ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

4(176) 2011

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с ноября 1995 г.

УЧРЕДИТЕЛЬ
Издательство "Новые технологии"

СОДЕРЖАНИЕ

Информация о журнале доступна по сети Internet по адресу http://www.**informika.ru/text/magaz/it/** или **http://novtex.ru/IT.**

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 004.942

М. С. Талалай[1,2], инженер-исследователь, аспирант,
К. В. Трушин[1,2], инженер-практикант, аспирант,
О. В. Венгер[2], инженер-исследователь,
[1] Московский физико-технический институт
(государственный университет)
[2] ЗАО "ИНТЕЛ А/О"
e-mail:mikhail.s.talalay@intel.com;
konstantin.truchin@intel.com,
oleg.v.venger@intel.com

Логический синтез комбинационных схем на основе транзисторных шаблонов с регулярной топологией

Рассматривается метод логического синтеза, предназначенный для проектирования интегральных схем с использованием современных технологических процессов, требующих регулярной топологии на базовых слоях. Предлагаемый подход использует транзисторные шаблоны в качестве минимальных функциональных элементов для логического синтеза. Описывается функциональность шаблонов и демонстрируются результаты синтеза на примерах. Эксперименты показывают, что предложенный стиль проектирования регулярных схем потенциально способен сократить площадь по сравнению с синтезом на стандартных ячейках.

Ключевые слова: *регулярная топология, регулярная заготовка, логический синтез, транзисторный шаблон*

Введение

Использование современных технологий производства интегральных схем (ИС) накладывает ряд дополнительных требований на процесс проектирования и производства. Одним из требований, обеспечивающих процесс литографии, является повышение регулярности фотошаблонов, т. е. наличие периодичности в топологии. К современным подходам к проектированию ИС, нацеленным на повышение регулярности, относятся: проектирование на основе регулярных топологических заготовок (Regular Fabrics), проектирование структурных ИС (structural ASICs). Подход на основе регулярных топологических заготовок включает, в том числе, использование стандартных логических матриц (Gate Array) и логических матриц, конфигурируемых межслойными переходами (VPGA).

Методология, основанная на топологических заготовках, представляется наиболее привлекательной с точки зрения регулярности и эксплуатационных характеристик [1, 2]. Базовые слои топологии изготавливаются заранее и имеют чрезвычайно регулярную структуру. Заготовка является универсальной, а проектировщик использует лишь верхние слои топологии для реализации нужной функциональности. Заготовка может включать сильнолегированные области p^+- или n^+-типа, равномерно расположенные затворы транзисторов и межслойные переходы [3]. Иногда фиксируют также и некоторые слои металла [4, 5], вводя строго регулярное расположение полигонов в них.

Критическим слоем на кристалле ИС является слой поликремния, на котором размещаются затворы транзисторов. Именно на этом слое наиболее трудно выполнить правила проектирования. Для современных субмикронных технологических процессов нормой стало размещение одинаковых сегментов поликремния с постоянным шагом между ними (рис. 1). При этом невозможно использовать абсолютно все сегменты для реализации затворов транзисторов. Неиспользуемые сегменты — нефункциональные затворы — являются накладным расходом площади кристалла.

В статье рассматривается новый подход синтеза ИС на регулярной заготовке, минимизирующий число нефункциональных и функциональных за-

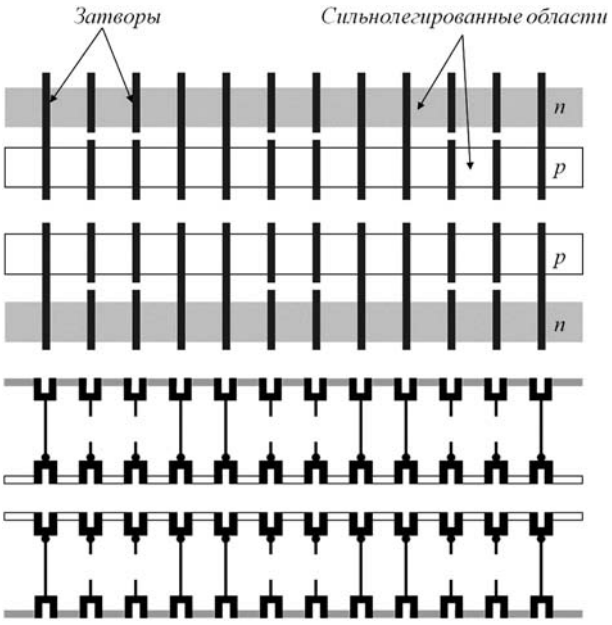


Рис. 1. Общий вид заготовки

творов. Отличие описываемого подхода от других заключается в том, что он использует в качестве минимальных функциональных элементов транзисторные шаблоны — промежуточные структуры между стандартной ячейкой и отдельным транзистором. Выбор шаблона в качестве минимального функционального элемента для логического синтеза дает возможность избежать неоптимального использования затворов. Также появляется возможность состыковать (конкатенировать) различные шаблоны, используя разделяемые сильнолегированные области одного типа, что позволяет уменьшить число изоляционных затворов.

В статье описывается способ получения множества логических функций, реализации которых на транзисторных шаблонах предусматривают размещение на кристалле, построенном на регулярной заготовке, с использованием минимального числа затворов. Чтобы описать логическую функциональность шаблона, предлагается модель транзистора на уровне переключений. Указанные преимущества демонстрируются на примерах.

Регулярная топология и основные определения

В качестве регулярной заготовки рассматривается структура с выровненными полосами сильнолегированных областей p^+- или n^+-типа фиксированной ширины (ширина может различаться для p^+- и n^+-частей) и с периодически меняющимися удлиненными и укороченными затворами. Длинный затвор одновременно управляет p- и n-транзисторами, в то время как короткий затвор управляет только одним транзистором. Получающиеся таким образом последовательно соединенные транзисторы формируют ряды; каждый ряд имеет инвертированный порядок p^+- и n^+-полос по отношению к соседнему ряду, чтобы использовать общий провод питания для двух соседних полос одного типа (рис. 1).

Транзисторный блок — это часть регулярной топологии, которая расположена в одном ряду и включает одинаковое число p- и n-транзисторов. *Размером блока* называется число пар p- и n-транзисторов (рис. 2).

Транзисторный шаблон — это транзисторный блок с внутренней трассировкой и с отмеченными выходами и входами. Соединение транзисторов в шаблоне получается либо посредством разделяемых участков сильнолегированных областей (для смежных транзисторов одного типа), либо посредством проводников на слое металла (в остальных случаях). В получившейся схеме отмечаются входные точки. Аналогично выделяются выходы схемы. Один и тот же блок может индуцировать несколько шаблонов из-за наличия различных способов построения внутренней трассировки и выбора входных/выходных точек. *Физический интерфейс шаблона* — это сильнолегированные области, расположенные в углах прямоугольника, занимаемого шаблоном. На рис. 3 приведен пример шаблона с восемью входами и одним выходом.

Рассмотрим набор независимых логических переменных $x_1, x_2, ..., x_n$. Обозначим все логические функции, которые зависят существенно не более чем от l из этих переменных, как $F(l, x_1, x_2, ..., x_n) = \{f_1, f_2, ..., f_k\}$. Зафиксируем значения напряжений, которые интерпретируются логической единицей и логическим нулем в схеме. Обозначим их как *vcc* и *vss* соответственно. Будем говорить, что к выбранной точке транзисторной схемы *подключена* функция $f \in F(l, x_1, x_2, ..., x_n)$, если напряжение, равное *vcc*, появляется в данной точке при $f = 1$, и *vss* при $f = 0$. Константные функции 0 и 1 будем обозначать соответственно *vss* и *vcc*. Для шаблона T с p входами выберем функции $i_1, i_2, ..., i_p \in F(l, x_1, x_2, ..., x_n)$ таким образом, чтобы на выходе T реализовалась логическая функция *out*. Тогда данное подключение является *логической реализацией функции out* на шаблоне T (рис. 4). Отметим, что для одной и той же функции *out* может существовать несколько логических реализаций на шаблоне T. Все функции, которые можно реализовать на выходе шаблона T при подключении функций из множества $F(l, x_1, x_2, ..., x_n)$, образуют (l, n) — *логическое пространство шаблона*. Стоит

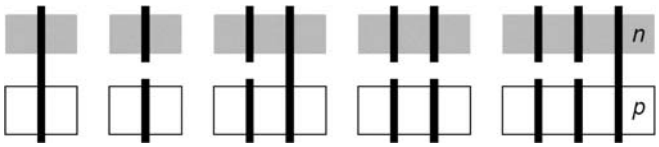


Рис. 2. Пример транзисторных блоков

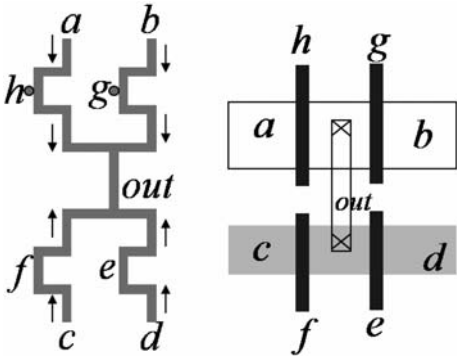


Рис. 3. Шаблон и транзисторная схема:
a, *b*, *c*, *d*, *e*, *f*, *g*, *h* — входы, *out* — выход; *a*, *b*, *c*, *d* формируют физический интерфейс

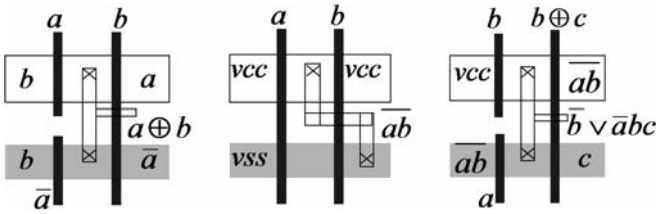


Рис. 4. Примеры логической реализации

отметить, что шаблон, в отличие от стандартных ячеек, в общем случае не реализует логическую функцию на выходе, зависящую от входов.

Изоляционный затвор — это всегда закрытый затвор транзистора, который постоянно подсоединен к соответствующему напряжению. *Нефункциональный* (НФ) затвор — это изоляционный затвор или затвор, который не подсоединен к слоям трассировки. Затвор транзистора будем называть *функциональным* (Ф) затвором, если во время работы схемы к нему в разные моменты времени подводятся напряжения обоих типов ("земля" и питание), и это приводит к переходу транзистора из закрытого состояния в открытое, или наоборот.

Транзисторная модель на уровне переключений

В этом разделе описывается способ определения логической функции на выходе данной транзисторной схемы с подключенными логическими функциями на входах. Подобный математический аппарат необходим для описания логического пространства шаблонов.

Структура состояния. Рассмотрим транзисторную схему, состоящую из транзисторов *p*- и *n*-типа и соединяющих их проводников. Также в схеме должны быть отмечены входы и выходы. Напряжения на входах схемы определяют состояние каждого транзистора схемы (открыт или закрыт) и напряжения на всех проводниках. Любой проводник может находиться в одном из следующих состояний:

vcc — означает, что проводник подсоединен к цепи питания напрямую или через цепочку открытых транзисторов *p*-типа;

vss — означает, что данный проводник подсоединен к цепи "земля" напрямую или через цепочку открытых *n*-транзисторов;

z (состояние высокого импеданса) — предписывается проводнику, который не имеет соединения через цепочку открытых транзисторов ни с цепью питания, ни с цепью "земля";

lowvcc — состояние на проводнике, которое возникает, если этот проводник подсоединен к цепи питания только через открытые *n*-транзисторы (на данном проводнике будет индуцировано напряжение, отличающееся от напряжения питания на некоторую значимую величину);

lowvss — состояние на проводнике, который подсоединен к цепи "земля" только через открытые *p*-транзисторы (напряжение на таком проводнике будет отличаться на значимую величину от нуля).

Только состояния *vcc* и *vss* интерпретируются как логические 1 и 0 соответственно.

Рассмотрим модель транзисторной схемы, в которой введено понятие направления распространения состояний. В действительности данным свойством элементы транзисторной схемы (*p*-транзисторы, *n*-транзисторы и объединение проводников) не обладают. Но схема пересчета состояний, которая описывается ниже, может быть обобщена на случай неориентированных элементов. Для каждого ориентированного элемента определяется таблица состояний (рис. 5, 6), описывающая состояние на выходе (сток для транзистора, объединенный проводник для соединения проводников) для различных пар состояний на входах базового элемента (исток и затвор — для транзистора, пара объединяемых проводов — при соединении).

Исток	Затвор	Сток
vss	*vss*	*lowvss*
vss	*vcc*	*z*
vss	*lowvss*	*bad*
vcc	*vss*	*vcc*
lowvcc	*vss*	*lowvcc*
…	…	…

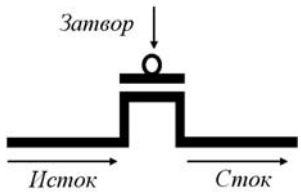


Рис. 5. Часть таблицы истинности для *p*-транзистора

Левый	Правый	Объединение
vss	*vss*	*vss*
lowvss	*vss*	*vss*
lowvss	*lowvcc*	*bad*
lowvcc	*vss*	*bad*
lowvcc	*vcc*	*vcc*
lowvcc	*lowvcc*	*lowvcc*
z	*z*	*z*
…	…	…

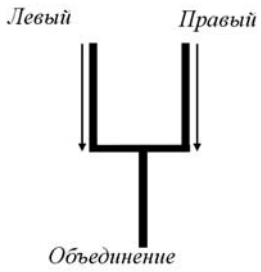


Рис. 6. Часть таблицы истинности для соединения проводов

Введено специальное состояние *bad*, чтобы подчеркнуть особые, запрещенные пары состояний на входах базовых элементов. Модель использует это состояние, чтобы запретить две ситуации:

- использование промежуточных значений напряжения на затворе (состояние *lowvcc* или *lowvss*), так как неясно, каким образом поведет себя транзистор в такой ситуации;
- замыкание, которое означает существование открытой цепи между питанием и "землей" (состояния *vcc* и *vss*).

Характеристические функции, операторы и сигналы. Подсоединим логические функции, зависящие от некоторого фиксированного набора переменных (необязательно существенно), к входам фиксированной транзисторной структуры. Из определения процедуры подсоединения функции к входу схемы следует, что каждая такая логическая функция является характеристической функцией *vcc*-состояния на входе. Тогда отрицание такой функции является характеристической функцией *vss*-состояния. Для каждого провода вводятся характеристические функции каждого состояния. Пятерку характеристических функций на фиксированном проводе будем называть *сигналом*.

Используя таблицы состояний и понятие сигнала, мы вводим набор операторов. Обозначим множество характеристических функций введенных состояний (сигнал), как

$$signal = \{\alpha = (\alpha_1 = vcc, \alpha_2 = vss, \alpha_3 = z,$$
$$\alpha_4 = lowvcc, \alpha_5 = lowvss) \in (\mathbb{B}^n \to \mathbb{B})^5 | (\alpha_i \wedge \alpha_j = 0,$$
$$\forall i \neq j) \ \& \ (\bigvee_{i=1}^{5} \alpha_i = 1)\}, \ \mathbb{B} = \{0, 1\}.$$

Введем обозначения. Пусть *source*, *gate*, *drain* $\in$ $\in$ *signal* и обозначают сигналы на истоке, затворе и стоке транзистора. Пусть *left*, *right* $\in$ *signal* и являются сигналами на объединяемых проводах, а *union* — сигнал на проводе объединения.

Оператор — это логическое отображение пары сигналов в сигнал и в функцию — условие возникновения *bad*-состояния:

$$operator = \{signal \times signal \to signal \times bad_{cond}\}.$$

Определение оператора для *n*-транзистора:

$$operatorN = \{source \times gate \to drain \times bad_{cond}\}.$$

Здесь *sourve*, *gate* и *drain* обозначают сигналы для истока, затвора и стока *n*-транзистора.

$$bad_{cond} = z_{gate} \vee lowvcc_{gate} \vee lowvss_{gate};$$
$$vcc_{drain} = 0;$$
$$vss_{drain} = vcc_{gate} \wedge vss_{source};$$
$$z_{drain} = vss_{gate} \vee (vcc_{gate} \wedge z_{source});$$
$$lowvcc_{drain} = vcc_{gate} \wedge (vcc_{source} \vee lowvcc_{source});$$
$$lowvss_{drain} = vcc_{gate} \wedge lowvss_{source}.$$

Здесь bad_{cond} — условие возникновения запрещенной ситуации. Запись $state_{net}$ обозначает характеристическую функцию состояния *state* на проводнике *net*. Аналогично введены формулы, описывающие логическую функциональность *p*-транзистора (оператор *P*) и объединения проводников (оператор *C*).

Оператор P:

$$bad_{cond} = z_{gate} \vee lowvcc_{gate} \vee lowvss_{gate};$$
$$vcc_{drain} = vss_{gate} \wedge vcc_{source};$$
$$vss_{drain} = 0;$$
$$z_{drain} = vcc_{gate} \vee (vss_{gate} \wedge z_{source});$$
$$lowvcc_{drain} = vss_{gate} \wedge lowvcc_{source};$$
$$lowvss_{drain} = vss_{gate} \wedge (vss_{source} \vee lowvss_{source}).$$

Оператор C:

$$bad_{cond} = (vcc_{left} \vee lowvcc_{left}) \wedge$$
$$\wedge (vss_{right} \vee lowvss_{right}) \vee (vss_{left} \vee lowvss_{left}) \wedge$$
$$\wedge (vcc_{right} \vee lowvcc_{right});$$
$$vcc_{union} = vcc_{right} \wedge (vcc_{left} \vee z_{left} \vee lowvcc_{left}) \vee$$
$$\vee vcc_{left} \wedge (z_{right} \vee lowvcc_{right});$$
$$vss_{union} = vss_{right} \wedge (vss_{left} \vee z_{left} \vee lowvss_{left}) \vee$$
$$\vee vss_{left} \wedge (z_{right} \vee lowvss_{right});$$
$$lowvcc_{union} = lowvcc_{right} \wedge (z_{left} \vee lowvcc_{left}) \vee$$
$$\vee (z_{right} \wedge lowvcc_{left});$$
$$lowvss_{union} = lowvss_{right} \wedge (z_{left} \vee lowvss_{left}) \vee$$
$$\vee (z_{right} \wedge lowvss_{left});$$

Данные формулы могут быть настроены на нужную логическую интерпретацию напряжений, некоторые пары состояний могут быть разрешены или, наоборот, другие комбинации запрещены. В модель, очевидно, также можно добавить новые значения напряжений в качестве новых состояний.

Синтез на транзисторных шаблонах

Для небольшого числа транзисторов (от двух до восьми) были рассмотрены все варианты транзисторных шаблонов с разной внутренней трассировкой. Были изучены свойства логических пространств таких транзисторных шаблонов. Также проведено сравнение с результатами синтеза на основе стандартных ячеек.

Функциональность транзисторных шаблонов. Функциональность транзисторного шаблона можно оценить мощностью множества логических функций, которые принадлежат логическому пространству шаблона.

Зафиксируем логические переменные $X = \{x_1, x_2, ..., x_n\}$, $F(k, x_1, ..., x_l)$ — множество функций, существенно зависящих не более чем от *k* переменных из перечисленных *l* переменных (т. е. $k \leqslant l$), $O(k, x_1, ..., x_l)$ — множество функций, зависящих существенно от *k* переменных. Для каждой функции $out \in Out(2, x_1, x_2)$ найдены логические реализации на шаблонах размера 2 (четыре транзистора) при подключении к входам функций $f \in F(1, x_1, x_2)$. При этом достаточно использовать отрицание лишь одной переменной (т. е. для реализации всех функций, существенно зависящих от двух переменных, достаточно использовать множество $\{0, 1, x_1, x_2, \bar{x}_1\}$ функций на входах шаблона). Для каждой функции $out \in Out(3, x_1, x_2, x_3)$ на шаблонах размера 2 найдены логические реализации при подключении к входам функций $f \in F(2, x_1, x_2, x_3)$. На шаблонах размера 3 найдены реализации для 24 логических функций $out \in Out(3, x_1, x_2, x_3)$ при условии, что к входам шаблонов подключались лишь функции $f \in F(1, x_1, x_2, x_3)$. На шаблонах размера 3 с двумя выходами были найдены реализации для 108 различных пар функций $out \in Out(2, x_1, x_2, x_3)$. Для шаблонов размера 4 (здесь рассматривались только шаблоны с общими затворами для пары транзисторов) найдены реализации для 160 логических функций $out \in Out(4, x_1, x_2, x_3, x_4)$ при подключении функций $f \in F(1, x_1, x_2, x_3, x_4)$.

