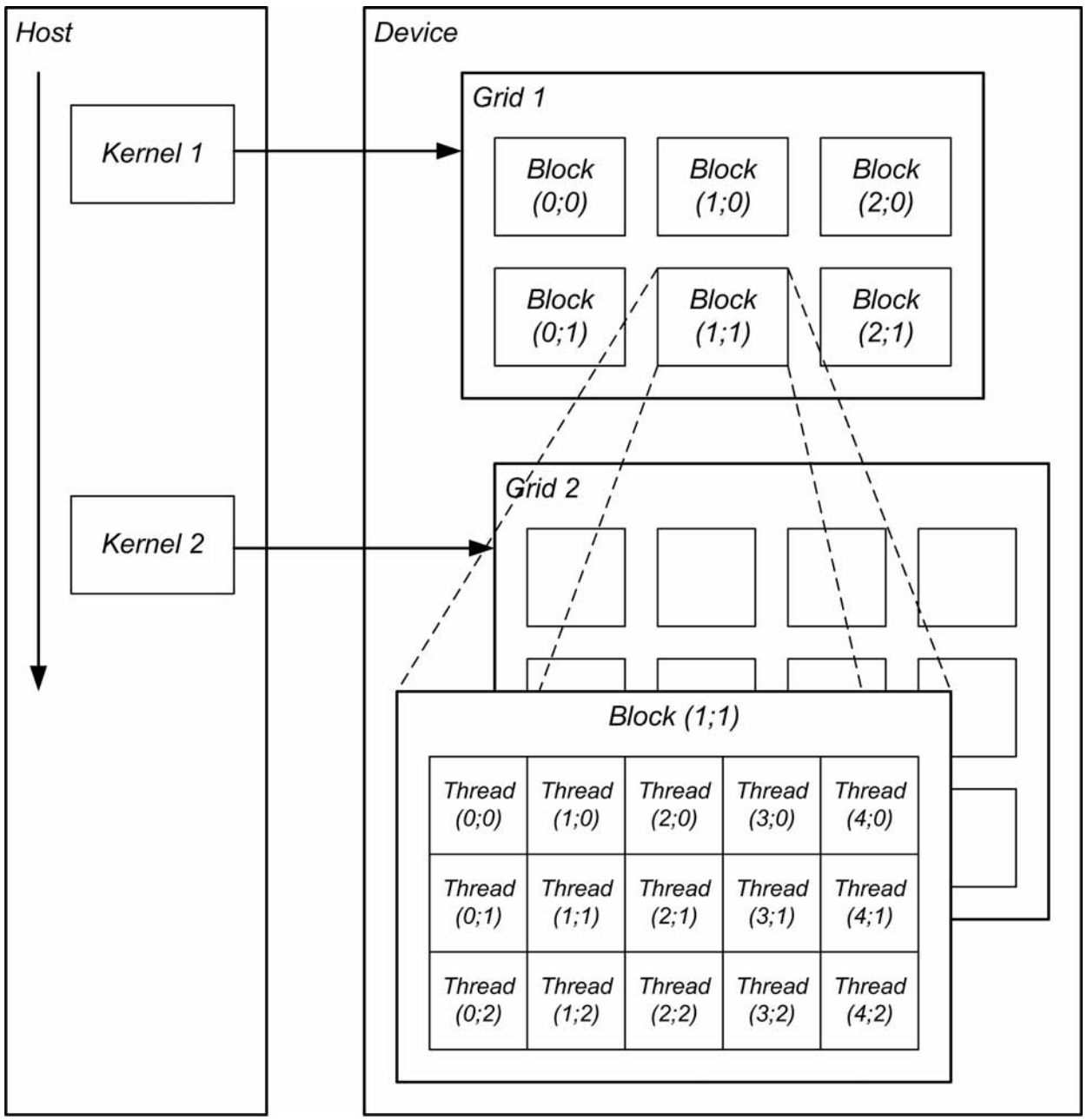


Рис. 2. Схема организации программы на CUDA

Таблица 1

Основные характеристики типов памяти, использующихся в CUDA

Тип памяти	Доступ из устройства	Доступ из хоста	Уровень выделения	Скорость работы
Регистровая	Чтение/запись	Нет	Поток	Высокая (на чипе)
Локальная	Чтение/запись	Нет	Поток	Низкая (GPU RAM)
Разделяемая	Чтение/запись	Нет	Блок	Высокая (на чипе)
Глобальная	Чтение/запись	Чтение/запись	Решетка	Низкая (GPU RAM)
Константная	Только чтение	Чтение/запись	Решетка	Высокая (кэшируется)
Текстурная	Только чтение	Чтение/запись	Решетка	Высокая (кэшируется)

цессор доступно 8192 32-разрядных регистра (в версиях 1.2-1.3 — 16384).

Локальная память необходима, если для работы потока недостаточно емкости регистровой памяти, либо если необходим доступ к локальным данным по индексу (поскольку регистровая память не поддерживает индексации). Локальная память, как и глобальная, расположена в GPU RAM.

Разделяемая память — это память, доступная всем потокам одного блока. Ее можно использовать для обмена данными только между потоками одного блока. Разделяемая память это некэшируемая, но быстрая память. В некоторых случаях ее рекомендуется использовать как управляемый кэш. На один мультипроцессор доступно 16 кбайт разделяемой памяти. Разделив это число на количество блоков в решетке, получим максимальную емкость разделяемой памяти, доступной для одного блока. Гарантируется, что во время исполнения блока на мультипроцессоре содержимое разделяемой памяти будет сохраняться. Однако сохранение информации в разделяемой памяти после выполнения блока не гарантируется. Поэтому не стоит пытаться организовывать обмен данными между блоками, оставляя в разделяемой памяти какие-либо данные и надеясь на их сохранность.

Глобальная память, память констант и текстурная память физически расположены в GPU RAM, однако оптимизированы для различных сценариев использования. Глобальная память не кэшируется. Она работает медленно, поэтому число обращений к глобальной памяти следует минимизировать. Глобальная память необходима прежде всего для сохранения результатов работы программы перед отправкой их на хост. Отличие памяти констант от глобальной памяти состоит в том, что из ядра возможно только чтение константной памяти. Однако память констант имеет кэш. Это означает, что скорость доступа к ней выше, чем к глобальной памяти. Один из возможных сценариев использования памяти констант — передача в ядро массива данных, который не изменяется во время выполнения ядра.

Текстурная память имеет особый режим адресации. При доступе к ней выявление и обработка ситуации выхода адреса данных за границы массива происходит автоматически. Кроме того, при обращении к текстурной памяти можно использовать дополнительную обработку данных (например, поддерживается интерполяция данных).

Глобальная память, память констант и текстурная память могут быть считаны и записаны хостом. Содержимое этих видов памяти сохраняется во время работы приложения, выполняющегося на хосте.

Основные расширения CUDA для языка *C++*:

- ключевые слова, позволяющие задать способ выполнения функции;
- ключевые слова, позволяющие задать тип памяти для каждой из переменных;
- конструкция, задающая конфигурацию исполнения для вычислительных ядер;
- переменные, содержащие индекс потока в блоке, индекс блока в решетке, а также размеры блока и решетки.