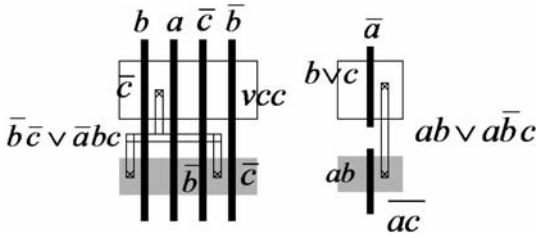


Рис. 7. Пример реализации на транзисторном шаблоне

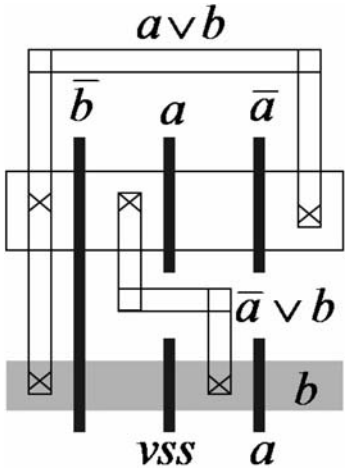


Рис. 8. Шаблоны с двумя выходами

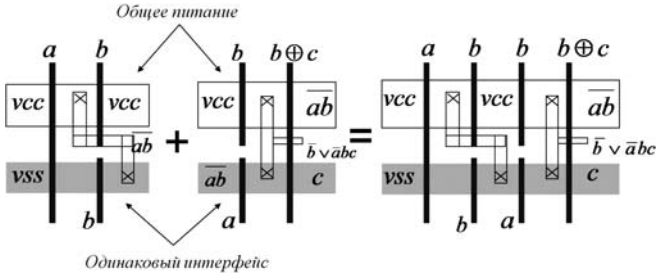


Рис. 9. Конкатенация шаблонов с одинаковым интерфейсом

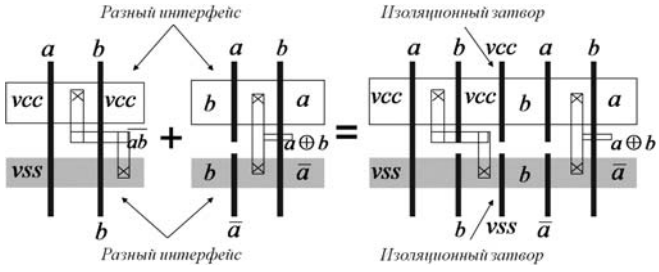


Рис. 10. Использование изоляционных затворов

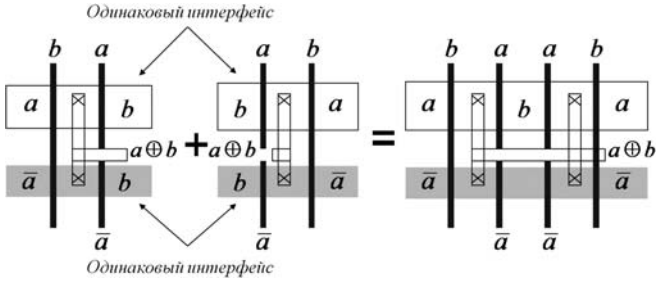


Рис. 11. Пример масштабирования шаблона

На рис. 7 показана возможность того, что на шаблоне, состоящем из двух транзисторов, можно реализовать трехвходовые функции, используя двухвходовые функции на входах. На рис. 8 показан пример использования шаблонов с двумя выходами.

Конкатенация шаблонов. Синтез логического блока может быть осуществлен посредством операции декомпозиции логической функции. Реализован алгоритм декомпозиции на основе заранее созданной базы данных реализаций логических функций на шаблоне. Алгоритм также учитывает, что логические реализации, имеющие подходящий набор логических функций в точках интерфейса шаблона, могут конкатенироваться посредством разделения общих сильнолегированных областей (рис. 9). В случае необходимости конкатенации транзисторных шаблонов, которые не имеют одинаковых логических функций в точках интерфейса, используется изоляционный затвор, который разделяет общую сильнолегированную область на электрически неэквивалентные части (рис. 10).

Симметрия регулярного шаблона позволяет отразить его зеркально относительно вертикальной оси. Это свойство позволяет внести способ масштабирования эквивалентных транзисторных шаблонов (рис. 11).

Пример синтеза на шаблонах: сумматор. На рис. 12 демонстрируется операция конкатенации шаблонов на примере сумматора (схема с двумя входными битами для суммирования: *a* и *b*; одним дополнительным входным битом переноса — *c* и с двумя выходными битами суммы и переноса: *sum* и c_{out}). В табл. 1 показано сравнение сумматора, спроектированного на транзиторных шаблонах и на стандартных ячейках.

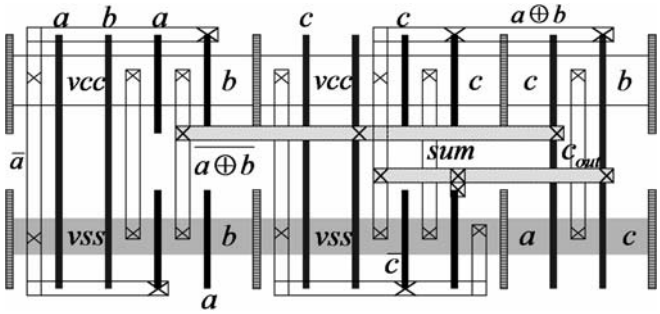


Рис. 12. Сумматор на шаблонах

Таблица 1

Сравнение для сумматора, функциональные и нефункциональные затворы (Ф и НФ)

Затворы	Ячейки	Шаблоны
Ф	32	20
НФ	30	8
Всего	62	28
% Ф	58 %	71 %

Изоляционные затворы выделены полосатой окраской, трассировка сделана в двух разных слоях металла (первый слой металла — вертикальные проводники, второй — горизонтальные).

Данный пример показывает преимущество метода на шаблонах в сравнении с синтезом на стандартных ячейках по числу используемых затворов.

Экспериментальные результаты

Проведено сравнение топологий на транзисторном уровне для синтеза на стандартных ячейках и для синтеза на основе шаблонов. В качестве входных задач для синтеза рассматривались логические блоки с тремя входами (логические переменные) и от одного до трех выходов (логические функции). Площадь оценивалась числом затворов как функциональных, так и нефункциональных, так как в обоих подходах используется регулярная сетка затворов.

Процедура синтеза на стандартных ячейках осуществлялась промышленными программами, минимизирующими площадь кристалла. В качестве библиотеки использовалась стандартная библиотека ячеек, включающая сложные AOI/OAI (And-Or-Inverter/Or-And-Inverter) элементы.

Для синтеза на транзисторных шаблонах подготовлена база данных шаблонов размера 2. В базе для каждой логической функции с 2...3 входами хранятся шаблон и логические реализации для данной функции на нем. Все функции на входах существенно зависели от меньшего числа переменных, чем функция на выходе. Число затворов для синтеза на шаблонах включает изоляционные затворы на границах логических блоков.

В табл. 2 приведены число затворов разного типа для сравниваемых подходов, а также отношение между ними. Число в скобках — это число различных входных задач. Статистика показывает, что решение на шаблонах требует меньшего числа функциональных и нефункциональных затворов. Доля функциональных затворов в среднем выше для синтеза на шаблонах (67 % против 51 %). Отношение числа затворов для синтеза на шаблонах и для синтеза на стандартных ячейках уменьшается с увеличением числа выходов. Причина этого эффекта заключена в более мощной операции суперпозиции, которая имеет место вследствие большего числа входных точек у шаблона по сравнению со стандартной ячейкой.

Таблица 2

Сравнение по площади

Тип схемы	Тип	Ячейки	Шаблоны	Отношение
3вх-1вых(218)	Ф	2712	2630	0,97
	НФ	2544	1692	0,67
	Всего	5256	4322	0,82
3вх-2вых(1000)	Ф	24085	22436	0,93
	НФ	22655	11042	0,49
	Всего	46740	33478	0,72
3вх-3вых(1000)	Ф	34221	30082	0,88
	НФ	32181	14372	0,45
	Всего	66402	44454	0,67

Заключение

Регулярная топология становится неотъемлемой частью интегральной схемы. Логические функции могут быть реализованы на регулярных заготовках. Предложено возможное определение регулярной заготовки и транзисторная модель на уровне переключений для описания функциональности такой подсхемы. Также показано, что при синтезе на основе шаблонов потенциально используется меньшее число функциональных и нефункциональных затворов. Доля функциональных затворов больше в предлагаемом подходе, чем в подходе на стандартных ячейках.

Подход на шаблонах дает большие возможности для операции суперпозиции. На входах стандартной ячейки может использоваться множество и зависимых, и независимых переменных. Такое естественное свойство стандартных ячеек, как способность работать на любых входных сигналах, увеличивает площадь реализации логических функций. В случае шаблонов все входы, как правило, не могут быть полностью независимыми из-за возможных замыканий или, в терминах введенной транзисторной модели переключений, по причине возможности возникновения *bad*-состояния на выходе. Другими словами, шаблон использует меньшее число транзисторов и имеет больше возможностей для суперпозиции логики.

В настоящее время авторы исследуют электрические характеристики получаемых схем, а также алгоритм синтеза на шаблонах для логических функций с большим числом существенных переменных, не использующий подготовленную базу данных логических реализаций. Также планируется рассмотреть такие алгоритмы оптимизации, как масштабирование транзисторов, буферизация критических путей и др.

Список литературы

1. **Kuon I., Rose J., Rogers S.** Measuring the Gap Between FPGAs and ASICs // Proc. of FPGA. 2006. P. 23—30.

2. **Ran Y., Marek-Sadowska M.** An integrated Design Flow for a Via-Configurable Gate Array // Proc. of DAC. 2004. P. 582—589.

3. **Koopman R. J. H., Kerkhoff H. G.** A General-Purpose High-Density Sea-of-Gates Architecture // Proc. of 1993 IEEE. 1993. P. 1388—1391.

4. **Ran Y., Marek-Sadowska M.** On designing Via-Configurable Cell Blocks for Regular Fabrics // Proc. of DAC. 2004. P. 198—203.

5. Ran Y., Marek-Sadowska M. The Magic of Via-Configurable Regular Fabric // Proc. of ICCD. 2004. P. 1—6.

УДК 004.946;519.673

И. В. Матюшкин, канд. физ.-мат. наук, нач. лаб.,
ОАО "НИИМЭ и завод Микрон"
e-mail:imatyushkin@sitronics.com

Перспективы развития современных средств проектирования клеточных автоматов

Проанализированы существующие машины клеточных автоматов (МКА), отмечены их недостатки и сформулированы требования, предъявляемые к ним современным уровнем развития нанотехнологии. Указаны направления модернизации МКА. Предложен новый МКА SoftCAM, архитектура которого зафиксирована UML-диаграммами.

Ключевые слова: *клеточные автоматы, САПР, UML*

Тематика клеточных автоматов оказалась широко востребованной в последнее десятилетие, число публикаций по ней неуклонно растет [1]. Клеточный автомат (КА) является фундаментальной абстракцией для представления параллельных вычислений подобно тому, как машина Тьюринга и конечный автомат репрезентируют последовательные вычисления [2]. Модель КА также может использоваться при эскизном проектировании мультипроцессорных систем или при исследовании сравнительно простых мультиагентных интеллектуальных систем. Прикладное значение КА связано с их использованием в качестве метода математического моделирования [3]. Например, на языке КА моделируются процессы диффузии, распространения лесного пожара, эпидемии и в целом рассчитываются пространственно-распределенные системы, включая квантовые [4]. В микроэлектронике известны применения КА для моделирования процесса травления при получении пористого кремния [5]. Не лишено значения применение КА в изобразительном искусстве, поскольку визуализации КА не уступают по красоте фрактальной графике [6].

Термин "машина клеточных автоматов" (МКА) введен Тоффоли. МКА является системой автоматизации проектирования клеточных автоматов; задавая правила функционирования клеточного автомата, можно реализовывать ту или иную математическую модель (или мультиагентную систему). Целью данной статьи является анализ существующих МКА, формулировка требований, предъявляемых к МКА современными нанотехнологиями, и выявление возможных путей развития МКА. Также впервые посредством UML-диаграмм предложена оригинальная архитектура МКА.

Обзор существующих МКА

Рассмотрим вначале "легкие" и свободно распространяемые МКА. Информация по ним суммирована в таблице. Долгое время, начиная с 70-х гг., когда игра "Жизнь" получила известность в университетской среде США и Европы, МКА создавались безызвестными энтузиастами. До сих пор создаются простые симуляторы игры "Жизнь" на основе флэш-анимации, Java-апплетов. Однако если раньше такие МКА создавались исследователями-любителями, ищущими новые конфигурации, то теперь здесь пробуют силы новички в программирова-

Характеристики основных свободно распространяемых МКА

Характеристики	Наименование программы			
	Fam life	MCell (MJCell)	Life32	Golly
Автор	Мозжухин Андрей, Фетисов Александр	Mirek Wijtowicz	Johan Bontes	Tomas Rokicki, Andrew Trevorrow, Dave Greene, Jason Summers, Tim Hutton
Год издания	1998/2002	1999/2001 (2005)	1999/2002	2005/2009
Сайт разработчика/Поддержка	http://www.fam-life.narod.ru/Нет	http://www.mirekw.com/ca/Нет	http://psoup.math.wisc.edu/Life32.html/Нет	http://golly.sourceforge.net/Да
Объем на жестком диске, Мбайт	1,2	1,75 (ядро)	1,73	9
Объем в оперативной памяти при запуске программы, Мбайт	1	13	9	12
Возможность изменять правила	Да	Да	Да	Да
Встроенные конфигурации	Да	Да	Нет	Да
Язык написания	Не документировано	Borland Delphi 5.0 (+Java)	Delphi 3.02	Delphi
Возможность замыкания границ пространства	Не документировано	Не документировано	Не документировано	Не документировано
Быстродействие, с (время расчета 10 000 ходов)	18	8	3	16 (в режиме Hyperspeed — 1)
Максимальный размер поля	1680 × 1025	100 000 × 2500	1 × 1 млн	Не ограничен

нии. Примером служит МКА Life 3D, где обычная 2D-игра визуализирована в 3D-пространстве с помощью API OpenGL. Также разработчики пакета MatLab включили в целях обучения в состав демонстрационных М-файлов небольшой симулятор игры "Жизнь".

Важным шагом в развитии МКА было появление симулятора Life 1.05 (автор: Alan Hensel, платформа: DOS, размер дистрибутива: 200 Кбайт, которое де-факто ввело стандарт LIF для записи конфигураций и упрощенную запись правил перехода. Наряду с российской разработкой FAM-life стоит отметить Life Editor 3 (автор: Владимир Крылов при сотрудничестве с фирмой "Геймос", дата: 1991—1994 гг.), продемонстрировавший арт-возможности клеточных автоматов.

Подробнее остановимся на последней по времени МКА Golly. Помимо открытого кода и кроссплатформенности она обладает следующими достоинствами:

- размер поля ограничен только физической памятью;
- число состояний ячейки до 256;
- использует быстрый и эффективный по занимаемой оперативной памяти алгоритм расчета QuickLife;
- возможность замены алгоритма расчета, в частности, алгоритм HashLife (Билл Госпер, 1984), основанный на хранении и хэшировании уже вычисленных фрагментов конфигураций и эффективный при симуляции на длительных временах больших по размеру конфигураций;
- наличие библиотек, содержащих многие классические варианты КА: ID-игра Стивена Вольфрама, "Мир-проволока", "Поколения", автомат фон Неймана с 29 состояниями, самовоспроизводящийся автомат Фредкина;
- интерфейс для задания собственных правил перехода;
- экспорт/импорт конфигураций и паттернов для LIF-, RLE-, MCL-файлов, а также форматов macrocell и dblife;
- поддержка стандартных графических bmp, tiff, gif, png-форматов;
- большая коллекция "удачных" паттернов;
- гибкое управление симуляцией через загрузку скриптов языков Perl и Python;
- специально разработанная система справки Life Lexicon, основанная на HTML.

Для Golly (рис. 1, см. четвертую сторону обложки) отметим великолепно реализованные элементы интерфейса для задания исходных конфигураций с возможностью масштабирования поля и импорта библиотечных паттернов, а также скриптовую поддержку.

Теперь кратко рассмотрим более "тяжелые" и соответственно закрытые и частично коммерческие МКА. Н. Марголус и Т. Тоффоли [2], начиная с середины 80-х гг. XX века, проводили в Массачусетском технологическом институте работы по симуляции клеточных автоматов для нужд биологии, кристаллографии и других прикладных дисциплин. Последней моделью указанной серии является CAM-8, технически представляющая собой систолический массив процессоров, симулирующий параллельную архитектуру SIMD-типа. Собственно говоря, авторы термина "машина клеточных автоматов" изначально придавали ему более узкое значение, чем используется в данной статье. Они неявно полагали дополнительно, что физическая реализация вычислителя предполагает параллелизм при симуляции КА. И действительно, клеточный автомат может сам моделировать многопроцессорную систему, но, вместе с тем, его симуляция действительно эффективна при проведении распределенных вычислений, а не последовательных, выполняемых на компьютере с архитектурой фон Неймана. В настоящее время создан ряд компьютеров, специализированных для таких вычислений. Одними из пионерских и серьезных практических разработок в данном направлении можно отметить клеточные процессоры Легенди [7] и ML-сопроцессоры. Итальянскими специалистами [8] в 1995 г. создана вычислительная среда CAMEL, использующая КА-модель в качестве теоретической основы и успешно применяемая для моделирования.

Безусловно, стоит отметить разработку питерских ученых Л. Наумова и А. Шалыто CAME&L (2007), которая не только обладает широкими функциональными возможностями МКА (изменение правил перехода достигается подключением внешних C++ библиотек), но и обеспечивает автоматизацию проектирования программного обеспечения систем с программируемой логикой (ПЛИС).

Для рядовых исследователей, потенциальных потребителей МКА, доступ к высокопроизводительным вычислительным системам затруднителен, поэтому становится актуальной задача создания МКА "средней тяжести", которые, с одной стороны, были бы реализуемы на персональном компьютере (с возможностью, если необходимо, сетевых вычислений), но, с другой стороны, обладали бы большей функциональностью, чем ориентированные скорее на занимательность "легкие" МКА.

Нельзя не сказать кратко об упомянутых форматах файлов. Здесь используется старая концепция знакоместа, а сами файлы имеют по сути текстовый формат. В RLE-файлах мертвая клетка кодируется символом "b", а живая — символом "o". Для перехода на следующую строку используется символ $, а для окончания записи паттерна символ !. Если более двух одинаковых клеток идут подряд, то они не выписываются все, а перед со-

ответствующим символом ставится число повторений. Обязательной является также вводная строка (заголовок), в которой указываются размеры ограничивающего прямоугольника и, возможно, правила перехода. Даже программе Golly правила задаются примитивно; для автомата "Поколения" — это три числа S/B/C, где S — набор цифр от 0 до 8, определяет число "живых" соседей, при котором клетка остается "в живых"; B — набор цифр от 0 до 8, определяет число "живых" соседей, при котором "мертвая" клетка становится "живой"; C — число, определяет число ходов "умирания" клетки.

Требования, предъявляемые к МКА

Цель разработчика программного обеспечения всегда состояла в предсказании и упреждении потенциальных пожеланий заказчика (пользователя). Можно выделить четыре группы пользователей:

1. Обычные пользователи, увлеченные красотой получаемых картинок и качеством анимации.

2. Специалисты по математическому моделированию, которым важно удобство задания КА, симулирующего физические (или даже социальные) процессы.

3. Разработчики распределенных систем, нуждающиеся в наибольшей свободе для выбора нестандартных правил, прежде всего относящихся к топологии КА и усложнению структуры ячейки, включая введение вероятностных мотивов и черт гетерогенности; тогда КА превращается в мультиагентную систему, что характерно, например, для мультипроцессорных систем.

4. Специалисты-математики, пытающиеся экспериментальным путем сформулировать или косвенно подтвердить гипотезы в области теории КА.

Из-за ограниченности вычислительных возможностей компьютера требования разных групп пользователей могут противоречить друг другу. Легко сообразить, что усложнение структуры ячейки (3-я группа), означающее увеличение памяти, приходящейся на ячейку, от 1 бит, достаточного для симуляции игры "Жизнь", до 64 байт, при 3D-топологии поля размера 256 × 256 × 256 повлечет за собой задействование всей оперативной памяти типичного компьютера (1 Гбайт), а при попытке использования дискового кэша катастрофически упадет быстродействие. При поиске компромисса следует попытаться удовлетворить принципу независимости архитектуры программы от деталей реализации.

Для всех групп пользователей, особенно 2-й и 3-й, необходимо предусмотреть расширение МКА на тот случай, когда потребуется вводить информацию непосредственно в КА во время симуляции. Классический подход к КА как к закрытой системе, реализованный в работе [2] и изначально полагаемый в теории КА, не приводит к успеху при моделировании существенно открытых систем или систем, нацеленных на обработку сигналов (в частности, когда КА рассматривается как объект управления).

При создании информационных систем обычно требуется вначале составить логическую модель предметной области или, по крайней мере, сформировать ее концептуальный базис. В нашем случае это по большей части, хотя и неполностью, сделано самими математиками при формулировке понятия КА. Основными концептами здесь выступают: ячейка, шаблон соседства, локальная функция переходов, конфигурация. При симуляции частично гетерогенных КА можно говорить об индексах соседства или функции соседства, а также о типах ячеек. Кроме того, при анализе опыта многочисленных программных реализаций игры Конуэя оказываются полезными следующие свойства МКА:

- широкое использование библиотек начальных фигур;
- интерактивное задание начальной конфигурации;
- управление скоростью симуляции;
- интерфейсные шаблоны для визуализации поля КА.

Основное требование к МКА состоит в том, чтобы предоставлять пользователю возможность средствами среды разработчика (IDE) конструировать КА из "кирпичиков" (они комплиментарны упомянутым выше понятиям), выбирать вариант его графического/файлового представления, вычислять общие характеристики его симуляции и, возможно, управлять процессом симуляции.

Конкретизируя это требование для разработчика, можно выделить такие желаемые свойства МКА:

- явное задание структуры ячейки (и ее типа);
- свободное задание в рамках одной МКА 1D-, 2D-, 3D-топологии и сложных топологий (например, на сфере или на поверхности геометрических тел), а также граничных условий для поля КА, в частности, его тороидальность;
- явное задание типа шаблона окрестности либо для каждой ячейки, либо для групп ячеек;
- возможность задания нешаблонной окрестности для индивидуальных ячеек;
- возможность динамического, т. е. во время симуляции, переопределения типа ячейки и ее шаблона окрестности;
- предоставление библиотек окрестностей и библиотек локальных функций перехода;
- возможность внедрения внешнего кода для локальных функций перехода;
- графический интерфейс для задания начальной конфигурации либо из ранее сохраненного файла, либо путем конструирования из библиотечных фигур (с возможностью их мультипликации);

- использование различных шаблонов оформления для КА, что особенно критично для 3D-структур;
- интерфейс для управления процессом симуляции, например, останова по требованию или автоостанова при "гибели" всех ячеек;
- возможность присоединения к глобальной функции перехода заданной пользователем функции, принудительно изменяющей состояния определенных ячеек (чаще всего граничных);
- сбор статистики симуляции, ее сохранение и визуализация (например, в игре Конуэя может рассчитываться число встретившихся периодических конфигураций или для каждой ячейки вычисляться ее среднее состояние).

Пути развития МКА

В настоящее время интерфейсы МКА уже хорошо развиты, предоставляя возможности экспорта/импорта конфигураций, управления цветовой гаммой и масштабированием при визуализации, а также изменения скорости симуляции. Многие программы обладают обширными библиотеками с присоединенной к ним справочной системой. Достигнут определенный прогресс в алгоритмах симуляции и стандартизован формат файлов конфигурации.

С нашей точки зрения, главной виной разработчиков "легких" МКА является загипнотизированность игрою "Жизнь", отсутствием инновационности в реализации базовых понятий клеточных автоматов с момента появления MKA Life 1.05. Из этого и следуют представляющиеся перспективными пути развития МКА, связанные с:

- усложнением структуры ячейки (от бита через байт к трехчленной монаде "вход—память—выход");
- усложнением топологии поля, заключающимся не только в переходе к 3D, но и в усложнении понятия "шаблона окрестности" (известно, например, что наноматериалы часто имеют пористую структуру со сложными связями, и моделирование таких структур инициирует прогресс МКА в этом направлении);
- введением черт гетерогенности/индивидуализации в классическое определение КА, т. е. в одном поле могут располагаться группы ячеек двух или трех типов, различающихся по окрестности и по правилам перехода;
- введением черт открытости, стохастичности и иерархичности архитектуры, что в целом приближает КА к классу нейронных сетей (можно еще добавить признак самомодифицируемости правил перехода, что приблизит КА к системам искусственного интеллекта и усилит эволюционный характер КА), но представляется все-таки отдаленной перспективой.

Указанные пути развития сохраняют значение и для "тяжелых" МКА. В меньшей степени специфичны для МКА тенденции, общие для современных САПР и носящие хотя и важный, но более технический характер: распараллеливание алгоритмов и вычислений, увеличение быстродействия за счет схем хеширования и предсказания, более эффективное использование оперативной памяти, кроссплатформенность, использование удаленных интерфейсов и поддержка сетевых вычислений, совершенствование форматов данных.

Описание МКА SoftCAM

Нами предлагается новый подход к созданию МКА, нашедший воплощение в архитектуре программы SoftCAM. Он неявно продолжает линию разработчиков Golly по скриптовой поддержке

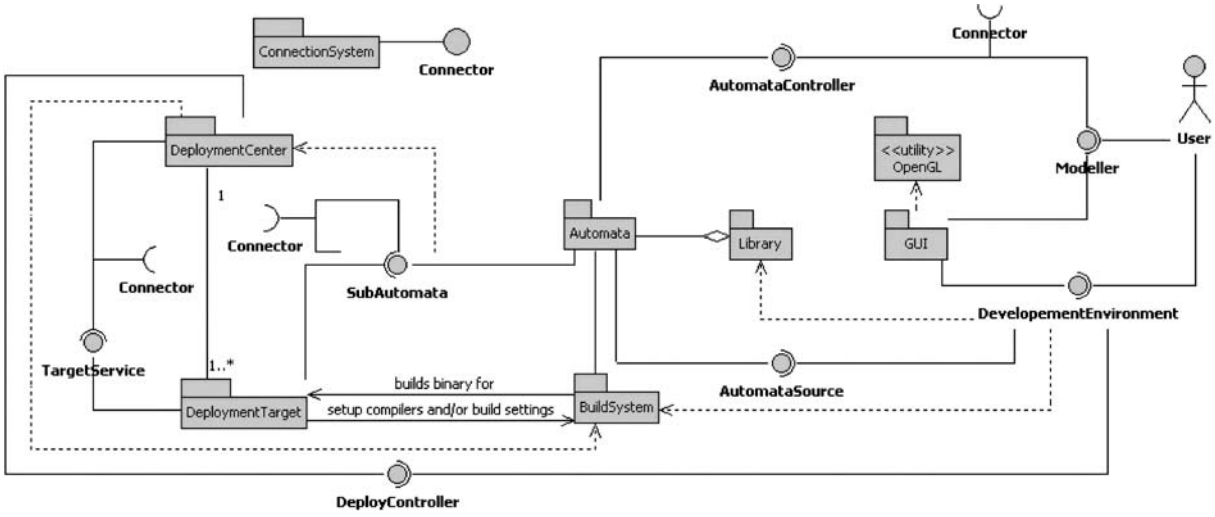


Рис. 3. Архитектура МКА SoftCAM в виде UML-диаграммы классов

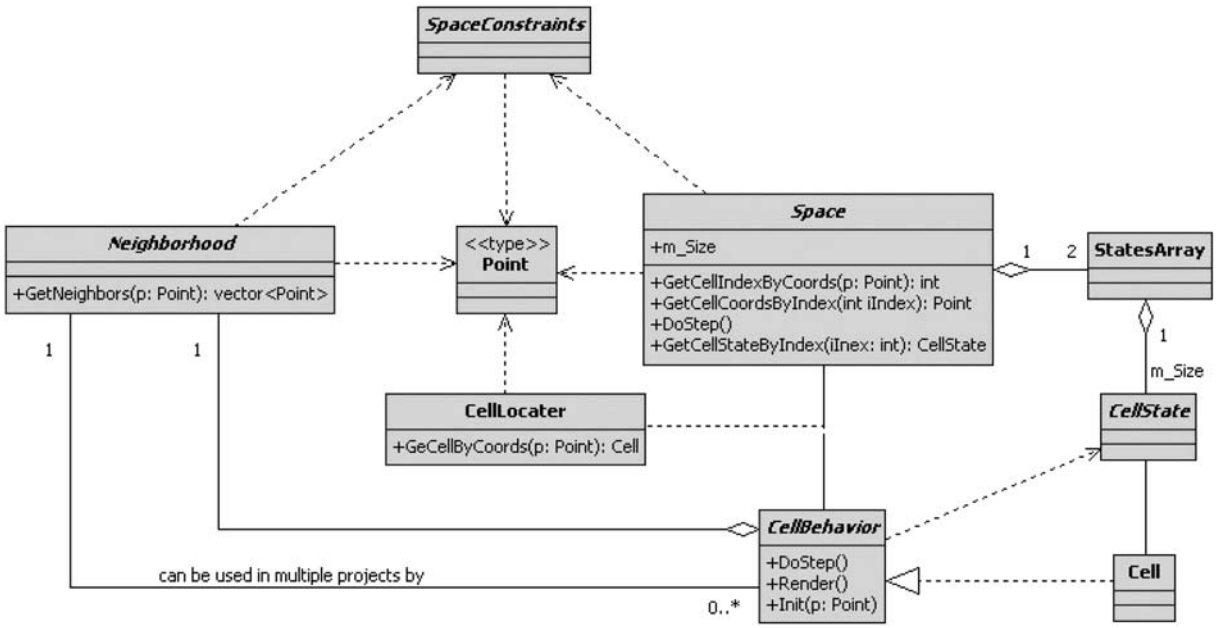


Рис. 4. Представление клеточного автомата в виде UML-диаграммы классов

(Perl и Python), делающее МКА более открытой и "доверительной" для пользователя. Однако интерпретатор кода с этих языков нуждается в дополнительной инсталляции, а общее быстродействие МКА неизбежно падает. Поэтому в SoftCAM встроен компилятор языка C++ и дополнительные библиотеки. Основные технические характеристики SoftCAM: объем на винчестере — 131 Мбайт, из них система компиляции — 110 Мбайт; занимаемая оперативная память при запуске — 9,3 Мбайт; быстродействие — 16 шагов конфигурации в секунду при размерах поля 640 × 480 и частоте процессора 3 ГГц (алгоритм симуляции не оптимизировался). Отметим, что МКА SoftCAM находится еще в стадии разработки.

Интерфейс программы скромный и ориентирован на пользователя, обладающего минимальными навыками программирования. Написанные пользователем внутри XML-структуры (рис. 2, см. четвертую сторону обложки) фрагменты кода, реализующие конкретную математическую модель, объединяются в один файл C++ кода, который затем компилируется и запускается. В программе также предусмотрены опции распределения вычислений по компьютерам локальной сети.

Ниже приведены две диаграммы классов для МКА SoftCAM, выполненные средствами программы StarUML. На рис. 3 показана архитектура в целом, а на рис. 4 — структура ядра симуляции МКА.

Заключение

Компромисс между широким спектром свободно распространяемых и немногочисленными академическими САПР в области клеточных автоматов, доступ к которым затруднен, позволит глубже их интегрировать в информационные и нанотехнологии. На основе обзора существующих решений в статье проанализированы перспективные пути развития МКА. Предложен новый подход к проектированию МКА, отличающийся большей гибкостью и открытостью по отношению к конечному пользователю, занимающемуся математическим моделированием распределенных физических, биологических или социальных объектов.

Список литературы

1. **Аладьев В. З.** Классические однородные структуры. Клеточные автоматы — CA. Palo Alto: Fultus Corporation, 2009. 536 с.

2. **Тоффоли Т., Марголус Н.** Машины клеточных автоматов. М.: Мир, 1991. 280 с.

3. **Wolfram S.** A New Kind of Science. N. Y.: Wolfram Media, 2002. 1197 с.

4. **Porod W.** Quantum-Dot Cellular Automata: Emerging Nanoelectronic Device Technologies // Proc. of Nano Engineering World Forum. Boston, Massachusetts, June 2003.

5. **Than O. and Buttgenbach S.** Simulation of anisotropic chemical etching of crystalline silicon using a cellular automata model // Sensors and actuators. Part a. October. 1994.

6. **Hopkins D.** Fun with Cellular Automata. URL: http://www.art.net/~hopkins/Don/art/cell.html.

7. **Legendi T.** Cellprocessors in Computer Architecture // Comp. Linguist. and Comp. Languages. 1976. Vol. 11. N 2. P. 147—167.

8. **Cannataro M.** et al. A parallel cellular automata environment on multicomputers for computational science // Parallel Computing. 1995. Vol. 21. P. 803—823.

9. **Наумов Л.** Метод введения обобщенных координат и инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов // Дисс. на соиск. уч. ст. канд. техн. наук. СПбГУ ИТМО. 2007. 283 с.

УДК 519.6

А. П. Карпенко, д-р техн. наук,
МГТУ им. Н. Э. Баумана, г. Москва
e-mail:apkarpenko@mail.ru

Методика оценки релевантности документов онтологической базы знаний

Работа выполнена в контексте исследований по разработке методов построения онтологических баз знаний, ориентированных на поддержку принятия решений в корпоративных информационных системах.

Рассматривается подход к поиску решений в базах знаний с использованием метаданных документа. Метаданные формируются на основе онтологии соответствующей предметной области, заданной в виде семантической сети.

Релевантность документа оценивается близостью в некоторой метрике семантической сети этого документа и семантической сети запроса.

Ключевые слова: *онтология, поддержка принятия решений, корпоративная база знаний, семантическая сеть, паттерн проектирования, релевантность*

Введение

Можно выделить три следующих класса систем поддержки принятия решений (СППР) — системы, основанные на использовании типовых решений, типовых правил синтеза решений и на поиске прецедентов. Корпоративная база знаний представляет собой, как правило, совокупность разного рода слабоструктурированных документов, в которых с той или иной степенью подробности описаны прецеденты — ситуации и решения, которые были приняты в этих ситуациях. В СППР, которые используют такие базы знаний, поиск решения заключается в поиске в этих базах наиболее подходящих прецедентов и соответствующих им документов [1].

Эффективность поиска решений в базах знаний прецедентов в значительной мере зависит от используемых методов поиска. Современные поисковые системы основаны, преимущественно, на применении полнотекстового поиска — поиска в каждом из документов всех терминов, входящих в запрос. При этом учитывается частота встречаемости терминов в документе и их средняя языковая частотность [2].

Более эффективной альтернативой полнотекстовому поиску является поиск по метаданным, т. е. по атрибутам документов, содержащимся в их метаданных. При этом классический атрибутивный поиск основывается на использовании в качестве метаданных документа преимущественно его регистрационных атрибутов (авторы документа, название документа, дата создания, тема и т. п.) [3].

Эффективный поиск решений в базах знаний прецедентов должен, очевидно, основываться не на регистрационных атрибутах документов, а на параметрах, характеризующих ситуацию принятия решения и само решение. Поэтому для СППР классический поиск по метаданным может играть лишь вспомогательную роль.

В работе рассматривается подход к поиску решений в базах знаний прецедентов, в котором метаданные формируются на основе онтологии соответствующей предметной области, заданной в виде семантической сети. Релевантность документов оценивается близостью в некоторой метрике концептов, входящих в метаданные документа, и концептов поискового запроса [4].

Важной составной частью предлагаемой методики оценки релевантности документа является построение его семантической сети. В работе данная задача ставится как задача огрубления графа семантической сети соответствующей онтологии [5, 6]. Рассматриваются три метода решения задачи на основе насыщенных паросочетаний — методы, использующие случайные паросочетания, паросочетания из тяжелых ребер, а также паросочетания из тяжелых клик [5, 7, 8].

В общей постановке о задаче поиска информации следует говорить в терминах модели поиска, которая включает в себя способ представления документов, способ представления поисковых запросов, вид критерия релевантности документов [9].

В данной работе документы в базе знаний, а также поисковые запросы к этой базе представляются в виде фреймов, называемых паттерном проектирования и паттерном запроса соответственно. Слоты этих паттернов соответствуют ролям концептов используемой онтологии (предметная область, объект, свойство, действие, задача и т. д.) [1].

Указанные роли разбивают концепты онтологии, документа и запроса к базе знаний на кластеры. Предлагается методика построения семантических сетей этих кластеров. Таким образом, поисковые образы документа и запроса представляются в виде

совокупности семантических сетей, соответствующих слотам паттерна проектирования и паттерна запроса.

В работе предложено несколько мер релевантности ролевых кластеров документа, формализующих близость семантических сетей поискового образа документа и семантических сетей запроса. На основе указанных мер предложен алгоритм оценки релевантности документа запросу.

Модели семантических сетей

Представим семантическую сеть $S(O)$ рассматриваемой онтологии O в виде взвешенного связного мультиграфа $G(O)$. Узлы этого графа соответствуют концептам множества $C(O) = \{c_i, i \in [1{:}n^O]\}$, а ребра — четким бинарным отношениям, каждое из которых принадлежит одному из типов U_p, $p \in [1{:}m^O]$.

Аналогично определим семантическую сеть $S(T)$ документа T в виде связного взвешенного обыкновенного графа $G(T)$. Узлы этого графа соответствуют $n^T \leqslant n^O$ концептам $C(T) \subset C(O)$ документа T, а ребра — связям между ними.

Так же как в работе [4] введем следующие обозначения ($p \in [1{:}m^O]$):

$d_p^O(c_i)$ — U_p-локальный кластер концепта c_i, представляющий собой совокупность всех концептов семантической сети $S(O)$, включая сам концепт c_i, которые связаны отношением типа U_p с концептом c_i в узком смысле;

$D_p^O(c_i)$ — U_p-глобальный кластер концепта c_i, образованный всеми концептами семантической сети $S(O)$, включая сам концепт c_i, которые связаны отношением типа U_p с концептом c_i в широком смысле;

$D_p^O(l, c_i)$ — $U_{p,l}$-глобальный кластер концепта c_i, имеющий смысл совокупности всех концептов кластера $D_p^O(c_i)$, включая концепт c_i, которые расположены на "расстоянии" $l \in [1{:}a(D_p^O(c_i))]$ от указанного концепта; $a(D_p^O(c_i))$ — диаметр кластера $D_p^O(c_i)$;

$n_p^O(c_i)$, $N_p^O(c_i)$, $N_p^O(l, c_i)$ — числа концептов в кластерах $d_p^O(c_i)$, $D_p^O(c_i)$, $D_p^O(l, c_i)$ соответственно.