Организация вычислительного устройства CUDA в документации имеет название *SIMT* (*single instruction*, *multiple threads*), одна инструкция, множество потоков. *SIMT*-организация похожа на *SIMD*-организацию тем, что одна инструкция исполняется многократно для разных данных. Ключевая разница между ними состоит в том, что *SIMT*-организация имеет аппаратный планировщик, позволяющий выполнять несколько ядер на одном устройстве. Кроме того, разные блоки одного и того же ядра могут выполняться на разных мультипроцессорах. Таким образом, распределение потоков по процессорам происходит автоматически, а программист при разработке ядра лишь определяет код потоков.

Основные ограничения, которые накладывает архитектура CUDA на вычислительные ядра:

- невозможность выделения памяти непосредственно из вычислительного ядра;
- невозможность инициализации обмена данными с хостом из ядра;
- отсутствие средств синхронизации вне блока потоков. Единственный способ взаимодействия между потоками из разных блоков — через GPU RAM;

- отсутствие поддержки рекурсии, ограниченная поддержка вызовов процедур. Как следствие, отсутствие поддержки виртуальных функций классов;
- ограниченная поддержка чисел с плавающей запятой двойной точности.

Перечисленные ограничения не позволяют эффективно организовать на вычислительном устройстве CUDA работу с целыми числами изменяемой разрядности, так как при сохранении результата каждой арифметической операции в этом случае требуется динамическое выделение памяти для хранения результата. Необходимость этого обусловлена тем, что количество разрядов, требующееся для хранения результата операции, заранее неизвестно.

Организация многопотоковых вычислений

Для эффективного использования ресурсов GPU при работе с числами большой разрядности необходимо изменить представление числа, поскольку при выполнении операции сложения над числами, представленными в традиционной позиционной системе счисления, значение переноса вычисляется последовательно. Следовательно, в этом случае операцию над каждой парой аргументов должен выполнять ровно один поток. Значит, аргументы и результат должны находиться в глобальной или локальной памяти (так как регистровая память устройства CUDA не поддерживает индексации). Такая реализация не будет эффективной на устройствах CUDA, так как на устройствах CUDA доступ к GPU RAM достаточно затратен. Кроме того, использование только параллелизма по данным на GPU затруднено из-за ограниченных возможностей выделения памяти, так как память (в том числе и под промежуточные данные) необходимо выделять вручную, перед запуском вычислений.

Поэтому для устройств CUDA оправдан подход, при котором с каждой парой операндов работает свой блок потоков. Кроме того, этот подход позволит обеспечить больший уровень распараллеливания (распараллеливание как по данным, так и по группам разрядов). Это означает, что необходимо изменить представление чисел так, чтобы арифметические операции могли бы быть распараллелены. Схема организации вычислений с распараллеливанием арифметических операций по группам разрядов показана на рис 3.

Схема организации вычислений состоит в следующем. Входные данные, находящиеся в памяти графического процессора (GPU RAM), представлены в одном из аппаратно поддерживаемых форматов. Например, они могут быть представлены 32-битными целыми числами или числами с плавающей запятой (как одинарной, так и двойной точности). При запуске вычислений на графическом процессоре происходит загрузка переменной из GPU RAM и преобразование переменной в формат, допускающий поразрядное распараллеливание арифметических операций. В результате каждый из потоков (GPU *threads*) будет содержать только часть информации о значении переменной. Для получения значения вычисленного выражения либо результата сравнения вычисленного выражения с каким-либо числом необходимо получить информацию от всех потоков. Одним из этапов сравнения чисел, представленных в формате, допускающем распараллеливание арифметических операций по группам разрядов, будет параллельная редукция.

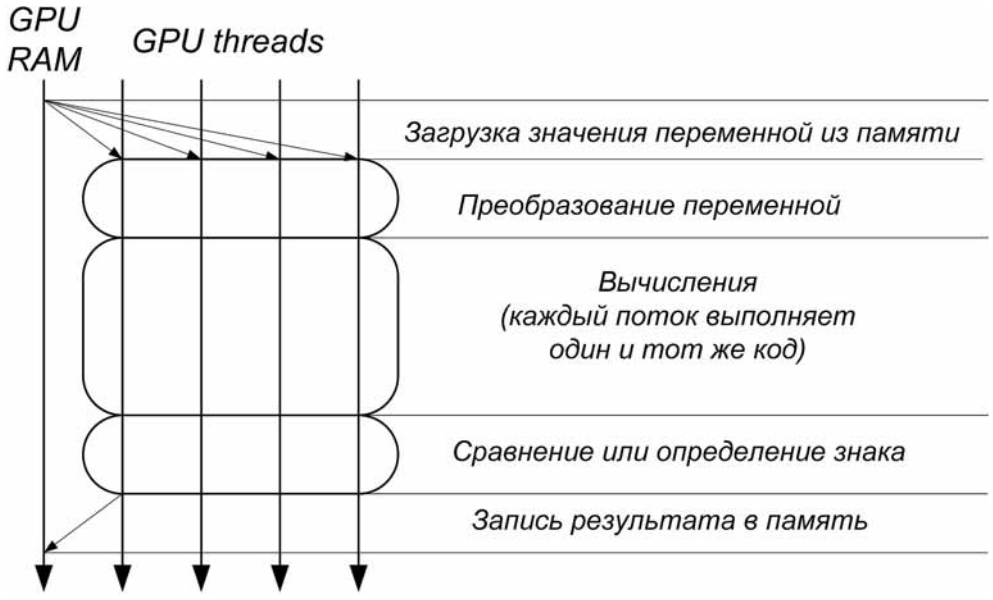


Рис. 3. Схема организации вычислений с поразрядным параллелизмом

Основным недостатком данного подхода является необходимость изменения формата представления чисел, так как при использовании позиционной системы счисления перенос в следующий разряд вычисляется последовательно.

Существуют два основных способа представления целых чисел, допускающих поразрядное распараллеливание арифметических операций:

- представление числа в знакоразрядной системе [4];
- представление числа в многомодульной системе [5].

Рассмотрим каждый из них подробнее.

Реализация вычислений в знакоразрядной системе счисления

Знакоразрядная система счисления отличается от обычной позиционной тем, что разряды в ней могут принимать и отрицательные значения.

Определение. Число x в знакоразрядной системе счисления с основанием q имеет следующую запись:

$$x = \overline{d_{n-1} \dots d_1 d_0},$$

где n — количество разрядов числа; d_i — цифры в записи числа, $d_i \in \{-q + 1, \dots, 0, \dots, q - 1\}$.

При этом

$$x = d_0 q^0 + d_1 q^1 + \dots + d_{n-1} q^{n-1}.$$

Обычно в записи числа, представленного в знакоразрядной системе счисления, отрицательные цифры обозначают с чертой наверху. Очевидно, что у одного и того же числа может быть несколько представлений в знакоразрядной системе. Например, для $q = 10$:

$$1 = 1\overline{9} = 1\overline{99} = \dots$$

Перечислим некоторые важные свойства знакоразрядной системы счисления:

- для получения противоположного числа достаточно изменить знак всех цифр числа;
- знак числа равен знаку его старшего ненулевого разряда;
- число 0 имеет единственное представление.

Важным свойством знакоразрядной системы счисления является возможность распараллеливания сложения чисел, представленных в ней, поскольку можно ограничить распространение переноса следующим разрядом [4].

Вычисление суммы $s = \overline{s_{n-1} \dots s_1 s_0}$ чисел $a = \overline{a_{n-1} \dots a_1 a_0}$ и $b = \overline{b_{n-1} \dots b_1 b_0}$, представленных в знакоразрядной системе счисления, можно свести к четырем операциям.