Веса узлов и ребер графа семантической сети онтологии

Обозначим w_i^O, $i \in [1{:}n^O]$, веса узлов графа $G(O)$. Пусть также $\{\nu_{i,j,p}^O, p \in [1{:}m^O]\}$ – $(1 \times m^O)$-вектор весов ребра (c_i, c_j), $i, j \in [1{:}n^O]$, $i \neq j$ этого графа. Здесь $\nu_{i,j,p}^O = 0$, если концепты (c_i, c_j)не связаны между собой отношением типа U_p; $\nu_{i,j,p}^O = \nu_p^O$ — в противном случае; ν_p^O — вес отношений типа U_p в онтологии O.

Определение весов ν_p^O, $p \in [1{:}m^O]$. Пусть индекс i пробегает значения из интервала $[1{:}n^O]$, а индекс l принимает значения из интервала $[1{:}a(D_p^O(c_i))]$.

В качестве веса ν_p^O могут быть использованы следующие величины [10]:

- общее число $\nu_p^O = n_p^O$ концептов онтологии O, связанных между собой отношением типа U_p;
- максимальная или суммарная высота $h(D_p^O(c_i))$ кластеров $D_p^O(c_i)$

$$\nu_p^O = \max_i h(D_p^O(c_i)),\ \nu_p^O = \sum_i h(D_p^O(c_i)),$$

если отношения типа U_p представляют собой отношения частичного порядка;
- максимальный или суммарный диаметры $a(D_p^O(c_i))$ кластеров $D_p^O(c_i)$

$$\nu_p^O = \max_i a(D_p^O(c_i)),\ \nu_p^O = \sum_i a(D_p^O(c_i));$$

- максимальная или суммарная реберная плотность $b(D_p^O(c_i))$ тех же кластеров

$$\nu_p^O(l) = \max_i b(D_p^O(l, c_i)),\ \nu_p^O(l) = \sum_i b(D_p^O(l, c_i)).$$

Наряду с рассмотренными весами ν_p^O могут быть использованы их нормированные тем или иным образом аналоги, например,

$$\nu_p^O = n_p^O/n^O,\ \nu_p^O = \max_i h(D_p^O(c_i))/h^O,$$

где h^O — высота графа $G(O)$. Большое число выражений для весов ν_p^O может быть получено на основе различных сверток рассмотренных весов.

Определение весов w_i^O, $i \in [1{:}n^O]$. Положим, что веса ν_p^O, $p \in [1{:}m^O]$ тем или иным образом определены и что индекс l принимает значения из интервала $[1{:}a(D_p^T(c_i))]$.

В качестве w_i^O могут быть использованы следующие величины [10]:

- взвешенное число концептов, содержащихся во всех U_p-локальных кластерах $d_p^O(c_i)$ или U_p-глобальных кластерах $D_p^O(c_i)$,

$$w_i^O = \sum_p \nu_p^O n_p^O(c_i); \; w_i^O = \sum_p \nu_p^O N_p^O(c_i);$$

- взвешенное нормированное число концептов, содержащихся во всех $U_{p,l}$-кластерах $D_p^O(l, c_i)$

$$w_i^O = \sum_p \nu_p^O \sum_l f(l) N_p^O(l, c_i),$$

где $f(l)$ — положительная убывающая функция своего аргумента, например $f(l) = 1/l$ (здесь и далее в подобных случаях полагается, что функция является вещественнозначной);

- взвешенный максимальный или суммарный диаметр U_p-глобальных кластеров $D_p^O(c_i)$:

$$w_i^O = \max_p \nu_p^O a(D_p^O(c_i)); \; w_i^O = \sum_p \nu_p^O a(D_p^O(c_i));$$

- максимальная или суммарная реберная плотность тех же кластеров

$$w_i^O = \max_{p,l} \nu_p^O b(D_p^O(l, c_i)); \; w_i^O = \sum_{p,l} \nu_p^O b(D_p^O(l, c_i)).$$

Наряду с рассмотренными весами w_i^O могут быть использованы их нормированные тем или иным образом аналоги, а также различные свертки этих весов. Для определения весов w_i^O может быть также использована их семантическая близость, полученная с помощью соответствующего словаря [11] или Википедии [12]. Кроме того, веса концептов могут быть определены на основе понятий центральности по близости и центральности по посредничеству [13].

Построение семантической сети документа

Перейдем, например, с помощью аддитивной свертки

$$\nu_{i,j}^O = \sum_p \lambda_p \nu_{i,j,p}^O, \; p \in [1{:}m^O] \qquad (1)$$

от взвешенного мультиграфа $G(O)$ к взвешенному обыкновенному графу, в котором вес ребра (c_i, c_j) равен $\nu_{i,j}^O$. Сохраним за полученным графом прежнее обозначение. Здесь и далее λ_p — положительный скалярный вещественный множитель, определяющий относительный вес p-го компонента аддитивной скалярной свертки вида (1).

Вес узла графа $G(T)$, соответствующего концепту $c_i \in C(T)$, обозначим w_i^T, а атрибуты его ребра (c_i, c_j) будем задавать парой $(l_{i,j}^T; \nu_{i,j}^T)$, где $l_{i,j}^T$ имеет смысл "расстояния" между концептами c_i, c_j, а $\nu_{i,j}^T$ — смысл веса ребра (c_i, c_j).

В терминах графа $G(T)$ задача построения семантической сети документа $S(T)$ сводится к решению двух следующих задач.

Задача 1 (задача определения топологии графа $G(T)$). По каким правилам связывать узлы графа $G(T)$ ребрами?

Задача 2 (задача определения весов узлов и атрибутов ребер графа $G(T)$). Исходя из каких соображений, назначать веса w_i^T узлов графа $G(T)$, а также атрибуты $l_{i,j}^T$, $\nu_{i,j}^T$ его ребер?

Определение топологии графа $G(T)$. Рассмотрим эту задачу, как задачу огрубления графа методом итерационного стягивания смежных узлов графа G^α в узлы графа $G^{\alpha+1}$, где $\alpha = 0, 1, 2, \ldots$ — номер итерации, $G^0 = G(O)$ [5]. Специфика нашего случая состоит в том, что должно быть запрещено стягивание в один узел тех узлов графа G^α, которые принадлежат графу $G(T)$.

Используем метод паросочетаний, когда граф $G^{\alpha+1}$ строится на основе графа G^α путем нахождения в графе G^α паросочетания и стягивания в мультиузел узлов, входящих в каждую из пар этого паросочетания. При этом непарные узлы графа G^α просто копируются в граф $G^{\alpha+1}$ [14]. Говоря более строго, используем насыщенные паросочетания, когда хотя бы один узел любого ребра, не вошедшего в паросочетание, инцидентен ребру, вошедшему в паросочетание [8].

Легко показать, что оценка снизу числа итераций, необходимых для построения графа $G(T)$ с использованием насыщенных паросочетаний, равна $\log(n^O/n^T)$ [4].

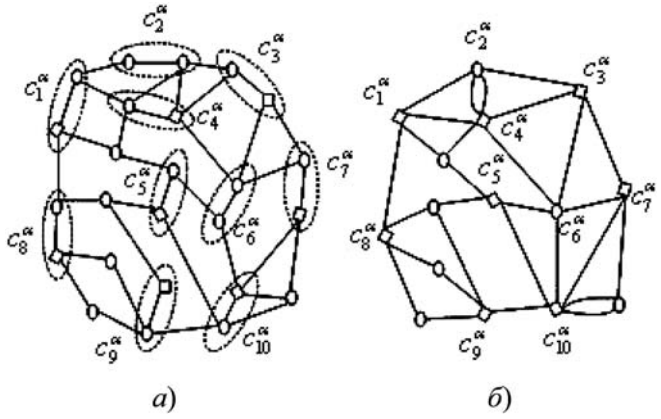
Наиболее известны три следующих метода построения насыщенных паросочетаний: случайных паросочетаний (*RM*); паросочетаний из тяжелых ребер (*HEM*); паросочетаний из тяжелых клик (*HCM*) [6].

Случайное паросочетание на итерации α строится по следующей схеме:

1) все узлы C^α текущего графа G^α объявляем немаркированными;

2) случайным образом выбираем немаркированный узел, еще не включенный в паросочетание, (пусть это будет узел c_i^α);

3) из числа немаркированных узлов, смежных узлу c_i^α, случайным образом выбираем узел (пусть это будет узел c_j^α), также еще не включенный в паросочетание;



а) *б)*

Рис. 1. К методу случайных паросочетаний: квадратиками показаны узлы графа $G(T)$:
а — граф $G^{\alpha-1}$; *б* — граф G^{α}

4) если оба узла или один из узлов пары c_i^{α}, c_j^{α} не принадлежат графу $G(T)$, то включаем ребро $(c_i^{\alpha}, c_j^{\alpha})$ в паросочетание и узлы c_i^{α}, c_j^{α} маркируем;

5) если ни одного немаркированного узла, смежного узлу c_i^{α}, не существует, то узел c_i^{α} маркируем и оставляем свободным (чтобы затем перенести его в граф $G^{\alpha+1}$);

6) если в графе G^{α} имеются еще немаркированные узлы, то переходим к п. 2.

Данную схему иллюстрирует рис. 1. На рис. 1, *а* показан граф $G^{\alpha-1}$ и сформированное на его основе паросочетание, а на рис. 1, *б* — граф G^{α}.

Паросочетание из тяжелых ребер. Схема построения этого паросочетания отличается от рассмотренной выше схемы п. 3, который в данном случае формулируется следующим образом:

- из числа немаркированных узлов, смежных узлу c_i^{α}, выбираем такой узел c_j^{α}, также еще не включенный в паросочетание, что вес ребра $(c_i^{\alpha}, c_j^{\alpha})$ является максимальным среди весов всех возможных ребер, связанных с узлом c_i^{α}.

Паросочетание из тяжелых клик. В данном случае также меняется только п. 3 рассмотренной схемы формирования случайного паросочетания:

- из числа немаркированных узлов, смежных узлу c_i^{α}, следует выбрать такой узел c_j^{α}, также еще не включенный в паросочетание, что реберная плотность мультиузла, который получается стягиванием узлов c_i^{α}, c_j^{α}, является максимально возможной по сравнению со всеми иными вариантами выбора узла c_j^{α}.

Итерации во всех рассмотренных методах формирования паросочетаний заканчиваются, когда в результате данной итерации не удалось выделить ни одной пары узлов. Другими словами, итерации заканчиваются, если в текущем графе G^{α} содержатся только узлы графа $G(T)$.

Отметим следующее обстоятельство. В силу наличия элемента случайности при формировании паросочетаний, различные итерационные процессы порождают, вообще говоря, графы $G(T)$, имеющие различную топологию. Таким образом, возникает задача получения в некотором смысле наилучшего графа $G(T)$. В качестве критерия оптимальности при этом можно использовать, например, реберную плотность графа (коэффициент кластеризации) [10, 13].

Определение весов узлов и атрибутов ребер графа $G(T)$. Выделим два следующих случая:

1) рассматриваемая пара узлов паросочетания включает в себя узел, принадлежащий графу $G(T)$ (например, пара c_1^{α} на рис. 1, *а*);

2) пара узлов содержит только узлы, не принадлежащие графу $G(T)$ (например, пара c_2^{α} на том же рисунке).

Случай 1. Пусть рассматриваемая пара узлов включает в себя узел (или мультиузел) $c_i^{\alpha} \in C(T)$ и узел (или мультиузел) $c_j^{\alpha} \notin C(T)$, веса которых равны w_i^{α}, w_j^{α} соответственно, а атрибуты ребра $(c_i^{\alpha}, c_j^{\alpha})$ определяются вектором $(l_{i,j}^{\alpha}; \nu_{i,j}^{\alpha})$. Если не оговорено противное, здесь и далее в данном разделе для простоты записи индекс T в обозначениях узлов и ребер графа $G(T)$, а также их весов и атрибутов опущен.

Будем полагать, что в процессе стягивания узлов c_i^{α}, c_j^{α} узел c_j^{α} стягивается к узлу c_i^{α}. В результате получается мультиузел $c_i^{\alpha+1} \in C(T)$, вес которого равен $w_i^{\alpha+1} = w_i^{\alpha+1}(w_i^{\alpha}, w_j^{\alpha}, l_{i,j}^{\alpha}, \nu_{i,j}^{\alpha})$, где $w_i^{\alpha+1}(w_i^{\alpha}, w_j^{\alpha}, l_{i,j}^{\alpha}, \nu_{i,j}^{\alpha})$ — некоторая положительная возрастающая функция своих аргументов w_i^{α}, w_j^{α}, $\nu_{i,j}^{\alpha}$ и такая же убывающая функция аргумента $l_{i,j}^{\alpha}$. В простейшем случае в качестве такой функции может быть использована функция вида

$$w_i^{\alpha+1}(w_i^{\alpha}, w_j^{\alpha}, l_{i,j}^{\alpha}, \nu_{i,j}^{\alpha}) = \lambda_1 w_i^{\alpha} + \lambda_2 w_j^{\alpha} \frac{\nu_{i,j}^{\alpha}}{l_{i,j}^{\alpha}}. \quad (2)$$

В результате стягивания узла c_j^{α} к узлу c_i^{α} атрибуты ребер, инцидентных узлу c_i^{α}, не меняются, а атрибуты ребер, инцидентных узлу c_j^{α}, должны быть по некоторому правилу изменены. Рассмотрим одно из таких ребер $(c_j^{\alpha}, c_p^{\alpha})$, атрибуты кото-

рого равны $l^{\alpha}_{j,p}$, $\nu^{\alpha}_{j,p}$. Это ребро заменяется ребром $(c^{\alpha+1}_i, c^{\alpha+1}_p)$ с атрибутами $l^{\alpha+1}_{j,p}$, $\nu^{\alpha+1}_{i,p}$. Естественно положить, что расстояние между узлами $c^{\alpha+1}_i$, $c^{\alpha+1}_p$ равно

$$l^{\alpha+1}_{i,p} = l^{\alpha}_{j,p} + l^{\alpha}_{i,j},$$

т. е. на расстояние $l^{\alpha}_{i,j}$ превышает расстояние $l^{\alpha}_{j,p}$. Логично также принять, что вес ребра $\nu^{\alpha+1}_{i,p} = \nu^{\alpha+1}_{i,p}(\nu^{\alpha}_{i,j}, \nu^{\alpha}_{j,p})$ является некоторой положительной возрастающей функцией своих аргументов. В простейшем случае можно положить

$$\nu^{\alpha+1}_{i,p} = \lambda_1 \nu^{\alpha}_{i,j} + \lambda_2 \nu^{\alpha}_{j,p}. \quad (3)$$

Схему рассмотренного алгоритма иллюстрирует рис. 2, где принято, что

$$w^{\alpha+1}_i(w^{\alpha}_i, w^{\alpha}_j, l^{\alpha}_{i,j}, \nu^{\alpha}_{i,j}) = w^{\alpha}_i + w^{\alpha}_j \frac{\nu^{\alpha}_{i,j}}{l^{\alpha}_{i,j}},$$

$$\nu^{\alpha+1}_{i,p} = \nu^{\alpha}_{i,j} + \nu^{\alpha}_{j,p}. \quad (4)$$

Случай 2. Положим, что оба узла рассматриваемой пары узлов c^{α}_i, c^{α}_j не принадлежат графу $G(T)$. В этом случае также можно считать, что узел c^{α}_j также стягивается к узлу c^{α}_i, так что в результате получается мультиузел $c^{\alpha+1}_i \notin C(T)$. Положим, что вес этого узла равен $w^{\alpha+1}_i$ и определяется, например, функцией вида (2).

В данном случае логично положить, что в результате стягивания узла c^{α}_j к узлу c^{α}_i меняются атрибуты всех ребер, инцидентных как узлу c^{α}_i, так и узлу c^{α}_j.

Рассмотрим ребра $(c^{\alpha}_i, c^{\alpha}_p)$, $(c^{\alpha}_j, c^{\alpha}_q)$, атрибуты которых равны $l^{\alpha}_{i,p}$, $\nu^{\alpha}_{i,p}$, $l^{\alpha}_{j,q}$, $\nu^{\alpha}_{j,q}$ соответственно. Эти ребра заменяются на ребра $(c^{\alpha+1}_i, c^{\alpha+1}_p)$, $(c^{\alpha+1}_i, c^{\alpha+1}_q)$, атрибуты которых равны $l^{\alpha+1}_{i,p}$, $\nu^{\alpha+1}_{i,p}$, $l^{\alpha+1}_{j,q}$, $\nu^{\alpha+1}_{j,q}$, где

$$l^{\alpha+1}_{i,p} = l^{\alpha}_{i,p} + 0{,}5 l^{\alpha}_{i,j},\ l^{\alpha+1}_{i,q} = l^{\alpha}_{j,q} + 0{,}5 l^{\alpha}_{i,j}, \quad (5)$$

а веса $\nu^{\alpha+1}_{i,p}$, $\nu^{\alpha+1}_{i,q}$ определяются, например, по формуле вида (3).

Схему рассмотренного алгоритма иллюстрирует рис. 3.

На рис. 3 принято, что веса узлов и ребер графа $G^{\alpha+1}$ определяются по формуле (4), а расстояние между его узлами — по формуле (5).

В результате итерации α в графе $G^{\alpha+1}$ могут появиться кратные ребра (см., например, узлы c^{α}_2, c^{α}_4 на рис. 1, *б*). Прежде чем переходить к основному циклу $(\alpha + 1)$-й итерации эти ребра следует объединить.

Положим, что двумя ребрами связаны узлы c^{α}_i, c^{α}_j и атрибуты этих ребер равны $l^{\alpha}_{1,i,j}$, $\nu^{\alpha}_{1,i,j}$, $l^{\alpha}_{2,i,j}$, $\nu^{\alpha}_{2,i,j}$ соответственно. В качестве расстояния между указанными узлами $l^{\alpha+1}_{i,j}$ примем минимальное из расстояний $l^{\alpha}_{1,i,j}$, $l^{\alpha}_{2,i,j}$:

$$l^{\alpha+1}_{i,j} = \min(l^{\alpha}_{1,i,j}, l^{\alpha}_{2,i,j}).$$

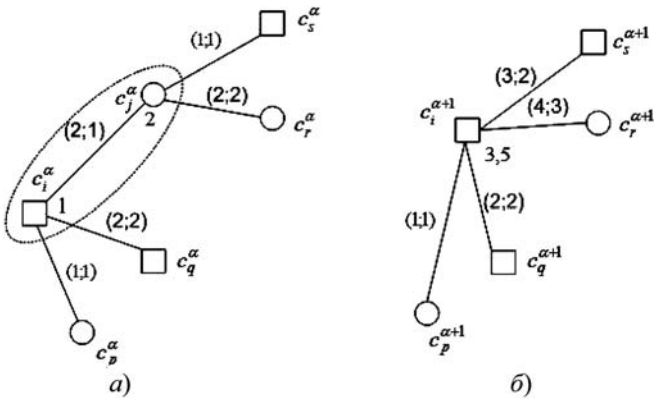


Рис. 2. К стягиванию узла $c^{\alpha}_i \in C(T)$ и узла $c^{\alpha}_j \notin C(T)$ (квадратиками показаны узлы графа $G(T)$)

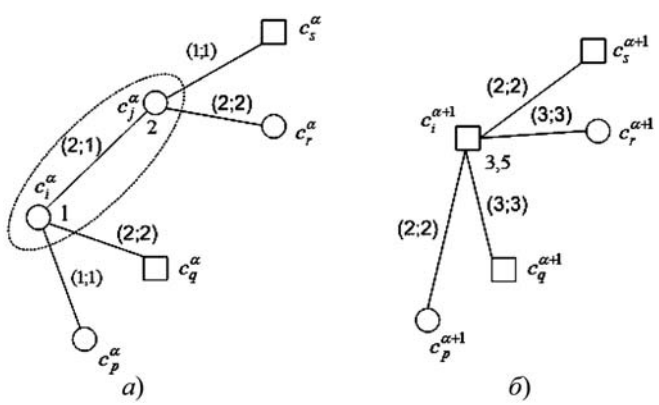


Рис. 3. К стягиванию узлов c^{α}_i, $c^{\alpha}_j \notin C(T)$ (квадратиками показаны узлы графа $G(T)$)

В качестве веса $\nu_{i,j}^{\alpha+1}$ логично принять сумму весов указанных ребер:

$$\nu_{i,j}^{\alpha+1} = \nu_{1,i,j}^{\alpha} + \nu_{2,i,j}^{\alpha}.$$

Таким образом, после завершения итераций оказываются полностью определенными топология графа $G(T)$, а также веса его узлов w_i и атрибуты ребер $l_{i,j}$, $\nu_{i,j}$; $i, j \in [1{:}n^T]$, $i \neq j$.