1. Вычисление частичных сумм разрядов (обозначим частичные суммы как p_i):

$$p_i = a_i + b_i.$$

2. Вычисление переносов

$$c_i = \begin{cases} 0, & -q - 1 < p_i < q - 1 \\ 1, & p_i \geqslant q - 1 \\ -1, & p_i \leqslant -q - 1. \end{cases}$$

3. Вычисление вспомогательной величины w_i:

$$w_i = p_i - c_i q.$$

4. Вычисление разрядов суммы

$$s_i = \begin{cases} w_i, & i = 0 \\ w_i + c_{i-1}, & i \neq 0 \end{cases}.$$

В данном алгоритме существует только один обмен между потоками — при вычислении разрядов суммы s_i.

Перенос при сложении чисел, представленных в знакоразрядной системе счисления, распространяется не далее, чем на следующий разряд, поскольку $|w_i| < q - 1$ и $|c_i| \leqslant 1$, то $|s_i| = |w_i + c_{i-1}| < q$. Значит, $\overline{s_{n-1} \dots s_1 s_0}$ — корректное представление числа s в знакоразрядной системе счисления.

Если все потоки могут выполняться параллельно, то время выполнения алгоритма сложения чисел, представленных в знакоразрядной системе счисления, не будет зависеть от количества разрядов числа.

Отметим, что вычитание чисел $a = \overline{a_{n-1} \dots a_1 a_0}$ и $b = \overline{b_{n-1} \dots b_1 b_0}$, представленных в знакоразрядной системе счисления, сводится к сложению чисел $a = \overline{a_{n-1} \dots a_1 a_0}$ и $-b = \overline{-b_{n-1} \dots -b_1 - b_0}$.

Алгоритм умножения чисел, представленных в знакоразрядной системе счисления, аналогичен алгоритму умножения чисел, представленных в обычной позиционной системе счисления. Отличие состоит лишь в том, что суммирование частичных произведений чисел, представленных в знакоразрядной системе счисления, выполняется по описанному выше алгоритму сложения чисел, представленных в знакоразрядной системе счисления. Алгоритм для чисел, представленных в знакоразрядной системе, приведен ниже:

```
Алгоритм Mul_IrrNum (A, B, N, q, j)
   Вход: A — множимое
   B — множитель
   N — количество разрядов множимого и множителя
   j — индекс данного потока (принимает значения от 0 до N-1)
   q — основание системы счисления
начало
   SM — значение сумматора
```

```
   SM = 0
   Для i от N-1 до 0 цикл: // все разряды множителя (от старшего к младшему)
      PP — частичное произведение
      PM — разряды частичного произведения без учета переносов
      // PM[j] - разряд, вычисляемый данным потоком
      CR — переносы, возникшие при вычислении частичного произведения
      // CR[j] - перенос, вычисляемый данным потоком
      // B[i] — текущий разряд множителя
      // A[j] — разряд множимого, используемый данным потоком
      CR[j] = A[j]*B[i]/q
      PM[j] = A[j]*B[i] %q
      Сдвиг CR на 1 разряд влево
      PP = PM+CR
      Сдвиг сумматора (SM) на 1 разряд влево
      SM = SM+PP
   конец цикла
   вернуть SM
конец
```

В данном алгоритме все операции сложения выполняются по алгоритму сложения чисел, представленных в знакоразрядной системе счисления. Частичное произведение представим двумя числами: одно содержит разряды произведения без учета переносов, другое — переносы. В модифицированном алгоритме операции сложения, сдвига, вычисления частичного произведения могут выполняться параллельно n потоками, где n — разрядность множимого и множителя.

Сдвиг числа, представленного в знакоразрядной системе счисления, имеет особенность. Если для представления чисел используется разрядная сетка фиксированной длины, то при сдвиге числа возможна потеря данных, даже если результат этой операции можно представить в данной разрядной сетке.

Пример. Пусть разрядная сетка имеет длину 4, а основание системы равно 10. Очевидно, что в данном случае имеем следующий диапазон представления целых чисел: $-9999 \leqslant x \leqslant 9999$. Возьмем число, равное 1. Оно может быть представлено в знакоразрядной системе (при длине разрядной сетки, равной 4) тремя способами: 1; $1\overline{9}$; $1\overline{99}$; $1\overline{999}$. Очевидно, что сдвиг числа, представленного в знакоразрядной системе, на один разряд влево эквивалентен умножению числа на основание системы счисления (в предположении, что разрядная сетка бесконечна). Для всех возможных представлений числа 1 в знакоразрядной системе счисления (при условии, что число разрядов равно четырем) выполним сдвиг на один разряд влево. Получим следующие числа: 10; $1\overline{9}0$; $1\overline{99}0$; $\overline{9990}$. Во всех случаях, кроме последнего, имеем верный результат 10. В последнем случае имеем неверный результат — 9990, несмотря на то, что результат сдвига (число 10) представим в используемой разрядной сетке. Это означает, что перед сдвигом числа, представленного в знакоразрядной системе счисления, на один разряд влево необходимо провести частичную нормализацию представления числа. Алгоритм выполнения этой операции на псевдокоде приведен ниже.

```
Алгоритм Partial_norm(D, N, q)
Вход: D — обрабатываемое число
   // d[0], d[1], ..., d[N-1] — разряды обрабатываемого числа
   q — основание системы счисления
начало
   если d[N-1] = = 1 и d[N-2] < 0, то
        d[N-1] = 0;
        d[N-2] = d[N-2]+q;
   иначе
        если d[N-1] = = -1 и d[N-2] > 0, то
             d[N-1] = 0;
             d[N-2] = d[N-2]-q;
        конец если
   конец если
   вернуть D
конец
```

Данный алгоритм должен выполняться только потоком, оперирующим старшим разрядом числа. Время выполнения данного алгоритма не зависит от количества разрядов числа. После выполнения данного алгоритма для числа $1\overline{999}$ получим результат $01\overline{99}$ (ведущий ноль приписан для наглядности).

В предположении, что число потоков, способных выполняться одновременно, больше количества разрядов числа, время сложения, сдвига, а также вычисления частичного произведения будет константным. Поскольку эти шаги выполняются для всех разрядов числа, то время выполнения алгоритма умножения составит NT_i, где N — количество разрядов числа; T_i — время выполнения одной итерации цикла. В случае, когда время выполнения итерации цикла константное, время выполнения алгоритма умножения составит $O(N)$. Таким образом, время выполнения алгоритма умножения в данном случае пропорционально количеству разрядов числа.

Сравнение чисел a и b, представленных в знакоразрядной системе счисления, сводится к определению их разности и проверке ее знака. Рассмотрим проверку знака разности. Знак числа, представленного в знакоразрядной системе счисления, совпадает со знаком его старшего ненулевого разряда. Следовательно, для определения знака числа необходимо определить значение его старшего ненулевого разряда. Сделать это можно, используя редукцию массива, содержащего разряды числа, относительно следующей операции:

```
Входные данные: r - значение младшего разряда
   l — значение старшего разряда
начало
   если l = = 0, то
        вернуть r
   иначе
        вернуть l
конец
```

Полученное значение старшего ненулевого разряда разности $a - b$ интерпретируется следующим образом: если оно < 0, то $a < b$; если оно > 0, то $a > b$; если оно равно 0, то $a = b$.

Частный случай сравнения двух чисел — это проверка их равенства. Алгоритм проверки равенства состоит в вычислении разности $a - b$ и проверке полученного результата на равенство 0.