Вернемся к использованию индекса T в обозначениях узлов и ребер графа $G(T)$, а также их весов и атрибутов.

Исключим из числа атрибутов ребер графа $G(T)$ расстояния $l_{i,j}^T$ и модифицируем веса $\nu_{i,j}^T$ ребер этого графа: положим, что "новый" вес ребра (c_i, c_j) равен $\nu_{i,j}^T = \nu(l_{i,j}^T, \nu_{i,j}^T)$, где $\nu(l_{i,j}^T, \nu_{i,j}^T)$ — некоторая положительная убывающая функция расстояния $l_{i,j}^T$ и такая же возрастающая функция "старого" веса $\nu_{i,j}^T$. Например, можно принять

$$\nu_{i,j}^T = \lambda_1 \frac{\nu_{i,j}^T}{l_{i,j}^T}.$$

Нормировать веса узлов и ребер полученного графа $G(T)$ можно, например, следующим образом:

$$w_i^T = \frac{w_i^T}{w_{\max}^T}; \ \nu_{i,j}^T = \frac{\nu_{i,j}^T}{\nu_{\max}^T}; \ i, j \in [1{:}n^T]; \ i \neq j.$$

Здесь $w_{\max} = \max\limits_i w_i^T$, $\nu_{\max}^T = \max\limits_{i,j} \nu_{i,j}^T$ — максимальный вес узла и ребра в графе $G(T)$ соответственно.

Ролевая кластеризация семантических сетей онтологии и документа

Положим, что выделено k ролей ω_i, $i \in [1{:}k]$, концептов $C(O)$ [1]. Роли ω_i разбивают все множество концептов $C(O)$ на k непересекающихся "ролевых" кластеров D_i^O, среди которых могут быть и пустые кластеры. Множество концептов, принадлежащих кластеру D_i^O, обозначим C_i^O, а число концептов в этом кластере (или, что то же самое, во множестве C_i^O) — n_i^O. Очевидно, что имеют место соотношения

$$C(O) = \bigcup_{i=1}^{k} C_i^O, \ \sum_{i=1}^{k} n_i^O = n^O.$$

Аналогично, роли ω_i разбивают множество концептов $C(T)$ документа T также на k ролевых кластеров D_i^T, концепты которых образуют множества C_i^T с числом концептов в них, равным n_i^T:

$$C(T) = \bigcup_{i=1}^{k} C_i^T; \ \sum_{i=1}^{k} n_i^T = n^T.$$

Кластерам D_i^O, D_i^T поставим в соответствие их семантические сети S_i^O, S_i^T и графы G_i^O, G_i^T; $i \in [1{:}k]$. Обозначим $w_{i,p}^O$ вес узла $c_{i,p}$ графа G_i^O, $\nu_{i,p,q}^O$ — вес ребра этого графа, связывающего его узлы $c_{i,p}$, $c_{i,q}$. Здесь $p, q \in [1{:}n_j^O]$, $p \neq q$. Аналогичные обозначения $w_{i,p}^T$, $\nu_{i,p,q}^T$ введем для графа G_i^T.

Графы G_i^O, G_i^T, $i \in [1{:}k]$, ролевых кластеров D_i^O, D_i^T могут быть построены по схеме, рассмотренной в предыдущем разделе. При этом графы G_i^O строятся на основе графа $G(O)$,а графы G_i^T — на основе графа G^T.

Отметим, что оценить качество рассмотренной ролевой кластеризации можно, например, с помощью величины, которая называется модулярность (*modularity*) графа [15].

Поисковые образы документа и запроса

Пусть в семантической сети документа T выделены ролевые кластеры D_i^T и тем или иным образом построены семантические сети этих кластеров S_i^T, а также соответствующие им графы G_i^T; $i \in [1{:}k]$. Положим, что паттерн проектирования $A(T)$ документа T имеет k слотов $A_i(T) = A_i$ и слот A_i соответствует роли ω_i [1].

Поисковый образ рассматриваемого документа T будем представлять в виде k семантических сетей S_i^T, $i \in [1{:}k]$ (рис. 4).

Рис. 4. Поисковый образ документа T_i

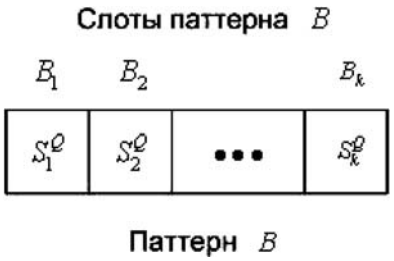


Рис. 5. Поисковый образ запроса Q

Не ограничивая общности рассуждений, положим, что поисковый образ запроса Q формируется паттерном $B = \{B_i, i \in [1:k]\}$, который также имеет k слотов B_i, где слот B_i соответствует роли ω_i.

Введем следующие обозначения ($i \in [1:k]$): C^Q — множество концептов запроса Q; D_i^Q, C_i^Q — ролевой кластер множества C^Q и множество концептов этого кластера, $C^Q = \bigcap_{j=1}^{k} C_i^Q$; n^Q, n_i^Q — числа концептов во множестве C^Q и кластере D_i^Q, $\sum_{i=1}^{k} n_i^Q = n^Q$; S_i^Q, G_i^Q — семантическая сеть кластера D_i^Q и граф этой сети; $w_{i,p}^Q$, $\nu_{i,p,q}^Q$ — вес узла $c_{i,p}^Q$ графа G_i^Q и вес ребра $(c_{i,p}^Q, c_{i,q}^Q)$ этого графа, где $p, q \in [1:n_i^Q]$, $p \neq q$.

Таким образом, поисковый образ запроса Q представляет собой k семантических сетей S_i^Q, $i \in [1:k]$ (рис. 5).

Графы G_i^Q, $i \in [1:k]$, ролевых кластеров D_i^Q также могут быть построены по рассмотренной выше схеме на основе графа $G(O)$.

Релевантность ролевого кластера документа

Определим вначале меру близости концептов множества C_i^O:

$$l(c_{i,p}, c_{i,q}) = l_{i,p,q} = \\ = \min(\nu_{i,p,\alpha}^O + \nu_{i,\alpha,\beta}^O + ... + \nu_{i,\zeta,q}^O), \qquad (6)$$

где минимум берется по всем возможным цепям $c_{i,p}$, $c_{i,\alpha}$, $c_{i,\beta}$, ..., $c_{i,\zeta}$, $c_{i,q}$, в которых все концепты принадлежат множеству C_i^O.

Во множестве C_i^T найдем для концепта $c_{i,p}$, такого, что $c_{i,p} \in C_i^Q$, $c_{i,p} \notin C_i^T$, концепт $c_{i,q}$, расстояние которого до концепта $c_{i,p}$ равно $l_{i,p,q} = l_{i,p,q}^1$. Включим полученный концепт $c_{i,q}$ во множество $D_{1,i}^Q$. Повторим процедуру для всех концептов множества C_i^Q, которые не принадлежат множеству C_i^T. Полученный в результате кластер $D_{1,i}^Q$ представляет собой совокупность концептов множества C_i^T, не принадлежащих множеству C_i^Q, но находящихся ближе всего (в смысле меры (6)) к концептам последнего множества. Положим, что мощность кластера $D_{1,i}^Q$ равна $n_{1,i}^Q$.

Аналогично, для каждого концепта $c_{i,p}$, такого, что $c_{i,p} \in C_i^Q$, $c_{i,p} \notin C_i^T$, найдем во множестве $C_i^T \setminus D_{1,i}^Q$ концепт $c_{i,q}$, расстояние которого до концепта $c_{i,p}$ равно $l_{i,p,q} = l_{i,p,q}^2$. Все полученные концепты образуют кластер $D_{2,i}^Q$. Мощность кластера $D_{2,i}^Q$ обозначим $n_{2,i}^Q$.

И т. д.

Взаимосвязи кластеров D_i^Q, $D_{1,i}^Q$, $D_{2,i}^Q$, ..., D_i^T, D_i^O иллюстрирует рис. 6.

Для каждого из концептов $c_{i,p}$ такого, что $c_{i,p} \in C_i^Q$, $c_{i,p} \notin C_i^T$, и каждого из концептов $c_{i,q} \in D_{t,i}^Q$ определим функцию $f_{i,p,q}^t(w_{i,q}^T, l_{i,p,q}^t)$, которая является положительной возрастающей функцией относительно первого аргумента и такой же убывающей функцией относительно вто-

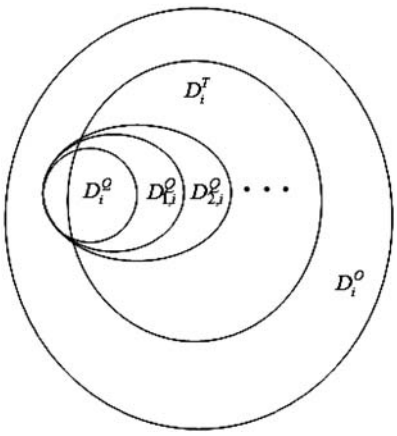


Рис. 6. К взаимосвязям кластеров D_i^Q, $D_{1,i}^Q$, $D_{2,i}^Q$, ..., D_i^T, D_i^O

рого аргумента; $t = 1, 2, \ldots$. Примером такой функции может служить функция

$$f^t_{i,p,q}(w^T_{i,q},\ l^t_{i,p,q}) = f^t_{i,p,q} = \lambda_1 \frac{w^T_{i,q}}{l^t_{i,p,q}}.$$

Функция $f^t_{i,p,q}(w^T_{i,q},\ l^t_{i,p,q})$ формализует уменьшение весов концептов из кластеров $D^Q_{t,i}$ по мере "удаления" их от кластера D^Q_i.

Обозначим $r(S^T_i,\ S^Q_i) = r_i$ меру близости семантической сети S^T_i поискового образа документа T и семантической сети S^Q_i запроса Q или, что то же самое, меру близости соответствующих графов $G^T_i,\ G^Q_i;\ i \in [1:k]$.

Меры, не учитывающие веса концептов и связей между ними. В простейшем случае в качестве мер близости графов $G^T_i,\ G^Q_i,\ i \in [1:k]$, можно использовать меры, которые не учитывают веса концептов и связей между ними. Рассмотрим некоторые из таких мер.

Мера на основе коэффициента Дайса, используемого при сравнении текстовых документов [16],

$$r_{i,1} = \frac{2n(G^Q_i \bigcap G^T_i)}{n(G^Q_i) + n(G^T_i)} = \frac{2n(G^Q_i \bigcap G^T_i)}{n^Q_i + n^T_i} \in [0,\ 1], \quad (7)$$

где $n(G^Q_i \bigcap G^T_i)$ — число узлов, содержащихся одновременно в графе G^Q_i и в графе G^T_i. Мера представляет собой относительное число концептов кластера D^Q_i, содержащихся в кластере D^T_i. В работе [16] эта мера называется мерой концептуальной близости графов $G^Q_i,\ G^T_i$.

Мера на основе относительной близости графов $G^Q_i,\ G^T_i$ [16]

$$r_{i,2} = \frac{2m(G^Q_i \bigcap G^T_i)}{m(G^Q_i) + m(G^T_i)} = \frac{2m(G^Q_i \bigcap G^T_i)}{m^Q_i + m^T_i} \in [0,\ 1], \quad (8)$$

где $m(G^Q_i) = m^Q_i$, $m(G^T_i) = m^T_i$ — числа ребер графов $G^Q_i,\ G^T_i$, соответственно; $m(G^Q_i \bigcap G^T_i)$ — число ребер графа G^T_i, принадлежащих также графу G^Q_i.

Известно, что меры вида (7), (8) сильно зависят от размеров графов [16]. Поэтому целесообразно использовать их следующие модификации, учитывающие размеры графов $G^Q_i,\ G^T_i$.

Модифицированная мера на основе меры $r_{i,1}$:

$$r_{i,3} = \frac{2n(G^Q_i \bigcap G^T_i)e_1}{n^Q_i + n^T_i}, \quad (9)$$

где

$$e_1 = \begin{cases} \lambda_1 \dfrac{n^Q_i}{n^T_i},\ n^Q_i \geqslant n^T_i, \\ \lambda_1 \dfrac{n^T_i}{n^Q_i},\ n^T_i > n^Q_i. \end{cases} \quad (10)$$

Модифицированная мера на основе меры $r_{i,2}$:

$$r_{i,4} = \frac{2m(G^Q_i \bigcap G^T_i)e_2}{m^Q_i + m^T_i}, \quad (11)$$

где величина e_2 определяется по формуле вида (10).

Очевидно, что при $e_1 = 1$ меры (9), (10) совпадают с мерами $r_{i,1}$, $r_{i,2}$ соответственно, так что последние являются частным случаем мер (9), (11).

Мера, являющаяся расширением меры $r_{i,3}$:

$$r_{i,5} = r_{i,3} + \lambda_1 \frac{n^Q_{1,i}}{n^Q_i + n^Q_{1,i}} + \lambda_2 \frac{n^Q_{2,i}}{n^Q_i + n^Q_{1,i} + n^Q_{2,i}} + \ldots . \quad (12)$$

Мера имеет смысл относительного числа узлов графа G^Q_i, содержащихся в графе G^T_i, и графах $G^Q_{1,i},\ G^Q_{2,i}, \ldots$

Мера, являющаяся расширением меры $r_{i,4}$ *и аналогичная мере* (12):

$$r_{i,6} = r_{i,4} + \lambda_1 \frac{m^Q_{1,i}}{m^Q_i + m^Q_{1,i}} + \lambda_2 \frac{m^Q_{2,i}}{m^Q_i + m^Q_{1,i} + m^Q_{2,i}} + \ldots . \quad (13)$$

Здесь $m^Q_{t,i}$ — число ребер графа $G^Q_{t,i}$; $t = 1, 2, \ldots$

Аддитивная свертка мер $r_{i,5}$, $r_{i,6}$:

$$r_{i,7} = \lambda_1 r_{i,5} + \lambda_2 r_{i,6},$$

включающая в себя все рассмотренные выше меры.

На основе мер (7)—(9), (11)—(13) легко сконструировать меры, которые учитывают только "сильные" узлы и ребра в графах $G^Q_i,\ G^Q_{1,i},\ G^Q_{2,i}, \ldots$, т. е. узлы и ребра, веса которых превышают некоторые заданные значения [17].

Меры, учитывающие веса концептов и связей между ними. Значительное число мер близости графов G_i^T, G_i^Q, $i \in [1:k]$, можно построить также путем учета весов концептов и связей между ними. Приведем примеры таких мер.

Взвешенная мера на основе меры $r_{i,1}$:

$$r_{i,8} = \frac{2\Sigma w_{i,\alpha}^Q}{\Sigma w_{i,\beta}^Q + \Sigma w_{i,\gamma}^T} \in [0, 1], \qquad (14)$$

где индекс α пробегает номера узлов, принадлежащих пересечению графов $G_i^Q \cap G_i^T$, что условно будем записывать в виде $\alpha \in [1{:}n(G_i^Q \cap G_i^T)]$; индексы β, γ пробегают номера узлов $[1{:}n_i^Q]$, $[1{:}n_i^T]$ соответственно.

Очевидно, что мера (14) эквивалентна мере (7), если принять следующие соглашения: $w_{i,\alpha}^Q = 1$ при $\alpha \in [1{:}n(G_i^Q \cap G_i^T)]$; $w_{i,\alpha}^Q = 0$ — в противном случае; $w_{i,\beta}^T = 1$, $w_{i,\gamma}^T = 1$. Таким образом, меру (7) можно считать частным случаем меры (14).

Взвешенная мера на основе меры $r_{i,2}$:

$$r_{i,9} = \frac{2\Sigma \nu_{i,\alpha,\beta}^Q}{\Sigma \nu_{i,\gamma,\delta}^Q + \Sigma \nu_{i,\varphi,\chi}^T} \in [0, 1], \qquad (15)$$

где, аналогично (14), $\alpha, \beta \in [1{:}n(G_i^Q \cap G_i^T)]$, $\gamma, \delta \in [1{:}n_i^Q]$, $\varphi, \chi \in [1{:}n_i^T]$; $\alpha \neq \beta$, $\gamma \neq \delta$, $\varphi \neq \chi$. Легко видеть, что мера (8) является частным случаем меры (15).

Модифицированные меры на основе мер $r_{i,8}$, $r_{i,9}$:

$$r_{i,10} = \frac{2\Sigma w_{i,\alpha}^Q e_1}{\Sigma w_{i,\beta}^Q + \Sigma w_{i,\gamma}^T}, \; r_{i,11} = \frac{2\Sigma \nu_{i,\alpha,\beta}^Q e_2}{\Sigma \nu_{i,\gamma,\delta}^Q + \Sigma \nu_{i,\varphi,\chi}^T}.$$

Мера, являющаяся расширением меры $r_{i,10}$:

$$r_{i,12} = r_{i,10} + \lambda_1 \frac{\Sigma w_{1,i,\beta}^Q}{\Sigma w_{i,\alpha}^Q + \Sigma w_{1,i,\beta}^Q} + \lambda_2 \frac{\Sigma w_{2,i,\gamma}^Q}{\Sigma w_{i,\alpha}^Q + \Sigma w_{1,i,\beta}^Q + \Sigma w_{2,i,\gamma}^Q} + \ldots, \qquad (16)$$

где $\alpha \in [1{:}n(G_i^Q \cap G_i^T)]$; индекс β пробегает номера узлов, принадлежащих графу $G_{1,i}^Q$, что условно будем записывать в виде $\beta \in [1{:}n(G_{1,i}^Q)]$; аналогично $\gamma \in [1{:}n(G_{2,i}^Q)]$.

Мера, являющаяся расширением меры $r_{i,11}$:

$$r_{i,13} = r_{i,11} + \lambda_1 \frac{\Sigma \nu_{1,i,\gamma,\delta}^Q}{\Sigma \nu_{i,\alpha,\beta}^Q + \Sigma \nu_{1,i,\gamma,\delta}^Q} + \lambda_2 \frac{\Sigma \nu_{1,i,\varphi,\chi}^Q}{\Sigma \nu_{i,\alpha,\beta}^Q + \Sigma \nu_{1,i,\gamma,\delta}^Q + \Sigma \nu_{2,i,\varphi,\chi}^Q} + \ldots,$$

аналогична мере (16). Здесь $\alpha, \beta \in [1{:}n(G_i^Q \cap G_i^T)]$, $\gamma, \delta \in [1{:}n(G_{1,i}^Q)]$, $\varphi, \chi \in [1{:}n(G_{2,i}^Q)]$; $\alpha \neq \beta$, $\gamma \neq \delta$, $\varphi \neq \chi$.

Аддитивная свертка мер $r_{i,12}$, $r_{i,13}$:

$$r_{i,14} = \lambda_1 r_{i,12} + \lambda_2 r_{i,13},$$

включает в себя все рассмотренные выше меры, которые учитывают веса концептов и связей между ними.

Модифицированная мера на основе меры $r_{i,10}$:

$$r_{i,15} = r_{i,10} + \lambda_1 \frac{\Sigma f_{i,\beta,\gamma}^1}{\Sigma w_{i,\alpha}^Q + \Sigma f_{i,\beta,\gamma}^1} + \lambda_2 \frac{\Sigma f_{i,\varphi,\chi}^2}{\Sigma w_{i,\alpha}^Q + \Sigma f_{i,\beta,\gamma}^1 + \Sigma f_{i,\varphi,\chi}^2} + \ldots,$$

где $\alpha \in [1{:}n(G_i^Q \cap G_i^T)]$, $\beta, \gamma \in [1{:}n(G_{1,i}^Q)]$, $\varphi, \chi \in [1{:}n(G_{2,i}^Q)]$.