Так как самая трудоемкая операция в этих двух алгоритмах — параллельная редукция, в данном случае время выполнения алгоритма будет пропорционально $\log_2(N)$ (в предположении, что число потоков, способных выполняться одновременно, больше количества разрядов числа).

Реализация вычислений в многомодульной системе счисления

Определение. Если целое число A представлено в виде $A = qm + r$, где q, r — целые числа, m — натуральное число, $0 \leqslant r < m$. Тогда r называется *вычетом* (или *остатком* от деления a на m).

Определение. Целое число *a сравнимо с целым числом b по модулю m*, тогда и только тогда, когда $b - a$ делится на m без остатка. Данное утверждение обозначается как $a \equiv b \pmod m$.

Вычисления в модулярной арифметике могут быть реализованы как в одномодульной, так и в многомодульной системах счисления. Представление чисел в многомодульной системе счисления эквивалентно представлению чисел в одномодульной системе счисления, но число представляется не одним, а n остатками, взятыми по модулям m_1, ..., m_n. Обозначим представление числа a в многомодульной системе как $(a_0, ..., a_1, ..., a_{n-1})$. Если любая пара чисел из модулей m_1, ..., m_n взаимно-простые, то такая многомодульная система эквивалентна одномодульной системе со значением модуля, равным произведению $m_1 ... m_n$ [4, 5].

Основным преимуществом представления числа в многомодульной системе счисления является возможность распараллеливания вычислений между одинаковыми вычислителями. Все вычислители выполняют одни и те же операции, но каждый вычислитель приводит результаты по своему модулю. Это позволяет распараллелить выполнение арифметических операций. Основной недостаток этой системы — высокая сложность преобразования чисел из многомодульной системы счисления в позиционную систему счисления, а также высокая сложность сравнения чисел.

В одномодульной арифметике вычетов любое целое число a отображается в одно из целых чисел в конечном множестве $I_M = \{0, 1, 2, ..., M - 1\}$. Такое отображение обозначается как $|b|_M = r$, при этом $0 \leqslant r < M$ и $b \equiv r \pmod M$. Число r называется наименьшим неотрицательным вычетом числа b по модулю M.

Справедливо следующее утверждение: если a, b, M — целые и $M > 1$, то выполняются следующие равенства:

$$|a + b|_M = \||a|_M + |b|_M|_M = \||a|_M + b|_M = |a + |b|_M|_M$$

$$|ab|_M = \||a|_M \cdot |b|_M|_M = \||a|_M \cdot b|_M = |a \cdot |b|_M|_M.$$

Данные равенства указывают способ выполнения операций сложения и умножения в модулярной арифметике.

Рассмотрим способы вычитания и деления в модулярной арифметике. Для этого введем понятия противоположного и обратного элементов по модулю M.

Элемент $\underline{a}$, противоположный данному элементу a по модулю M, определим как:

$$\underline{a} = |-a|_M = M - a.$$

Тогда операцию вычитания по модулю M определим как сложение с противоположным элементом по модулю M:

$$|a - b|_M = |a + \underline{b}|_M.$$

Алгоритмы сложения, вычитания и умножения чисел, представленных в многомодульной системе, аналогичны алгоритмам сложения, вычитания и умножения чисел, представленных в одномодульной системе. Пусть $a = (a_0, a_1, ..., a_{n-1})$, $b = (b_0, b_1, ..., b_{n-1})$. Результат некоторой арифметической операции обозначим $c = (c_0, c_1, ..., c_{n-1})$. Остатки c_i от деления результатов операций сложения, вычитания и умножения на m_i приведены в табл. 2.

Как видно из табл. 2, при выполнении операций сложения, вычитания и умножения над числами, представленными в многомодульной системе, нет необходимости в обмене данными между потоками.

Проверить равенство двух чисел $a = (a_0, a_1, ..., a_{n-1})$ и $b = (b_0, b_1, ..., b_{n-1})$, представленных в многомодульной системе, достаточно просто: для этого необходимо сравнить соответственно все остатки a_i и b_i, и затем провести редукцию массива результатов относительно операции "логическое или".

Время выполнения алгоритмов сложения, вычитания и умножения в предположении, что число потоков, способных выполняться одновременно больше количества модулей в системе, является константным (не зависит от количества модулей в системе).

Так как самая трудоемкая операция в алгоритме проверки равенства чисел — параллельная редукция,

Таблица 2

Выполнение операций в многомодульной системе

Операция	Результат
$a + b$	$c_i = \|a_i + b_i\|_{m_i}$
$a - b$	$c_i = \|a_i - b_i\|_{m_i}$
ab	$c_i = \|a_i b_i\|_{m_i}$

в данном случае время выполнения алгоритма будет пропорционально $\log_2(N)$ (в предположении, что число потоков, способных выполняться одновременно, больше количества разрядов числа).

Существуют несколько способов определения знака числа, представленного в многомодульной системе счисления. Наиболее очевидный — это преобразование числа в двоичную систему счисления. Однако этот способ обладает достаточно низким быстродействием [5]. Кроме того, он предполагает вычисления с числами изменяемой разрядности, представленными в двоичной системе счисления, что отрицательно сказывается на возможности распараллеливания алгоритма данного преобразования. Это означает, что реализация данного алгоритма на GPU архитектуры CUDA не будет эффективной.

В работе [5] приведен алгоритм, связанный с преобразованием числа в систему счисления со смешанным основанием.

Введем следующие определения. Рассмотрим упорядоченный набор из n натуральных чисел $\rho = [r_1, r_2, ..., r_n]$, компоненты которого назовем *основаниями*. Обозначим $R = \prod_{i=1}^{n} r_i$.

Теорема. Любое целое число s, такое, что $0 \leqslant s < R$, можно единственным образом представить в виде [3]:

$$s = d_0 + d_1(r_1) + d_2(r_1 r_2) + ... + d_{n-1}(r_1 r_2 ... r_{n-1}),$$

где $0 \leqslant d_i < r_{i+1}$, $i = 0, 1, ..., n-1$. Назовем d_i *цифрами стандартного представления для смешанного основания*. Упорядоченный набор цифр $d_0, d_1, ..., d_{n-1}$ назовем *представлением числа s в системе со смешанным основанием* ρ. Данное представление запишем в следующем виде:

$$\langle s \rangle_\rho = \langle d_0, d_1, ..., d_{n-1} \rangle.$$

Стандартной системой со смешанным основанием для ρ назовем множество всех возможных наборов $\langle s \rangle_\rho$ для всех целых чисел $0 \leqslant s < R$. Отметим, что в частном случае, когда $r_1 = r_2 = ... = r_n$ приходим к представлению числа в позиционной системе счисления с фиксированным основанием.

Для определения знака числа следует отметить следующий факт. Поскольку многомодульная система с основаниями $R = m_1, ..., m_n$ эквивалентна одномодульной системе со значением модуля, равным произведению $m_1, ..., m_n$, будем считать, что числа $0 \leqslant s \leqslant \left[\frac{R}{2}\right]$ отображаются в самих себя, а числа $-\left[\frac{R}{2}\right] < s < 0$ отображаются в $R + s$ (т. е. представляют отрицательные числа). Для определения знака числа, представленного в многомодульной системе, необходимо сделать следующее:

1. Перевести число s в стандартное представление для смешанного основания $\rho_m = [m_1, m_2, ..., m_n]$. Таким образом, будет получено $\langle s \rangle_{\rho_m}$.