Рассмотренные меры легко модифицировать, учитывая только "сильные" узлы и ребра в графах G_i^Q, $G_{1,i}^Q$, $G_{2,i}^Q$, ... [17]. Значительное число мер релевантности ролевого кластера документа может быть построено на основе мер семантической близости в сетях документов [18].

Оценка релевантности документа

Пусть поисковый образ документа T представлен паттерном проектирования $A = \{A_i, i \in [1:k]\}$, слотам которого A_i соответствуют семантические сети S_i^T и графы G_i^T. Пусть, аналогично, поисковый образ запроса Q сформирован в виде паттерна $B = \{B_i, i \in [1:k]\}$, который представляет собой совокупность k семантических сетей S_i^Q и графов G_i^Q.

Обозначим $R(T, Q) = R(r_1^Q, r_2^Q, ..., r_k^Q)$ релевантность документа T запросу Q, где $R(r_1^Q, r_2^Q, ..., r_k^Q)$ — некоторая неотрицательная возрастающая функция всех своих аргументов, например,

$$R(r_1^Q, r_2^Q, ..., r_k^Q) = \sum_{i=1}^{k} \lambda_i r_i^Q. \qquad (17)$$

Нормировать величину $R(r_1^Q, r_2^Q, ..., r_k^Q)$ можно, например, отнеся ее к сумме релевантностей всех документов рассматриваемой базы знаний.

Общая схема предлагаемой методики оценки релевантности документа T имеет вид, представленный на рис. 7.

Определение релевантности (17) можно расширить путем учета априорной "значимости" документа T, которую можно построить, например, на основе мер $\mu(S_i^O, S_i^T) = \mu_i^T$ близости семантических сетей S_i^O онтологии и семантических сетей S_i^T документа T или, что то же самое, мер близости соответствующих графов G_i^O, G_i^T; $i \in [1{:}k]$ [19]. Так, в качестве меры μ^T можно использовать подходящим образом нормированную взвешенную сумму мер $\mu_1^T, \mu_2^T, ..., \mu_k^T$:

$$\mu^T = \sum_{i=1}^{k} \lambda_i \mu_i^T.$$

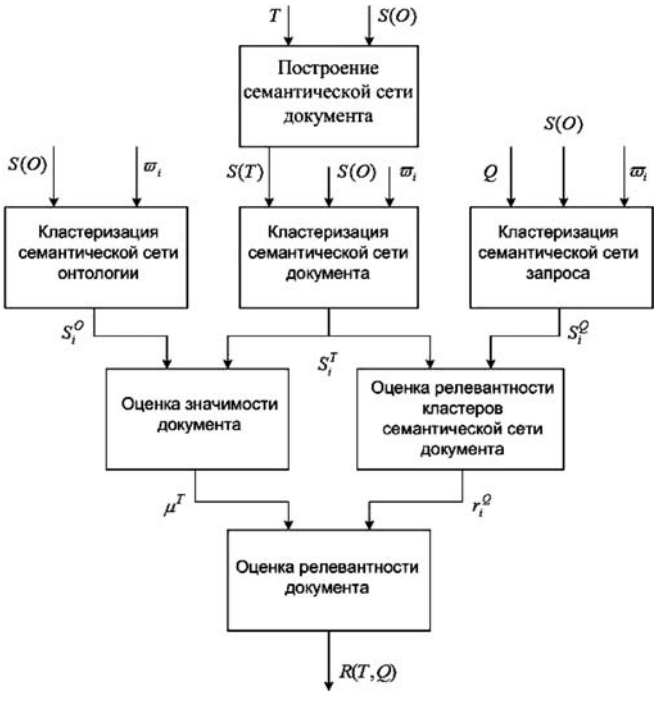


Рис. 7. Схема оценки релевантности документа: $i \in [1{:}k]$

С учетом меры μ^T формула (17) модифицируется следующим образом:

$$R(r_1^Q, r_2^Q, ..., r_k^Q, \mu^T) = \mu^T \sum_{i=1}^{k} \lambda_i r_i^Q. \qquad (18)$$

Отметим, что формулы (17), (18) не учитывают эффективность решений, которые содержатся в документе T. На основе опыта эксплуатации рассматриваемой базы знаний эта эффективность может быть оценена лицом, принимающим решения, и сохранена в базе знаний.

Заключение

В данной работе под онтологией O понимается так называемая "легкая" онтология, определяемая парой вида $O = \langle \mathbf{c}, \mathbf{r} \rangle$, где **c** — множество концептов, а **r** — множество отношений между ними. В развитии работы планируется применить предложенную в ней методику к "тяжелой" онтологии, которая определяется тройкой $O = \langle \mathbf{c}, \mathbf{r}, \mathbf{f} \rangle$, где **f** — множество функций интерпретации, определенных на концептах и/или отношениях онтологии.

Под отношениями **r** в работе понимаются четкие отношения. Однако во многих случаях более адекватной является модель онтологии, в которой эти отношения понимаются как нечеткие. В этом случае возможен анализ важности концептов с учетом различий в "силе" связей между ними.

Предложенная в работе методика оценки релевантности документов имеет высокую вычислительную сложность. Подавляющая часть требуемых вычислительных затрат обусловлена выполнением следующих работ.

Во-первых, для каждого из документов T базы знаний методика требует построения соответствующей семантической сети $S(T)$, а также построения семантической сети S_i^T, $i \in [1{:}k]$, каждого из слотов поискового образа документа. Если онтология предметной области фиксирована, то эта работа выполняется лишь однажды, при помещении документа в базу знаний.

Во-вторых, методика требует построения аналогичных семантических сетей S_i^O онтологии рассматриваемой предметной области. Опять же, если онтология предметной области фиксирована, то эта работа выполняется лишь однократно.

В-третьих, в соответствии с методикой для каждого из запросов Q также требуется формирование семантических сетей S_i^Q. Данная работа должна выполняться системой управления базой знаний при обработке каждого из запросов.

В работе широко используются аддитивные скалярные свертки (см., например, формулы (1), (2), (3) и т. д.). Очевидно, что наряду с аддитивными сверт-

ками могут быть использованы и иные, например, мультипликативные свертки или комбинации [20].

Основная задача работы — определение релевантности документа — формулируется, как задача многокритериальной (точнее — k-критериальной) оптимизации [см. формулы (17), (18)]. Использованный при решении этой задачи метод аддитивной скалярной свертки является простейшим и далеко не всегда эффективным методом решения многокритериальных задач. Поэтому представляет интерес исследование целесообразности использования других, более "тонких", методов решения указанной многокритериальной задачи [20].

Широкое использование сверток приводит к тому, что методика содержит большое число свободных параметров (см. формулы (1)—(3) и т. д.). Имеется немного содержательных оснований для априорного выбора значений этих параметров. Поэтому представляется перспективным ставить задачу определения их значений, как задачу метаоптимизации [21]. Отметим, что при этом в базе знаний требуется хранить оценки успешности поиска, сформированные лицом, принимающим решения.

Одной из проблем, которая возникает при использовании рассмотренного подхода к определению релевантности документов, является проблема лексической многозначности терминов. Правильное значение многозначного слова может быть установлено только путем анализа контекста, в котором это слово упоминается. Известен ряд методов решения данной задачи, например, методы, основанные на использовании Википедии [22].

В развитие работы планируется также экспериментальная проверка эффективности предложенной методики.

Автор выражает благодарность И. П. Норенкову за постановку рассмотренной в работе задачи, а также за конструктивные обсуждения подходов к ее решению.

Работа выполнена при поддержке гранта РФФИ 10-07-00401.

Список литературы

1. **Норенков И. П.** Интеллектуальные технологии на базе онтологий // Информационные технологии. 2010. № 1. С. 17—23.

2. **Толчеев В. О.** Методы выявления информационных признаков в задачах классификации текстовых документов // Информационные технологии. 2005. № 8. С. 14—21.

3. **The Dublin** Core® Metadata Initiative URL: http://dublincore.org/.

4. **Карпенко А. П.** Оценка релевантности документов онтологической базы знаний // Наука и образование: электронное научно-техническое издание. 2009. № 9. URL: http://technomag.edu.ru/doc/157379.html.

5. **Karypis G., Kumar V.** Multilevel k-way Partitioning Scheme for Irregular Graphs // Journal of Parallel and Distributed Computing. 1998. Vol. 8, N 1. P. 96—129.

6. **Бувайло Д. П., Толок В. А.** Быстрый высокопроизводительный алгоритм для разделения нерегулярных графов // Вісник Запорізького державного університету. 2002. № 2. С. 1—10.

7. **Bui T. N., Chaudhuri S., Leighton F. T., Sipser M.** Graph bisection algorithms with good average case behavior // Combinatorica. 1987. N 7. P. 171—191.

8. **Miller G. L., Shang-Hua Teng, Vavasis S. A.** An unified geometric approach to graph separators // Proceedings of 31st Annual Symposium on Foundations of Computer Science. 1991. P. 538—547.

9. **Когаловский М. Р.** Перспективные технологии информационных систем. М.: ДМК Пресс; М.: Компания АйТи, 2003. 288 с.

10. **Карпенко А. П.** Меры важности концептов в семантической сети онтологической базы знаний // Наука и образование: электронное научно-техническое издание. 2010. № 7. URL: http://technomag.edu.ru/doc/151142.html.

11. **Miller G. A.** et al. Wordnet: a lexical database for the english language. URL: http://wordnet.princeton.edu/.

12. **Gabrilovich E., Markovitch S.** Computing semantic relatedness using wikipedia-based explicit semantic analysis: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (UCAI-07), Hyderabad, India, January 6—12, 2007: AAAI Press, 2007. P. 1606—1611.

13. **Целых Ю. А.** Теоретико-графовые методы анализа нечетких социальных сетей. URL: http://swsys.ru/print/article_print.php?id=742.

14. **Hendrickson B., Leland R.** An improved spectral graph partitioning algorithm for mapping parallel computations. Sandia National Laboratories // Technical Report SAND92-1460, 1992. P. 192.

15. **Гринева М., Лизоркин Д.** Анализ текстовых документов для извлечения тематически сгруппированных ключевых терминов. URL: http://citforum.ru/database/articles/kw_extraction.

16. **Богатырев М. Ю., Латов В. Е., Столбовская И. А.** Применение концептуальных графов в системах поддержки электронных библиотек: Труды 9-й Всероссийской научной конференции "Электронные библиотеки: перспективные методы и технологии, электронные коллекции" — RCDL'2007. Переславль-Залесский, Россия, 2007. — Т. 2. С. 104—110.

17. **Бородкин Л. И.** Математические методы и компьютер в задачах атрибуции текстов. URL: http://www.textology.ru/library/book.aspx?bookId=11&textId=13.

18. **Lizorkin D.** et al. Accuracy Estimate and Optimization Techniques for SimRank Computation // Proc. of the 34th International Conference on Very Large Data Bases (VLDB'08). 2008. Vol. 1. Is 1. P. 422—433.

19. **Галямова Е. В., Карпенко А. П., Соколов Н. К., Ягудаев Г. Г.** Контроль понятийных знаний субъекта обучения в обучающей системе // Вестник МАДИ (ГТУ). 2009. № 2(17). С. 82—86.

20. **Ларичев О. И.** Теория и методы принятия решений, а также Хроника событий в Волшебных странах. М.: Университетская книга, Логос, 2006. 292 с.

21. **Zhang Hong, Ishikawa Masumi.** Evolutionary Canonical Particle Swarm Optimizer — A Proposal of Meta-Optimization in Model Selection. Berlin: Springer-Verlag, 2008.

22. **Mihalcea R.** Using Wikipedia for Automatic Word Sense Disambiguation: Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL 2007), Rochester. April 2007. P. 196—203.

УДК 519.76;811.93

А. В. Болховитянов, аспирант,
А. М. Чеповский, канд. техн. наук, доц.,
МГТУ им. Н. Э. Баумана
e-mail:alex.daiv@gmail.com

Методы автоматического анализа словоформ

Рассматриваются методы автоматического анализа словоформ для индоевропейских языков. Предлагается обобщенный подход к обработке словоформ из различных классов языков и приводятся реализации разработанных алгоритмов анализа. Представлены результаты тестирования качества и производительности разработанных алгоритмов на реальных массивах текстов.

Ключевые слова: *обработка естественно-языковых текстов, морфология, автоматический анализ словоформ*

Введение

В ряде задач автоматической обработки текстовой информации, например, в задачах информационного поиска, требуется проводить морфологический анализ словоформ [1, 2]. Например, в задаче построения индекса поисковой системы требуется хранить некоторую нормальную форму каждой анализируемой словоформы для обеспечения высокого качества поиска. Нормальной формой может быть как каноническая форма слова (например, единственное число, мужской род, именительный падеж для имен существительных в русском языке), так и псевдооснова, выделенная из анализируемой словоформы на основе некоторых принятых правил и не обязательно совпадающая с корнем слова.

Задача выделения псевдоосновы словоформы может решаться тремя способами:

- с помощью морфологического словаря;
- с помощью аналитических методов анализа словоформ;
- с помощью статистических методов анализа словоформ.

Первый подход имеет ограниченное применение, объясняемое тем, что в естественном языке постоянно появляются новые слова, которые отсутствуют в текущей актуальной версии словаря.

Второй подход является более естественным с той точки зрения, что слова в языке, даже новые, изменяются по правилам грамматики языка. Таким образом, при использовании аналитических методов анализа словоформ пропадает проблема новых слов, но в то же время эти методы имеют более низкие показатели качества анализа, так как в грамматических правилах часто возникают исключительные ситуации, например, неправильные глаголы в английском языке. Аналитические методы автоматического анализа словоформ представляют собой процедуры последовательного применения правил, построенных по грамматике рассматриваемого языка с введением списка слов-исключений.

Третий подход обладает большей универсальностью и не зависим от грамматики конкретного языка. В этом кроется как достоинство, так и недостаток, связанный со свойствами используемых для обучения текстовых массивов. К статистическим методам относятся:

- N-граммный метод [3];
- метод, основанный на скрытой марковской модели [4];
- методы, использующие кластерный анализ с введением специфичных функций расстояния между словоформами (метрик) [5].

Для использования статистических методов требуется наличие достаточно большого размеченного обучающего корпуса, получение которого является более трудоемкой задачей, чем актуализация морфологического словаря. При этом очевидно, что качество таких методов будет во многом определяться типом текстов, используемых для обучения и для последующей обработки. В свою очередь, аналитические методы лишены подобных недостатков.

Предлагаемый обобщенный метод автоматического анализа словоформ является обобщением алгоритмов, предложенных Портером [7] и развитых в рамках проекта snowball [8]. Реализации общего алгоритма автоматического анализа словоформ для конкретного языка имеют некоторые отличия в зависимости от типа морфологического строения языка [6].

1. Обобщенная схема автоматического анализа словоформ

Для каждого языка можно выделить набор минимальных элементов — аффиксов, используемых для образования различных форм слова. Различают несколько типов аффиксов: префиксы, словообразовательные и словоизменительные суффиксы, флексии. Аффиксы разбиваются на классы в соответствии с их назначением, и каждый класс аффиксов помещается в соответствующую структуру данных, чаще всего в качестве такой структуры выбирается бор (*trie*). Получается набор T_1, ..., T_N.

В грамматиках языков могут быть определены правила удвоения согласных, снятия или добавления диакритических знаков, явления фузии на стыках морфем. Такие правила формируют множество функций трансформации $F_т$.

Определим множество F_c функций проверки свойств анализируемой словоформы, накладывающих ограничения на применяемые к конкретной словоформе трансформации. Например, функция $f \in F_c$ проверки свойства может быть определена следующим образом: пусть R_1 — область, начинающаяся после первой согласной буквы, следующей за гласной (если такой согласной нет, то область пустая) и функция f возвращает результат проверки принадлежности аффикса вычисленной области R_1. Также функция $f \in F_c$ может обозначать принадлежности буквы заданному набору букв и т. д.

Анализируемую словоформу будем обозначать $w = w_0 = w_{01} \ldots w_{0n}$, длина словоформы $|w| = n$. Определим вспомогательную функцию *TestTrie* (T_j, w_i), предназначенную для проверки того, что словоформа оканчивается на один из аффиксов, определенных в боре T_j. Эта функция возвращает найденный в боре T_j аффикс словоформы w_i, или, если такого аффикса нет, то функция возвращает пустую строку ε.

Далее приводится описание обобщенного алгоритма анализа:

Вход: словоформа $w = w_0$.

Выход: основа w_i — словоформа после i-го преобразования.

1. Пользуясь грамматикой языка, определить боры $T_1, \ldots, T_N$.
2. На основе грамматики языка определить множества функций $F_т$ и F_c.
3. Применить набор функций трансформации $\{f_т^1, \ldots, f_т^l\} \subset F_т$, не зависящих от ограничений, накладываемых функциями из множества F_c, к подаваемой на вход словоформе $w = w_0$.
4. Определить множество аффиксов-кандидатов для обработки $A = \varnothing$.
5. Для всех боров T_j, если $TestTrie(T_j, w_i) = w_i^s \neq \varepsilon$:
   5.1. Добавить w_i^s к множеству A.
6. Если множество $A \neq \varnothing$:
   6.1. Выбрать аффикс $w_i^s \in A$ для отсечения в соответствии с ограничением, накладываемым функцией $f_r \in F_c$.
   6.2. Отсечь от w_i найденный аффикс w_i^s и в зависимости от выполнения одного из условий F_c применить одно из правил трансформаций $F_т$ и получить словоформу $w_i + 1$.
   6.3. $i = i + 1$; перейти к шагу 4.
7. Применить к w_i функции трансформации, обратные к тем, что были применены на шаге 3, и тем самым получить основу анализируемой словоформы.

Далее, в разделах 2, 3 и 4, приведено описание конкретных алгоритмов для различных языков. Важно отметить особенности реализации конкретных алгоритмов, связанные с характерными признаками морфологической структуры соответствующего языка.

Для аналитических языков характерно наличие малого числа морфем, участвующих в словоизменении. Поэтому в алгоритме анализа использование бора оказывается не эффективным и выбор делается в пользу массива.

В свою очередь, флективные языки богаты разного рода словоизменительными аффиксами, в силу чего, для повышения быстродействия алгоритма, аффиксы хранятся в структуре данных типа бор. Так же, так как в алгоритме, зачастую нужен поиск наиболее длинного аффикса словоформы, в боре хранится дополнительная служебная информация о длине аффикса.

Агглютинативные языки отличаются наличием большого числа аффиксов (причем они могут быть выражены и суффиксами, и префиксами). Также особенностью является определенная схема словоизменения (хотя имеется ряд исключений, в целом схему можно считать достаточно регулярной). Таким образом, наиболее подходящим решением для реализации алгоритмов анализа словоформ агглютинативных языков является использование конечных автоматов. При этом в рамках обобщенной схемы изменяется только смысл поиска по бору на поиск по исходящим из состояния дугам, что можно трактовать в терминах обобщенного алгоритма анализа словоформ.

2. Методы автоматического анализа словоформ языков с преобладанием аналитизма

В качестве наиболее яркого представителя языков с преобладанием аналитизма рассмотрим датский язык.

Для датского языка вводится понятие области R_1. Это область, начинающаяся после первой согласной буквы, следующей за гласной (если такой согласной нет, то область пустая).

В датском языке следующие буквы являются гласными: *a*, *e*, *i*, *o*, *u*, *y*, *æ*, *å*, *ø*, следующие — согласными: *b*, *c*, *d*, *f*, *g*, *h*, *j*, *k*, *l*, *m*, *n*, *p*, *q*, *r*, *s*, *t*, *v*, *w*, *x*, *z*.