2. Сравнить число $\langle s \rangle_{\rho_m}$ с числом $\left\langle \left[\frac{R}{2}\right] \right\rangle_{\rho_m}$. В позиционной системе счисления алгоритм сравнения чисел очевиден. Если $\langle s \rangle_{\rho_m} > \left\langle \left[\frac{R}{2}\right] \right\rangle_{\rho_m}$, то число $s < 0$, иначе $s \geqslant 0$.

Пусть число s имеет следующее представление в многомодульной системе:

$$|s|_m = \left(|s|_{m_1}, |s|_{m_2}, ..., |s|_{m_n} \right),$$

а в системе со смешанным основанием ρ_m оно будет следующим:

$$\langle s \rangle_{\rho_m} = \langle d_0, d_1, ..., d_{n-1} \rangle.$$

Тогда $s = d_0 + d_1(m_1) + d_2(m_1 m_2) + ... + d_{n-1} \times$ $\times (m_1 m_2 ... m_{n-1})$. Заметим, что $|s|_{m_1} = d_0$. Таким образом, определен один из элементов представления числа s в многомодульной системе.

Рассмотрим число

$$s - d_0 = d_1(m_1) + d_2(m_1 m_2) + ... + d_{n-1}(m_1 m_2 ... m_{n-1}) = m_1(d_1 + d_2(m_2) + ... + d_{n-1}(m_2 ... m_{n-1})).$$

Очевидно, что $(s - d_0)/m_1 = d_1 + d_2(m_2) + ... + d_{n-1} \times$ $\times (m_2 ... m_{n-1})$, следовательно $d_1 = |(s - d_0)/m_1|_{m_2}$. Значение $(s - d_0)/m_1$ можно вычислить, введя сокращенное основание многомодульной системы счисления $(m_1, ..., m_n)$. Поскольку все модули системы $(m_1, ..., m_n)$ взаимно-простые, то для каждого из них существует число, обратное m_1.

Описанные выше преобразования необходимо повторять до тех пор, пока в сокращенном основании многомодульной системы не останется одна цифра.

Описание на псевдокоде алгоритма преобразования числа из многомодульной системы счисления $(m_1, ..., m_n)$ в систему счисления со смешанным основанием для $\rho_m = [m_1, m_2, ..., m_n]$ приведено ниже.

```
Алгоритм Convert (A, M, N)
Вход: A — число в многомодульной системе
счисления
   M — модули системы
   N — количество модулей системы
Выход: R — число в знакоразрядной системе
счисления
начало
   T — промежуточные результаты
   T = S
   Для i от 1 до N цикл
        R[i] = T[i];
        Для j от i+1 до N цикл
             T[j] = (T[j]-T[i]) % M[j]
             m1 = M[i]-1 (mod M[j]) // обратное
        число по модулю M[j]
             T[j] = (m1*T[j]) % modules[j]
        конец цикла
   конец цикла
   вернуть R
конец
```

Данный алгоритм допускает распараллеливание, поскольку внутренний его цикл можно запустить параллельно:

```
Алгоритм Convert_Parallel (A, M, N, j)
Вход: A — число в многомодульной системе
счисления
   M — модули системы
   N — количество модулей системы (равно ко-
личеству потоков)
   j — индекс данного потока
Выход: R — число в знакоразрядной системе
счисления
начало
      T — промежуточные результаты
      T = S
      Для i от 1 до N цикл
      если j = = i, то
            R[j] = T[j]
      конец если
      если j > i, то
            T[j] = (T[j]-T[i]) % M[j]
            m1 = M[i]-1 (mod M[j]) // обрат-
      ное число по модулю M[j]
            T[j] = (m1*T[j]) % modules[j]
      конец если
   конец цикла
   вернуть R
конец
```

Оценим время выполнения данного алгоритма в предположении, что число потоков, способных выполняться одновременно, больше количества модулей в системе. В этом случае время выполнения итерации цикла не зависит от количества модулей в системе. При этом необходимо сделать N итераций, где N — количество модулей в системе. Таким образом, время выполнения данного алгоритма будет пропорционально количеству модулей в системе, т. е. $O(N)$.

Практическое применение разработанных алгоритмов

Итак, на GPU архитектуры CUDA могут быть достаточно эффективно реализованы операции сложения, вычитания, умножения и сравнения чисел произвольной разрядности.

Оба способа работы с числами произвольной разрядности реализованы в виде библиотек классов для CUDA C++. Для разработанных классов перегружены операции сложения, вычитания, умножения, а также операции сравнения. Это позволяет программисту использовать арифметические выражения языка C++.

Вычисления с использованием разработанных классов могут быть организованы следующим образом. Алгоритм кодируется с использованием естественных выражений языка C++. Каждый блок работает с одним набором входных данных. Для распараллеливания программист может работать только с индексом блока.

Основным недостатком разработанной библиотеки классов является отсутствие обратного преобразования результата в позиционную систему счисления. Это ограничивает возможность применения библиотеки. Однако библиотека позволяет вычислить результат сравнения двух чисел. Поэтому одной из возможных областей применения разработанной библиотеки является вычисление предикатов, т. е. функций, чья область значений имеет конечное число элементов. Вычисление подобных функций проводится в большинстве алгоритмов вычислительной геометрии.

Особенностью как предикатов, так и алгоритмов вычислительной геометрии является их чувствительность к вычислительным ошибкам. Вследствие этого, на выходе алгоритма можем получить результаты, качественно отличающиеся от истинных (пресекающиеся отрезки могут быть интерпретированы как непересекающиеся, выпуклый многоугольник может быть воспринят как невыпуклый). В некоторых случаях возможно даже зацикливание алгоритма [1, 7, 8]. Такие ситуации возникают достаточно редко, и для большинства задач с ними можно смириться, для других задач (в частности, для задач САПР) важно получить достоверный результат вычисления.

Исследование быстродействия разработанных алгоритмов

Приведем таблицу (табл. 3) асимптотических сложностей арифметических операций над числами, представленными в двух рассматриваемых форматах. Оценки даны в предположении, что число потоков, способных выполняться одновременно, больше количества модулей (для многомодульной системы счисления) или количества разрядов числа (для знакоразрядной системы счисления). Количество элементов (соответственно, модулей или разрядов) обозначено как N.

Измерим время выполнения операции умножения чисел, представленных в знакоразрядной системе счисления, и операции определения знака числа, представленного в многомодульной системе счисления. Эти операции являются наиболее трудоемкими. На рис. 4 представлено время выполнения этих операций в зависимости от разрядности чисел.