Характерной особенностью словообразования имен существительных является прибавление суффикса "*s*" к концу слова. Поэтому в датском языке вводится понятие *S*-окончаний. Ими являются: *a*, *b*, *c*, *d*, *f*, *g*, *h*, *j*, *k*, *l*, *m*, *n*, *o*, *p*, *r*, *t*, *v*, *y*, *z*, *å*.

Таким образом, для датского языка в терминах обобщенного алгоритма определяются следующие функции $f \in F_c$:

- буква является *S*-окончанием;
- буква (или последовательность букв) располагается в области R_1.

В соответствии с обобщенной схемой алгоритма анализа будут сформированы списки аффиксов T_1, ..., T_6 (например, T_4 = *igst*).

Далее приводится описание алгоритма аналитического выделения основы.

Шаг 1:

Найти самый длинный аффикс из списка T_1 и, если он находится в области R_1, то удалить его.

Удалить суффикс *s*, если ему предшествует буква, определенная как *S*-окончание (при этом *S*-окончание может не входить в область R_1).

Шаг 2:

Найти самый длинный аффикс из списка T_3 и, если он находится в области R_1, то удалить последнюю букву.

Шаг 3:

Найти аффикс из списка T_4 и заменить его на *ig*.

Найти самый длинный аффикс из списка T_5 и, если он находится в области R_1, то удалить его и перейти к шагу 2.

Найти аффикс из списка T_6 и, если он находится в области R_1, то заменить его на аффикс *løs*.

Шаг 4:

Если слово оканчивается на удвоенную согласную в области R_1, то необходимо удалить последнюю согласную — этот шаг соответствует применению функций трансформации $f \in F_T$ в обобщенной схеме алгоритма анализа словоформы.

Полученный результат является искомой основой анализируемой словоформы.

Кроме этого, в датском языке определен ряд слов, не подлежащих обработке алгоритмом выделения основ. Обычно в роли этих слов выступают предлоги, местоимения и союзы.

3. Методы автоматического анализа словоформ языков с преобладанием флективных свойств

В качестве наиболее яркого представителя языков с преобладанием флективных свойств рассмотрим итальянский язык.

В итальянском языке определены следующие буквы с акцентом: *á*, *é*, *í*, *ó*, *ú*, *à*, *è*, *ì*, *ò*, *ù*.

В начале работы необходимо заменить все акуты на грависы и поместить *u* после *q*, а буквы *u*, *i* между гласными перевести в верхний регистр.

Гласными буквами являются: *a*, *e*, *i*, *o*, *u*, *à*, *è*, *ì*, *ò*, *ù*.

Определим область *RV* следующим образом: если в слове вторая буква согласная, то *RV* — это часть слова после следующей гласной; если первые две буквы гласные, то *RV* — часть слова после следующей согласной буквы, иначе, если первая — согласная, а вторая — гласная, то *RV* — это область слова после третьей буквы. Но в данных условиях *RV* не может начинаться в конце слова.

Определим R_1 как часть слова за первой негласной (согласной или заглавной гласной), следующей за гласной, или конец слова, если такой негласной нет. R_2 — часть слова за первой негласной, следующей за гласной, находящейся в R_1, или конец слова, если такой негласной нет. R_1 может содержать в себе *RV*, и *RV* может содержать в себе R_1.

В соответствии с обобщенной схемой алгоритма анализа будут сформированы списки аффиксов T_1, ..., T_{12} (например, T_4 = *logia*, *logie*).

Далее приведено описание алгоритма аналитического выделения основы(шаги алгоритма 0 и 1 выполняются всегда).

Шаг 0:

Найти самый длинный аффикс из списка T_1 в области *RV*, которому предшествует:

- *ando*, *endo* — в этом случае удалить аффикс.
- *ar*, *er*, *ir* — в этом случае заменить найденный аффикс на *e*.

Шаг 1:

Найти самый длинный аффикс из списка T_2 и, если он находится в области R_2, то удалить его.

Найти самый длинный аффикс из списка T_3 и, если он находится в области R_2, то удалить его. Если перед найденным аффиксом находится аффикс *ic*, то удалить его, если он находится в области R_2.

Найти самый длинный аффикс из списка T_4 и, если он находится в области R_2, то заменить его на *log*.

Найти самый длинный аффикс из списка T_5 и, если он находится в области R_2, то заменить его на *u*.

Найти самый длинный аффикс из списка T_6 и, если он находится в области R_2, то заменить его на *ente*.

Найти самый длинный аффикс из списка T_7 и, если он находится в области *RV*, то удалить его.

Найти самый длинный аффикс из списка T_8 и, если он находится в области R_1, то удалить его. Если найденному аффиксу предшествует аффикс *iv*, то удалить его, если он находится в области R_2 (если при этом *iv* предшествует аффикс *at*, то удалить его, если тот находится в области R_2); если суффиксу предшествует один из аффиксов *os*, *ic* или *abil*,то удалить его, если тот находится в области R_2.

Найти самый длинный аффикс из списка T_9 и, если он находится в области R_2, то удалить его. Если найденному аффиксу предшествует один из аффиксов *abil*, *ic* или *iv*, то удалить его, если тот находится в области R_2.

Найти самый длинный аффикс из списка T_{10} и, если он находится в области R_2, то удалить его. Если найденному аффиксу предшествует аффикс *at*, то удалить *at*, если он находится в области R_2 (если при этом *at* предшествует аффикс *ic*, то удалить *ic*, если тот находится в области R_2).

Если ни один из аффиксов не был удален на шаге 1, то перейти к шагу 2. Иначе — перейти к шагу 3а.

Шаг 2:

Найти самый длинный аффикс из списка T_{11}, и, если он находится в области *RV*, то удалить его.

Шаг 3а:

Удалить аффикс из списка T_{12}, если он находится в области *RV*, и предшествующую ему *i*, если она находится в области *RV*.

Шаг 3b:

Заменить последний аффикс *ch* (или *gh*) на *c* (или *g*), если он находится в области *RV*.

В заключение нужно вернуть *I* и *U* в нижний регистр — этот шаг соответствует применению функций трансформации $f \in F_T$ в обобщенной схеме алгоритма анализа словоформы.

Различие в методах, приведенных в разделах 2 и 3, заключается в выборе оптимальных эффективных структур данных для хранения списков аффиксов.

4. Методы автоматического анализа словоформ агглютинативных языков

В отличие от методов анализа аналитических и флективных языков для агглютинативных языков оказывается эффективным метод, в основу которого положено использование конечного автомата.

Наиболее характерным представителем агглютинативных языков является турецкий язык.

Так как турецкий язык относится к синтетическим агглютинативным языкам, то он обладает богатой и сложной морфологией. Слова в нем обычно состоят из основы и добавляемых к ней аффиксов, которых бывает, по крайней мере, два или три.

В турецком языке аффиксы имен существительных можно разделить на две группы:

- суффиксы имени существительного;
- именные глагольные суффиксы.

Слова, оканчивающиеся на именные глагольные суффиксы, могут выступать в предложении в роли глаголов.

Рассматриваемая морфологическая модель анализирует имена существительные и глаголы, образованные от имен существительных, так как они составляют основную часть турецкого языка.

В турецком языке аффиксы добавляются к основе с помощью конечного числа правил. Это позволяет представить морфологическую модель в виде конечного автомата. Состояние, помеченное символом "A", является стартовым. Терминальные состояния отмечены двойными окружностями. На дугах конечного автомата будем указывать номер суффикса (приведены далее в списках), который позволяет перейти из текущего состояния в следующее. При проходе по автомату, как только мы достигаем терминального состояния, то проверяем, есть ли переход из этого состояния, который можно осуществить при выполнении некоторого условия. Если переход совершить невозможно, то анализ словоформы завершается, иначе происходит переход в следующее состояние.

Боры, используемые в рамках обобщенной схемы автоматического анализа словоформ, формируются с учетом всех алломорфов суффиксов.

Имеется 19 суффиксов имени существительного и 15 именных глагольных суффиксов. Суффиксы рассматриваются с учетом алломорфов. Например, суффикс "-DAn" имеет четыре алломорфа: "dan", "den", "tan", "ten".

На рис. 1 и 2 представлены конечные автоматы для выделения основы имен существительных и образованных от имен существительных глаголов. Переход осуществляется при наличии соответствующего аффикса и выполнении заданных условий (например, условий гармонии гласных).

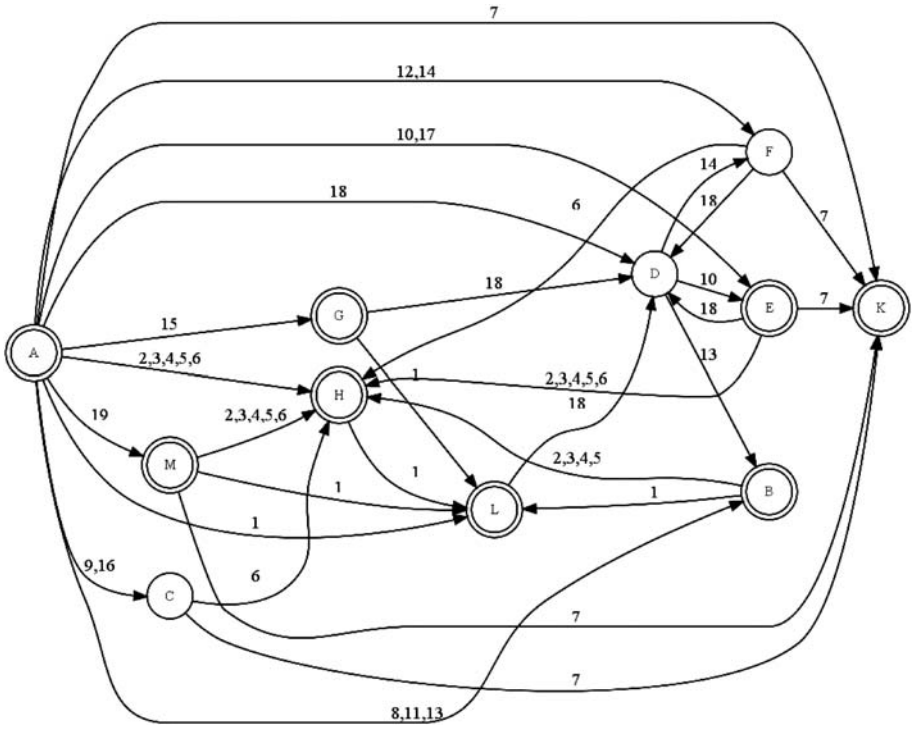


Рис. 1. Конечный автомат для отсечения суффиксов имени существительного

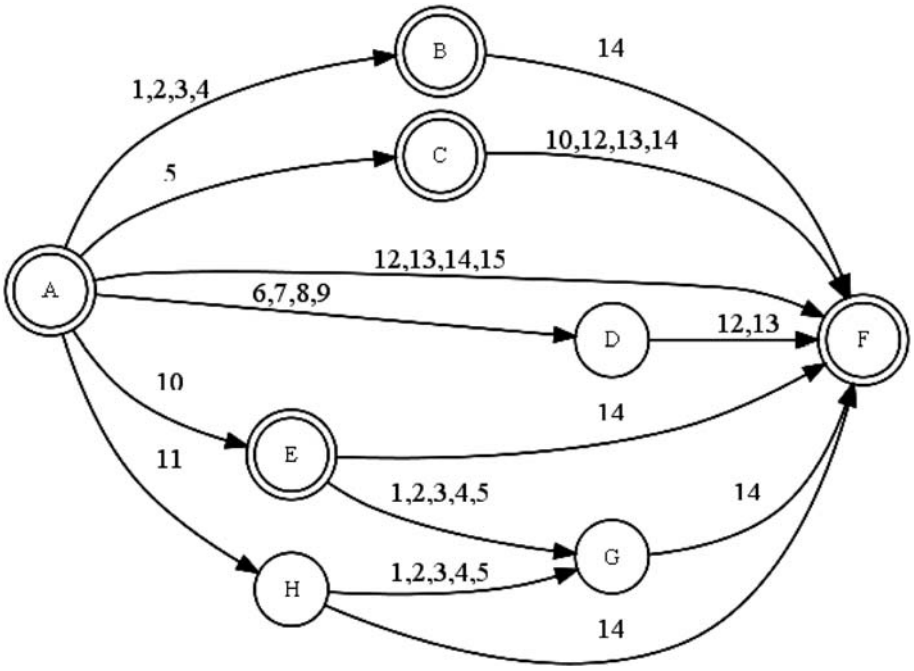


Рис. 2. Конечный автомат для отсечения именных глагольных суффиксов

Для турецкого языка характерно такое явление, как гармония гласных (сингармонизм). Поэтому при выделении основы алгоритм учитывает это явление следующим образом.

В турецком языке выделяют два параметра, которыми характеризуются гласные: переднеязычность и округленность.

Правило переднеязычности: в турецком языке гласные разделяются по принципу их произношения на передние ("e", "i", "ö", "ü") и задние ("a", "ı", "o", "u"). Правило гармонии гласных говорит о том, что в слове не могут идти подряд передняя и задняя гласные буквы.

Правило округленности: в турецком языке гласные разделяются по принципу произношения на скругленные ("o", "ö", "u", "ü") и неокругленные ("a", "e", "ı", "i"). Правило гармонии гласных в этом случае гласит, что если гласная в слоге неокругленная, то и следующая гласная должна также быть неокругленной. Если гласная в слоге скругленная, то следующей может идти одна из гласных "a", "e", "u", "ü".

Правила округленности и переднеязычности (гармонии гласных) соответствуют функциям проверки свойств словоформы $f \in F_c$.

Алгоритм аналитического выделения основы для турецкого языка начинает свою работу с того, что проверяет число слогов в анализируемой словоформе. Если слог один (одна гласная), то вся исходная словоформа считается основой.

Затем применяются процедуры поиска по представленным на рис. 1 и 2 конечным автоматам в целях получения предполагаемой основы. Подчеркнем, что процедуры применяются только для отыскания основ имен существительных и глаголов, образованных от имен существительных.

На заключительном шаге осуществляется постобработка полученной основы, что соответствует применению функций трансформации $f \in F_T$ в обобщенной схеме алгоритма анализа словоформы. В турецком языке слово не может оканчиваться на согласные "b", "c", "d" и "ğ". Однако, когда удаляется суффикс, начинающийся на гласную, то стоящие на конце "p", "ç", "t" и "k" преобразуются в "b", "c", "d" или "ğ" соответственно. Таким образом, на заключительном шаге нам нужно выполнить обратное преобразование.

Еще одной особенностью турецкого языка является тот факт, что если слово оканчивается на гласную, то перед суффиксом может появляться одна из согласных: "y", "n", "s". В тех случаях, когда такая согласная может появиться, представление суффикса дополнено опциональным значением согласной, указанным в круглых скобках ("-(y)Um"). При анализе это позволяет удалить появляющуюся гласную на конце предполагаемой основы.

По аналогии определяются суффиксы, перед которыми может появляться гласная буква. Например, в случае суффикса "-(U)mUz" между основой и суффиксом появляется гласная (одна из ряда "ı", "i", "u", "ü" в зависимости от гармонии гласных).

Заключение

Приведенные в разделах 2, 3 и 4 методы автоматического анализа словоформ были применены к другим языкам.

Так, метод из раздела 2 для языков с преобладанием аналитического типа словоизменения был применен к шведскому и английскому языкам. Метод из раздела 3 для языков с преобладанием флективного типа словоизменения был применен к испанскому и португальскому языкам. Для французского языка из-за флективности служебных слов (артикль, личное местоимение 3-го лица) приходится применять алгоритм из раздела 3, учитывая, таким образом, флективно-аналитический характер французского языка. В немецком языке имена существительные показывают склонность к аналитизму, в то время как глагол обладает выражен-

Таблица 1

Оценка качества алгоритмов

Язык	ICF
Английский	0,25
Датский	0,25
Шведский	0,25
Французский	0,52
Немецкий	0,31
Итальянский	0,58
Испанский	0,53
Португальский	0,55
Турецкий	0,35
Грузинский	0,46

Таблица 2

Тестирование производительности алгоритмов аналитического выделения основ

Язык	Размер коллекции, Мбайт	Скорость алгоритма, Мбайт/с		
		Windows 32 (Visual C++ 9.0)	Windows 64 (Visual C++ 9.0)	Linux 64 (GCC 4.4.1)
Английский	104	22,3	33,1	33,3
Датский	52	23,5	35	38,3
Шведский	56	25,1	33	37
Французский	71	6,3	7,4	9,4
Немецкий	51	10,2	12,9	13,3
Итальянский	57	11,1	13,1	13,4
Испанский	67	14,1	14,0	18,3
Португальский	52	9,9	11,1	14,8
Турецкий	51	13	17	20,4
Грузинский	53	20,4	24,2	35,4

ными флективными свойствами. Таким образом, немецкий язык проявляет свойства, аналогичные французскому языку. Метод из раздела 4 для языков с преобладанием агглютинативного типа словоизменения был применен к грузинскому языку.

Тестирование качества и производительности разработанных алгоритмов проводилось на корпусах текстов размерами от 50 до 100 Мбайт, в текстовом формате и содержащих от 50000 до 380000 уникальных словоупотреблений. В качестве тестовых данных были выбраны литературные произведения на различных европейских языках, а также коллекции коротких текстов, представляющие собой статьи из Интернета или записи с форумов и блогов.

Для оценки качества реализации мы использовали предложенный в работе [9] коэффициент сжатия индекса, который показывает, насколько коллекция уникальных основ меньше коллекции уникальных словоформ:

$$\mathrm{ICF} = \frac{N - S}{N},$$

где N — число уникальных словоупотреблений в тексте; S — число уникальных основ, полученных после применения процедуры анализа словоформ.

В табл. 1 приводятся значения коэффициента сжатия индекса для индоевропейских языков, для которых проводилось тестирование.

Если сопоставить результаты для английского языка (ICF = 0,25) с результатами, приведенными для алгоритма Портера в работе [9] (ICF = 0,17), то можно заметить, что предлагаемый нами алгоритм показывает лучшее качество и в большей степени подходит для построения систем автоматизированной обработки текстовой информации.

В табл. 2 приводятся результаты тестирования производительности разработанных алгоритмов. Переносимый код тестировался на трех платформах, и наилучшие показатели были достигнуты на 64-битной платформе. Полученные результаты показывают применимость предлагаемого алгоритма и методов его реализации для решения реальных задач автоматической обработки текстовой информации.

Список литературы

1. **Всеволодов А. В.** Компьютерная обработка лингвистических данных. М.: Наука; Флинта. 2007. 96 с.

2. **Леонтьева Н. Н.** Автоматическое понимание текстов: системы, модели, ресурсы. М.: Издат. центр "Академия", 2006. 304 с.

3. **James M., McNamee P.** Single N-gram stemming // Proceeding of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. 2003. P. 415—416.

4. **Melucci M., Orio N.** A novel method for stemmer generation based on hidden Markov models // Proceeding of the 12th international conference on Information and knowledge management. 2003. P. 131—138.

5. **Majumber P., Mitra M., Parui S. K., Kole G., Mitra P., Datta K.** YASS: Yet another suffix stripper // ACM Transactions on Information Systems. 2007. Vol. 25. N 4. P. 18—38.

6. **Бенвенист Э.** Классификация языков //Новое в лингвистике. 1963. Вып. 3. С. 36—59.

7. **Porter M. F.** An algorithm for suffix stripping // Program. 1980. Vol. 14. N 3. P. 130—137.

8. **Snowball.** URL: http://snowball.tartarus.org (дата обращения: 08.07.2010).

9. **Frakes W. B., Fox C. I.** Strength and Similarity of Affix Removal Stemming Algorithms // SIGIR Forum 37. 2003. P. 26—30.

УДК 004.925.3

В. А. Бобков, д-р техн. наук, зав. лаб.,
С. В. Мельман, мл. науч. сотр.
Институт автоматики
и процессов управления ДВО РАН
E-mail:gruzd@dvo.ru

Параллельная трассировка октантных деревьев на языке CUDA[1]

Рассмотрена реализация на языке CUDA для графического процессора параллельного алгоритма трассировки лучей в октантном дереве, описывающем воксельную сцену. Проведен анализ вычислительной эффективности предложенного алгоритма.