Таблица 3

Оценки быстродействия параллельных алгоритмов выполнения арифметических операций

Операция	Система счисления	
	Многомодульная	Знакоразрядная
Сложение, вычитание	$O(1)$	$O(1)$
Умножение	$O(1)$	$O(N)$
Проверка равенства	$O(\log(N))$	$O(\log(N))$
Сравнение, определение знака	$O(N)$	$O(\log(N))$

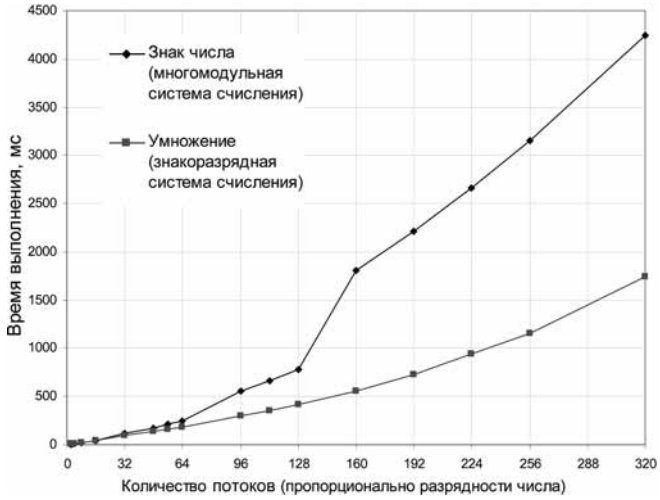


Рис. 4. Зависимость времени выполнения наиболее трудоемких операций от разрядности числа

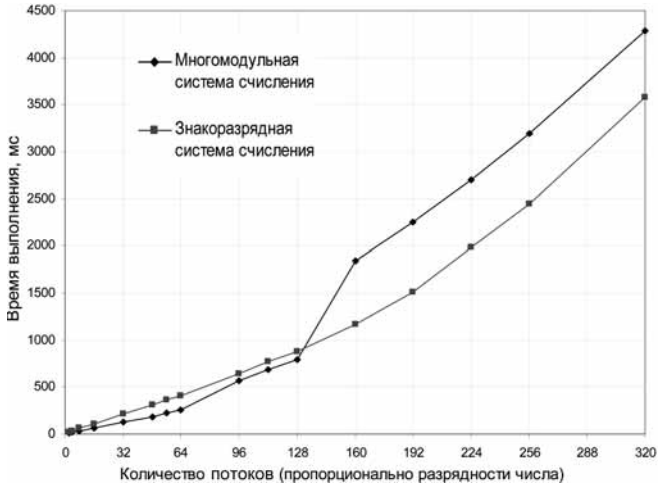


Рис. 5. Зависимость времени решения тестовой задачи от разрядности числа

Сравним время вычисления тестового выражения при использовании различных систем счисления. В качестве тестовой задачи выбрана задача определения ориентации троек точек (A, B, C):

$$\text{orientation}(A, B, C) = \\ = \text{sgn}((x_b - x_a)(y_c - y_a) - (y_b - y_a)(x_c - x_a)).$$

Зависимость времени решения задачи (в мс) от разрядности используемых чисел представлена на рис. 5. Отметим, что чем шире диапазон представимых чисел, тем больше потоков используется для работы с одним числом.

Измерения проводились на ЭВМ со следующей конфигурацией:

- процессор Intel CORE2 DUO T9600, тактовая частота 2,8 ГГц;
- 2 ГБ ОЗУ;
- графический процессор Quadro FX 2700M (48 потоковых процессоров, 512 МБ ОЗУ);
- операционная система OpenSUSE Linux 11.2.

Из полученных графиков можно сделать следующие выводы:

- использование чисел, представленных в многомодульной системе счисления позволяет обеспечить большее быстродействие, чем использование чисел, представленных в знакоразрядной системе счисления в случае, если вычисляемое выражение содержит большое количество операций умножения. Если необходим только знак выражения, то при увеличении количества арифметических операций в выражении эффективность многомодульной системы счисления будет возрастать;
- использование чисел, представленных в многомодульной системе счисления, позволяет обеспечить большее быстродействие при меньшей разрядности числа;
- необходимо совершенствовать реализацию алгоритма определения знака числа, представленного в многомодульной системе счисления, и алгоритма умножения чисел, представленных в знакоразрядной системе счисления.

Заключение

Разработаны и реализованы алгоритмы выполнения арифметических операций над числами большой разрядности для GPU архитектуры CUDA. Разработанные программные средства являются библиотекой классов. Предложен способ организации вычислений с использованием разработанной библиотеки.

Показана область эффективного применения разработанных программных средств — вычисление предикатов вычислительной геометрии в задачах САПР.

Список литературы

1. **Kettner L., Mehlhorn K., Pion S., Schirra S., and Yap C.** Classroom Examples of Robustness Problems in Geometric Computations // ESA, LNCS. 2004. Vol. 3221. P. 702—713. URL: http://www.mpi-inf.mpg.de/~kettner/pub/nonrobust_cgta_06.pdf

2. **Боресков А. В., Харламов А. А.** Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. 232 с.

3. **NVIDIA CUDA** Programming Guide Version 3.0. Nvidia Corporation, 2010.

4. **Поспелов Д. А.** Арифметические основы вычислительных машин дискретного действия. М.: Высшая школа, 1970.

5. **Грегори Р., Кришнамурти Е.** Безошибочные вычисления. Методы и приложения: пер. с англ. М.: Мир, 1988. 208 с.

6. **Дзегеленок И. И., Оцоков Ш. А.** Алгебраизация числовых представлений в обеспечении высокоточных суперкомпьютерных вычислений // Вестник МЭИ. 2010. № 3. С. 107—116.

7. **Орлов Д. А.** Организация многопотоковой обработки данных с исключением аномалий при решении задач вычислительной геометрии // Автореферат диссертации на соискание ученой степени кандидата технических наук. М.: Полиграфический центр МЭИ (ТУ), 2010.

8. **Орлов Д. А., Дзегеленок И. И., Харитонов В. Ю.** Вычислительные аспекты построения распределенных систем виртуальной реальности // Вестник Московского энергетического института. 2008. № 5. С. 27—32.

УДК 004.41, 004.02, 372.8

А. С. Хританков, канд. физ.-мат. наук, доц.,
Московский физико-технический институт,
e-mail: anton.khritankov@acm.org.

Опыт обучения методам проектирования программных систем

Рассказано об опыте преподавания университетского курса по проектированию программных систем на факультете ИВТ МФТИ. На данный момент курс был прочитан шесть раз, суммарная аудитория составила более сотни студентов. В данной работе изложены основные подходы к построению курса, обсуждаются соответствие программы курса рекомендациям IEEE и ACM по составу программ обучения в области разработки программного обеспечения (SEEK), организация задания студентам, используемые инструменты и достигнутые результаты.

***Ключевые слова**: проектирование программных систем, университетское образование*

Введение

Целью курса "Проектирование программных систем" является ознакомление студентов с дисциплиной проектирования программного обеспечения, в том числе с основополагающими концепциями, историей развития дисциплины и современными методами анализа и проектирования.

Курс разработан автором и входит в обязательную программу обучения студентов факультета ИВТ МФТИ. Учебная нагрузка включает по 34 академических часа лекций и семинаров, а также время на самостоятельную работу студентов.

Близкие по тематике курсы читаются, например в МГУ [1], в ВШЭ [2, 3] и обычно называются "методы объектно-ориентированного проектирования", "моделирование программных систем".

При разработке курса предполагалось совместно рассматривать моделирование систем как средство снижения сложности их создания и методы проектирования для построения моделей этих систем. Поэтому изложение тем по моделированию и проектированию было объединено в рамках одного семестрового курса. Возможно, при этом несколько снижается детальность изложения, но получается выигрыш в изначальном создании у студентов взгляда на программную инженерию как на строгую дисциплину.

В рамках курса рассказывается о современных концепциях и методах проектирования ПО, студенты обучаются использовать UML [4] при объектно-ориентированном проектировании. В процессе обучения студенты реализуют и защищают командный учебный проект, применяя полученные знания, а также овладевают навыками совместной работы: коммуникаций, организации и планирования. В конце курса для индивидуальной оценки знаний предусмотрена контрольная работа.

По итогам обучения студенты способны строить модели с использованием UML и проектировать простые системы.

Структура и организация курса

Программа курса [5] включает следующие основные разделы: введение в программную инженерию, моделирование с помощью UML, методы структурного и объектно-ориентированного анализа и проектирования. Предполагается, что к началу изучения данного курса студенты уже имеют опыт программирования,

владеют техническим английским языком, освоили курс по алгоритмам и структурам данных.

Во введении курса рассказывается о характеристиках качества ПО, организации разработки, процессах и моделях жизненного цикла.

При изложении UML важным является создание целостного представления о языке как средстве представления моделей, а не формальной нотации для построения диаграмм. Значительная доля времени уделяется семантике заложенных в UML распространенных методов моделирования: схем состояний [6], схем последовательности сообщений [7], достижений в области объектно-ориентированного моделирования "трех гуру" [8], технологий архитектурного моделирования (ADL).

Так как тема анализа требований не является главной, по ней излагаются только основы и базовые методы, необходимые при разработке моделей предметной области, такие как методы выделения классов и моделирования поведения с помощью карточек CRC [9].

Раздел по структурному проектированию является важной частью курса. В первую очередь, в нем уделяется внимание таким принципам как модульность, абстракция, сокрытие информации [10], постепенное уточнение [11] (*stepwise refinement*) и критериям качества дизайна.

Соответствие программы курса разделам SEEK

Раздел SEEK	Название раздела (пер. с англ.)	Число тем в разделе	
		Всего	Отражены в курсе
Область знаний: Проектирование			
DES.con	Основные понятия	8	8
DES.str	Стратегии проектирования	2	2
DES.ar	Архитектура	6	2
DES.hci	Пользовательский интерфейс	7	0
DES.dd	Детальное проектирование	5	3
DES.ste	Инструменты проектирования	3	3
Область знаний: Моделирование и анализ (показаны не все разделы)			
MAA.md	Основы моделирования	6	5
MAA.tm	Типы моделей	5	5
MAA.af	Основы анализа	6	4
Область знаний: Качество ПО (показаны не все разделы)			
QUA.cc	Понятие о качестве	7	3
Область знаний: Управление разработкой ПО (показаны не все разделы)			
MGT.con	Понятие об управлении	5	4

При изложении тем по объектно-ориентированному проектированию рассматриваются два основных подхода: абстрактные типы данных (ADT) [12] и подход на основе обязанностей [13] (*Responsibility-Driven Design*). В программу также входит изучение принципов объектно-ориентированного проектирования, эвристик GRASP [14] и паттернов проектирования на уровне классической книги Gang-of-Four [15].

Результаты сравнения программы курса с рекомендациями IEEE и ACM по составу программ обучения в области разработки программного обеспечения (SEEK) [16] приведены в таблице. Программа охватывает все разделы по проектированию (DES), кроме взаимодействия с пользователем, а также часть разделов по моделированию (MAA), качеству программного обеспечения (QUA) и управлению проектами (MGT).

Курс достаточно плотный и рассчитан на 68 аудиторных часов и 34 часа самостоятельной работы. Согласно рекомендациям SEEK, охваченным разделам следует уделять не менее 75 академических часов.

Учебный проект

Суть учебного проекта состоит в разработке модели системы согласно краткому описанию. Описания проектов прорабатываются заранее для достижения баланса сложности, интереса и охвата тем курса. По согласованию студенты могут предлагать свои проекты.

Проект включает задание по построению модели анализа и задание по детальному проектированию согласно построенной модели. На рис. 1 приведен пример описания проекта [5].

В первом задании студенты на основе текстового описания создают модель вариантов использования и модель предметной области.

До начала выполнения второго задания студентам необходимо согласовать с преподавателем уровень детализации, интерфейсы используемых библиотек и

Программа управляет лифтом многоэтажного дома. Кроме стандартных кнопок выбора этажа, нужны кнопки вызова диспетчера и управления дверьми, датчик задымления. ПО должно работать с разными лифтами, домами и датчиками.

В целях экономии ресурса ламп свет в лифте должен выключаться, если лифт не используется. Необходимо отслеживать перегрузку лифта, задымление и препятствия закрытию дверей. В случае задымления должен быть отправлен сигнал диспетчеру.

Лифт управляется пользователями с кнопок на панели внутри кабины лифта и кнопок вызова на этажах. Программа должна обслуживать лифты в различных конфигурациях: с табло индикации этажа и направлением движения лифта, с обычной кнопкой вызова и с кнопкой вызова с указанием направления. Соответственно программа должна включать несколько алгоритмов управления движением кабины в зависимости от конфигурации.

Диспетчер может отключить или включить лифт удаленно, связаться с пассажирами, а также получить текущее состояние лифта - положение кабины и дверей, суммарный вес пассажиров в кабине.

Рис. 1. Описание проекта "Универсальный лифт"

каркасов разработки (*framework*). Студентам предоставляются заготовки интерфейсов, которые они могут адаптировать под проект.

Второе задание состоит в разработке детального проекта системы, реализующего модель анализа, построенную в первом задании, с использованием согласованных интерфейсов библиотек и каркасов разработки. Отчет по выполненному проекту включает описание предметной области и вариантов использования, динамическую и согласованную с ней структурную модели системы, а также обоснования принятых проектных решений и промежуточные артефакты.

Отчет предоставляется заранее. Для проверки описанных в нем решений используются контрольные списки. Проверяется согласованность частей модели, правильность применения методов и соответствие постановке задачи.

Защита проекта происходит на семинаре, студенты представляют к обсуждению результаты в виде презентации.

Инструменты моделирования

На основе сравнительного анализа нескольких инструментов моделирования с помощью UML выбор был остановлен на средстве MagicDraw [17]. Данный инструмент используется для проведения занятий по курсу в рамках академической программы [18] компании No Magic Inc. Инструмент обеспечивает наиболее полную поддержку UML и высокую эффективность разработки моделей.

Среди недостатков для курса следует отметить специализацию инструмента только на UML. Например, моделирование при помощи CRC или структурные графики (*structure charts*) не поддерживаются.

Общая оценка курса

На момент написания данной статьи курс был прочитан шесть раз. На основе анализа и отзывов студентов были выявлены следующие особенности.

Субъективно, курс по проектированию лучше усваивается студентами, которые не обладают блестящими навыками программирования. Оказывается, что при проектировании отличное знание C++ вызывает неявные аналогии с кодированием и мешает рассуждать абстрактно. Классы воспринимаются как участки кода или наборы методов. Данная гипотеза подтверждается тем, что задачи, решаемые архитектором, проектировщиком и разработчиком, различны и требуют разных подходов [19].

Курс читается в бакалавриате и магистратуре. Как и можно было ожидать, старшекурсникам проще работать с абстрактными моделями. Из практики также можно сделать вывод, что по предложенной программе курс можно успешно освоить, занимаясь прилежно и регулярно.

К текущим недостаткам, по отзывам студентов, можно отнести достаточно сложный переход от заданий на семинарах к самостоятельной работе над проектом. Не вызывает сомнений, что методы обучения и стиль изложения будут развиваться и дорабатываться. Например, для учебных целей была разработана систематическая процедура объектно-ориентированного проектирования на основе процесса ICONIX [20].

Об одном свойстве оценок студентов

При анализе результатов за 2009 и 2010 гг. было замечено разделение оценок за семестровую контрольную работу на две области: со средними, близкими друг другу оценками, и с более высокими, более отдаленными друг от друга. Учитывая расположение областей, граница между ними была названа *точкой инсайта (insight)*.

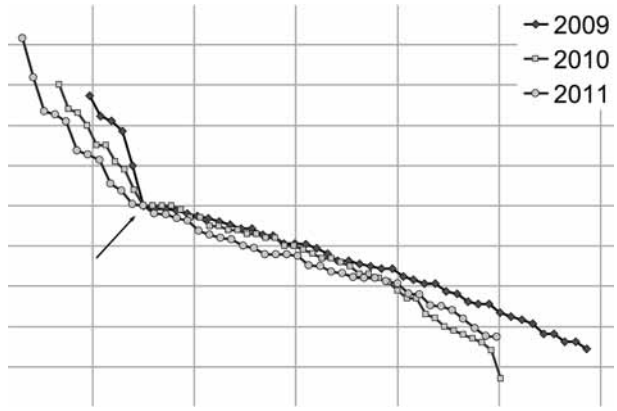


Рис. 2. Расположение точки инсайта (указана стрелкой) в оценках студентов за семестровую контрольную работу (по вертикальной оси) по годам, оценки студентов упорядочены по убыванию слева направо (по горизонтальной оси)

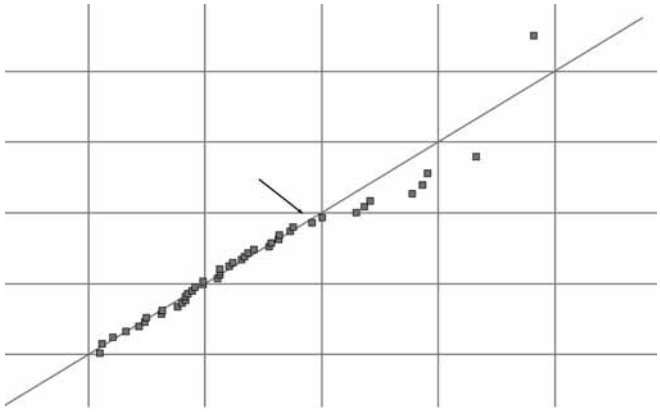


Рис. 3. График квантилей (*QQ*-график) оценок за семестровую контрольную работу за 2011 г. (по вертикальной оси) с указанием расположения точки инсайта (указана стрелкой) по отношению к бета-распределению (по горизонтальной оси)

Упорядочим оценки студентов по убыванию, приведем количество оценок к интервалу [0, 1], применим линейное преобразование такое, чтобы совместить точки инсайта для разных лет. Полученные графики показаны на рис. 2, оценки студентов убывают слева направо. На графиках точка инсайта указана стрелкой. Для 2011 г. визуально разделить области сложно, поэтому потребуется уточнить определение точки инсайта.

Пусть оценка является реализацией некоторой случайной величины. Из графика квантилей, приведенного на рис. 3, видно, что распределение оценок за 2011 г. напоминает бета-распределение

По смыслу, *точку инсайта* разумно определить как оценку, в которой вторая производная функции распределения достигает наименьшего значения. На рис. 3 она отмечена стрелкой. При этом формальное и графическое определения точки инсайта совпадают для 2009 и 2010 гг.

Заключение

В данной работе рассмотрены основные положения курса по методам проектирования программных систем. Программа курса в целом соответствует рекомендациям IEEE и ACM, но включает дополнительные разделы по смежным областям. Важной частью курса является применение студентами полученных знаний и навыков при выполнении командного учебного проекта.

Критически рассмотрев результаты курса, можно сказать, что еще не достигнут уровень преподавания, когда студенты по завершении курса полностью понимают, как правильно применять полученные знания. В этом направлении курс требует дальнейшего развития. В качестве одного из шагов планируется увеличивать долю рассматриваемых практических примеров по проектированию, в том числе из индустриальной практики.

Наличие точки инсайта является интересным свойством оценок студентов. В дальнейшем планируется продолжить исследования данного свойства и его возможных применений.

Список литературы

1. **Курс** "Объектно-ориентированный анализ и проектирование", URL: http://sp.cs.msu.su/courses/ooap/
2. **Курс** "Проектирование архитектуры программных систем", URL: http://www.hse.ru/edu/courses/23108664.html
3. **Курс** "Моделирование и анализ программного обеспечения", URL: http://www.hse.ru/edu/courses/23112343.html
4. **Unified** Modeling Language, URL: http://www.uml.org/
5. **Курс** "Проектирование программных систем", страничка курса на 2011 г., URL: http://pps-fall2011.dyndns.org
6. **Harel D.** Statecharts: A Visual Formalism for Complex Systems // Science of Computer Programming. 1987. N 8. P. 231—274.
7. **ITU-T** Recommendation Z.120. Message Sequence Charts (MSC). URL: http://www.itu.int/ITU-T/2005-2008/com17/languages/Z120.pdf
8. **Буч Г., Якобсон А., Рамбо Дж.** UML. Классика CS. 2-е изд. / Пер. с англ. Под общей редакцией проф. С. Орлова. СПб.: Питер, 2006. 736 с.
9. **Beck K., Cunningham W.** A laboratory for teaching object oriented thinking // ACM SIGPLAN Notices. (New York, NY, USA: ACM). 1989. 24 (10). P. 1—6.
10. **Parnas D. L.** On the criteria to be used in decomposing system into modules // Communications of the ACM. 1972. Vol. 15. I. 12. P. 1053—1058.
11. **Wirth N.** Program development by stepwise refinement // Communications of the ACM. 1971. Vol. 14. N 4. P. 221—227.
12. **Мейер Б.** Объектно-ориентирование конструирование программных систем. / Пер. с англ.: В. Биллиг, М. Дехтерь, Н. Супонев, Е. Павлова, под ред. В. Биллига. М.: Русская редакция, 2005. 1204 с.
13. **Wirfs-Brock R., Wilkerson B.** Object-oriented design: a responsibility-driven approach // In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications. OOPSLA '89. New Orleans, Louisiana, United States. October 02—06 1989. ACM Press, New York, 1989. P. 71—75.
14. **Larman C.** Applying UML and Patterns — An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.). New Jersey: Prentice Hall, 2005.
15. **Гамма Э., Хелм Р., Джонсон М.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. С-Пб.: Питер, 2007. 366 с.
16. **Software** Engineering 2004. Curriculum guidelines for undergraduate degree programs in software engineering, URL: http://sites.computer.org/ccse/
17. **No Magic Inc.**, MagicDraw. URL: http://www.magicdraw.com
18. **Академическая** программа No Magic. URL: http://www.magicdraw.com/md_academic_program
19. **Shuja A. K., Krebs J.** IBM Rational Unified Process Reference and Certification Guide: Solution Designer (RUP). IBM Press, 2008. 336 p.
20. **Rosenberg D., Stephens M., Collins-Cope M.** Agile Development with ICONIX Process. NY: Apress, 2005. 261 p.

ИНФОРМАЦИЯ

10th IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS 2012)

Харьков, Украина, 14—17 сентября 2012 г.

Цель симпозиума **IEEE East-West Design & Test Symposium** (EWDTS) - расширение международного сотрудничества и обмен опытом между ведущими учеными Западной и Восточной Европы, Северной Америки и других стран в области автоматизации проектирования, тестирования и верификации электронных компонентов и систем.

Официальный сайт URL: ***http://www.ewdtest.com/conf/***

CONTENTS

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

Дизайнер *Т.Н. Погорелова*. Технический редактор *Е.М. Патрушева*. Корректор *М.Г. Джавадян*

Сдано в набор 12.04.2012 г. Подписано в печать 28.05.2012 г. Формат 60×88 1/8.
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.