Ключевые слова: *параллельный алгоритм, GPU общего назначения, графический процессор, трассировка лучей, октантные деревья*

Введение

В последнее время наметилась тенденция к активному использованию стандартных графических процессоров (GPU) для ускорения вычислений в различных тематических приложениях. Ускорение достигается за счет присущей GPU многопроцессорной архитектуры и наличию системных программных средств для организации параллельной обработки данных. На сегодняшний день в качестве высокоуровневых средств параллельного программирования для них используются технологии CUDA, OpenCL, OpenMP и др., которые в основном ориентированы на реализацию универсального параллелизма в отличие от ранее разработанных и активно используемых средств традиционного программирования, таких как CG, шейдерные языки (GLSL, HLSL и др.). Следует при этом отметить, что не только разработка новых, но и адаптация существующих алгоритмов к условиям работы в программно-архитектурной среде GPU является задачей нетривиальной, требующей новых подходов к организации данных и их обработке.

[1] Работа выполнена при финансовой поддержке проектов ДВО РАН, соответствующих направлениям Программы № 2 Президиума РАН.

В данной статье решается задача эффективной алгоритмической реализации параллельной трассировки октантных деревьев на GPU средствами CUDA применительно к подходу к визуализации воксельных сцен, предложенному авторами ранее в работе [1]. В упомянутой работе были предложены октантная структура данных и оригинальный целочисленный алгоритм ее трассировки, предназначенные для последовательной обработки данных на центральном процессоре. В настоящей работе предлагается два варианта параллельной реализации трассировки на GPU. Первый непосредственно основывается на алгоритме, описанном в работе [1]. Второй разработан с учетом особенностей программирования на языке CUDA и архитектуры GPU. Вычислительные эксперименты были проведены на следующем оборудовании: GeForce 9600GT, 512Mb и CPU Intel Core 2 Duo 3GHz, 2Gb.

Вопросам генерации, редактирования и трассировки структур данных на GPU было уделено много внимания уже на начальном этапе использования графических процессоров как при обработке графических данных, так и для общих вычислений (GPGPU). В качестве структур рассматривались главным образом 3D-решетки и разного рода деревья с соответствующими методами/алгоритмами работы с ними. Хороший обзор по визуализации объемов с применением GPU можно найти, например, в работах [2, 3]. Типичными методами ускорения трассировки лучей (*raycasting*) во многих реализациях на GPU (например, [4, 5]) стали: а) раннее завершение трассировки луча по условию (пересечение с поверхностью или превышение порога по прозрачности); б) сворачивание/игнорирование пустого пространства.

В ряде работ предлагались схемы эффективной работы с большими объемами данных, учитывающие ограниченные объемы аппаратной графической памяти. В основном, эти схемы базируются на применении компрессии данных, мультиразрешении и так называемых "внеядерных" методов (*out-of-core*), предполагающих размещение очень больших объемов данных во внешней памяти компьютера [6—9]. Например, в работах [8, 9] акцент делается на обеспечение высоких скоростей рендеринга при работе с большими объемами воксельных данных за счет эффективных механизмов передачи актуальных данных из CPU в GPU. На CPU используются октантные структуры, в листьях ко-

торых возможна организация данных в виде 3D-решеток. Эффект достигается за счет выборки и передачи на GPU только требуемых для текущего кадра визуализации данных. При этом учитывается и различная степень детализации для разных участков сцены и невидимость некоторых фрагментов поверхностей для заданной точки наблюдения. Характерным для упомянутых работ является применение наряду с шейдерными программами программ на языке CG, который, относясь к традиционной парадигме программирования, не предназначен для реализации параллелизма, потенциально присущего GPU.

Упомянутые выше языки параллельного программирования CUDA и др. начали применяться для создания параллельных программ на GPU в последние 2—3 года. Однако в текущих версиях языка CUDA существуют ограничения, препятствующие извлечению максимальной эффективности из параллельной архитектуры. Наиболее узкими местами являются реализация возможных многочисленных ветвлений в алгоритме, отсутствие рекурсий в языке и медленная работа с оперативной памятью. Поскольку современные видеокарты достаточно быстро прогрессируют, актуальность преодоления ограничений по объему памяти постепенно снижается, и на первый план выходит задача эффективной реализации параллелизма для повышения скорости расчетов и рендеринга. В настоящей статье акцент делается именно на эффективном распараллеливании вычислений применительно к трассировке октантных деревьев для визуализации воксельных сцен. Из близких работ в доступной авторам литературе можно отметить, например, работы [10, 11]. В первой работе предлагается оптимизированный и простой в реализации подход к трассировке лучей на GPU, который учитывает когерентность лучей и специфику используемого аппаратного обеспечения. Высокая производительность обеспечивается за счет эмпирической настройки программы на используемое оборудование. Автор второй работы [11] также отмечает простоту своей GPU-реализации применительно к рендерингу больших наборов точек (частиц), организованных в октоструктуру. Особенностью построенного октантного дерева является его компактность с возможностью задания нескольких уровней детализации, что, в свою очередь, позволяет управлять соотношением "степень детализации—скорость рендеринга".

Последующий текст статьи организован следующим образом. В разделе 1 описаны октантная структура данных и целочисленный алгоритм трассировки октантных деревьев, изначально предназначенные для последовательного выполнения. В разделе 2 описана реализация параллельной трассировки октантных деревьев на языке CUDA. В разделе 3 приведены результаты вычислительных экспериментов с оценкой эффективности реализации параллелизма на CUDA.

1. Трассировка октантных деревьев для однопроцессорной обработки

Эффективный целочисленный алгоритм трассировки лучей на октантных деревьях для однопроцессорной конфигурации был предложен авторами в работе [1]. Реализованная октантная структура ориентирована на визуализацию видимой поверхности сцены (так называемой "оболочки"), образуемой граничными вокселями. Корневой узел октантного дерева содержит информацию о размерах сцены и положении в мировой системе координат, а также восемь ссылок на узлы следующего уровня. Узлам дерева соответствуют параллелепипеды — боксы в пространстве сцены. Узлы предпоследнего уровня содержат ссылки на граничные воксели с информацией о базовом цвете вокселя и нормалью для расчета освещения. Подобная структура легко реализуется на языке C++ для последовательного выполнения на центральном процессоре, не требует больших объемов оперативной памяти и при интерактивном рендеринге дополнительных вычислений. В алгоритме используется традиционный метод трассировки октантных деревьев "сверху вниз" от корневого узла к граничным вокселям с горизонтальным поиском пересеченных лучом подбоксов в рамках родительского бокса. Координаты источника освещения и всех лучей трассировки пересчитываются в систему координат куба с длиной ребра 2^{14}. Для ускорения вычислений применен быстрый табличный поиск пересеченных подбоксов в родительском боксе. На первом этапе работы алгоритма вычисляется главное направление и точки входа и выхода луча из бокса. Суть алгоритма заключается в том, что, зная наклон луча, точку пересечения луча и средней линии трех проекций узла (рис. 1, 2), можно простым выбором из таблицы получить "байт состояния" узла.

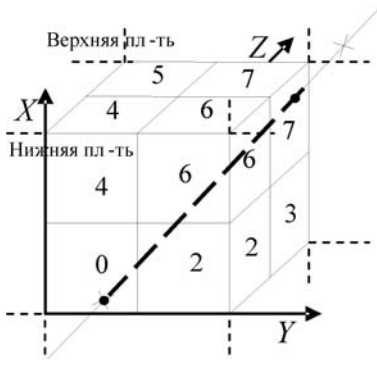


Рис. 1. Нумерация подбоксов в боксе

Рис. 2. Проекция бокса на плоскость *YZ*

"Байт состояния" — это список пересеченных лучом узлов более низкого уровня. Далее, по "байту состояния" и наклону луча из таблицы выбирается последовательность пересеченных подбоксов (узлов) в порядке следования от ближнего к дальнему. Переход от родительского узла к дочернему узлу выполняется операциями целочисленного сложения и умножения на 2. Рекурсивно перебирая пересеченные узлы, находим граничный воксель, вычисляем его освещенность и закрашиваем соответствующий лучу пиксель.

2. Параллельная трассировка

Особенности программирования на языке CUDA. Многопоточный мультипроцессорный чип nVidia, специализированный для вычислений с плавающей точкой одинарной точности с помощью технологии CUDA, позволяет реализовывать алгоритмы на языке программирования C. Известно, что язык CUDA имеет ряд ограничений, специфических для графических ускорителей на базе nVidia:

- CUDA не полностью совместим с C++, поэтому необходима адаптация переносимых программ;
- CUDA — для специализированного "железа", поэтому нужна серьезная переработка существующих алгоритмов для достижения наилучшего по скорости вычислений результата.

Описание структуры октантного дерева для алгоритма на CUDA. Анализ особенностей программирования на языке CUDA привел к следующим требованиям и выводам применительно к реализации рассмотренных выше алгоритма трассировки и октантной структуры данных на CUDA.

1. Для повышения эффективности работы со структурой необходима такая организация хранения данных, чтобы каждое поддерево представлялось непрерывным участком памяти.

2. Необходима минимизация и оптимизация обращений к памяти с учетом того, что доступ к глобальной памяти имеет большие задержки и не кэшируется, но доступ к последовательно расположенным ячейкам памяти параллельными потоками выполняется за одну транзакцию.

3. Необходима минимизация ветвлений, так как эффективность графического процессора в случае ветвления снижается.

4. Необходим отказ от рекурсии, поскольку CUDA не поддерживает рекурсию.

Первое достигается традиционным приемом разделения структурной информации и воксельных данных. В структуре дерева при достижении граничного вокселя делается запись со ссылкой на воксель в массиве вокселей. Таким образом, общая длина массива со структурой дерева существенно уменьшается и, соответственно, уменьшается "расстояние" между соседними поддеревьями, что может быть использовано при кэшировании. Массивы в глобальной памяти не кэшируются, но структуру дерева можно хранить и в текстурной памяти, а она уже кэшируется.

Структура октодерева состоит из последовательно стоящих друг за другом блоков с описанием промежуточного узла (рис. 3). Каждый узел содержит 8 индексов узлов более низкого уровня. Если индекс равен 0, то соответствующий подбокс пуст. А узел граничного вокселя содержит индекс вокселя в массиве вокселей. Для однозначной идентификации узлов, относящихся к граничным вокселям, их индексы хранятся со знаком минус.

Минимизация обращений к памяти достигается заменой считывания данных на их вычисление на лету. Например, текущие координаты бокса, при переходе к подбоксам, эффективнее не записывать в стек, а сохранить только номер нового подбокса, и при необходимости вернуться на уровень назад. Зная номер, можно вычислить координаты родительского бокса простым сложением и умножением на 2. Ускорения обращений к памяти можно добиться за счет последовательного доступа к соседним

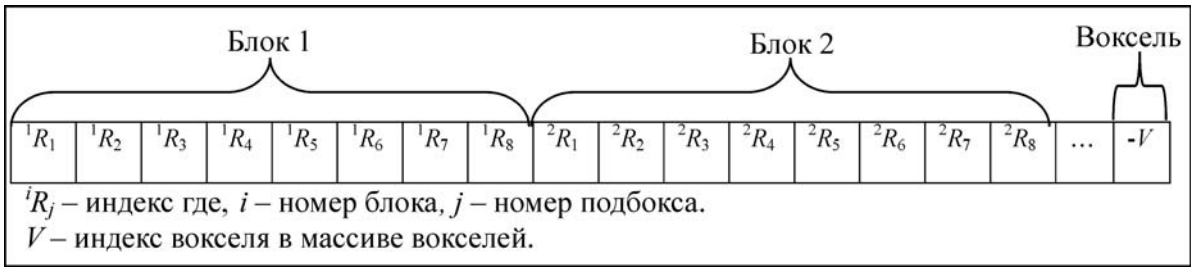


Рис. 3. Структура октодерева. Каждый блок содержит 8 индексов по 4 байта

ячейкам данных в глобальной памяти. Например, организовать вычисления потоков таким образом, чтобы пучок трассируемых лучей был максимально "компактным" (когерентным), тогда ветвления минимизируются, и кэш работает эффективнее.

Описание алгоритма 1. При реализации упомянутого выше целочисленного алгоритма использовались базовые возможности языка Си, поэтому этот алгоритм без существенного редактирования был скомпилирован для выполнения на графическом процессоре. Изменениям подверглась лишь та часть, которая касалась работы с октантной структурой и рекурсией. Используемые в алгоритме таблицы были помещены в "константную" память графического ускорителя, которая кэшируется. Для того чтобы "обойти" известные узкие места в архитектуре графического процессора, — чтение/запись данных в память и ветвления, — в алгоритм были внесены следующие изменения. Были построены небольшие по объему части кода, обрабатывающие каждый из конкретных случаев, учитывающих все возможные варианты пересечения луча и бокса, варианты главного направления, варианты подбоксов. На центральном процессоре алгоритм, построенный на использовании таблиц, позволял заменить большое количество вычислений заранее вычисленными табличными значениями. На графическом процессоре именно это и стало узким местом, поскольку требовалось большое число ветвлений для учета всех возможных вариантов, а также хранение промежуточных вычислений в оперативной памяти. В результате алгоритм № 1, сконструированный для применения на графическом процессоре, показал низкую эффективность, сопоставимую по времени вычислений с центральным процессором. Поэтому потребовалась серьезная переработка базового алгоритма, направленная на более высокую степень эффективности реализации параллелизма на графическом процессоре.

Описание алгоритма 2. Итак, из базового алгоритма за основу берутся понятия главного направления и "состояние проекций луча", которые позволяют нам однозначно определять последовательность перебора подбоксов в родительском боксе. Также используется перевод луча в новую систему координат (СК) на каждом шаге, что позволяет построить все вычисления для узла константного размера и избежать дополнительных обращений к памяти графического ускорителя. Алгоритм должен быть сконструирован таким образом, чтобы стек хранил минимальное число данных для перехода между уровнями дерева. Для однозначного определения состояния трассировки на промежуточных уровнях достаточно хранить ссылку на поддерево уровнем выше и номер узла, в который алгоритм спускается для поиска дальнейших пересечений. Переходы из одной СК в другую при смене уровня дерева оказалось эффективнее вычислять на лету, поскольку вычисления над регистровыми переменными для графического процессора выполняются быстрее, чем обращение к оперативной памяти для чтения предыдущего состояния. Отметим, что требование отказа от рекурсий было выполнено за счет преобразования исходной рекурсивной структуры программы в последовательную, с использованием условных переходов (ветвлений). Схематически алгоритм трассировки октантного дерева можно представить следующим образом:

```
для каждого луча
    перевести луч в СК корневого узла
начало:
    получить следующее поддерево
        если текущее поддерево воксель
            обработать воксель
            закончить
продолжение:
    для каждого непустого поддерева
        перевести луч в СК непустого поддерева
        если поддерево пересечено лучом
            сохранить в стек текущее состояние
            перейти в начало
    если стек не пуст
        взять из стека предыдущее состояние
        перевести луч в СК родительского поддерева
        перейти в продолжение
    если стек пуст
ничего не пересекли
```

Для запуска ядра трассировки нужен массив структуры дерева, информация о "камере" и адрес выходного массива для найденных, пересеченных лучами вокселей.

Блок задач (потоков) выполняется на мультипроцессоре частями или пулами, называемыми *warp*. Размер *warp* на текущий момент в видеокартах с поддержкой CUDA равен 32 потокам.

Разбиение на потоки делается таким образом, чтобы один блок потоков обрабатывал $4 \times 16 = 64$ лучей, — это оптимум между компактностью лучей и эффективностью распределения по *warp*. При обработке *warp* на мультипроцессоре за один проход обрабатывается 8 строк по 4 пикселя в выходном изображении. Это хорошо соотносится с принципом эффективного использования подсистемы записи в оперативную память, т. е. за одно обращение к памяти можно записать последовательный блок памяти, кратный 16. Общее число потоков (4×16) было выбрано из соображений полного использования доступного числа регистров на один мультипроцессор. Каждый поток вычисляет параметры луча для трассировки, сканирует дерево до пересечения с ближайшим граничным вокселем и записывает его номер в выходной массив или выполняет вычисление цвета пикселя с учетом освещения.

3. Результаты вычислительных экспериментов

Для оценки эффективности "параллельной" реализации базового алгоритма трассировки были проведены две серии экспериментов с использованием графической платы GeForce 9600GT, 512 Mb и CPU Intel Core 2 Duo 3 GHz, 2 Gb. Использовались сцены с глубиной дерева до 9 уровней. Например, сцена Bucky (рис. 4, 5), имеет 8 уровней и 292 106 граничных вокселей. Линейная структура дерева занимает 10 299 780 байт. Объем информации о граничных вокселях — 4 673 696 байт в памяти графического ускорителя. Каждый воксель представлен базовым цветом (4 байта) и нормалью (12 байт).

В первой серии экспериментов (рис. 4 и 5) было выявлено очевидное преимущество в скорости на GPU алгоритма № 2 по отношению к алгоритму № 1 (табл. 1, 2). Этот ожидаемый результат объясняется более последовательным учетом в реализации алгоритма № 2 ограничений и особенностей параллельного программирования для GPU на языке CUDA. Также очевидно преимущество алгоритма № 1 по отношению к алгоритму № 2 на CPU. Поэтому во второй серии экспериментов

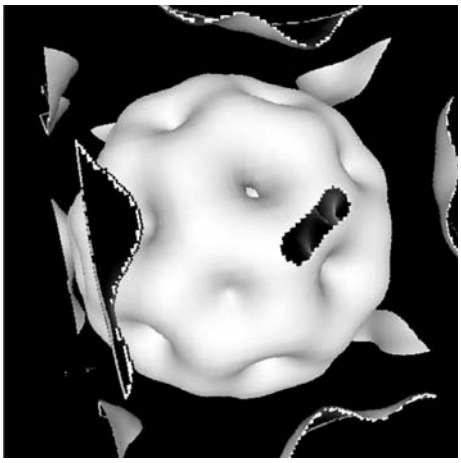


Рис. 4. Сцена "Bucky". Объект общим планом

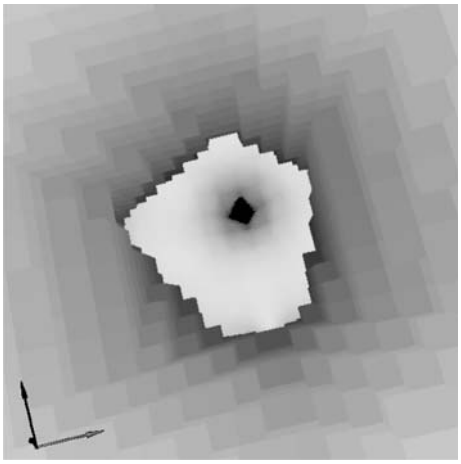


Рис. 5. Сцена "Bucky". Камера приближена к объекту

Таблица 1

Время рендеринга (мс), рис. 4

Рис. 4	CPU	GPU
Алгоритм № 1	222	314
Алгоритм № 2	1027	174

Таблица 2

Время рендеринга (мс), рис. 5

Рис. 5	CPU	GPU
Алгоритм № 1	222	74
Алгоритм № 2	1050	35

Таблица 3

Время рендеринга (мс) для сцен рис. 6—8 под разными ракурсами

Сцена (Глубина)	Ракурс	Число граничных вокселей	Алгоритм № 1 (CPU) 1 ядро	Алгоритм № 1 (CPU) 2 ядра	Алгоритм № 2 (GPU)
Рис. 6 ($L = 8$)	*а*	292106	222	111	174
	б	292106	315	159	140
	в	292106	222	111	35
Рис. 7 ($L = 9$)	*а*	883993	236	120	140
	б	883993	310	170	105
Рис. 8 ($L = 9$)	*а*	821420	217	106	170
	б	821420	353	167	222
	в	821420	250	123	180
	г	821420	360	230	170

(рис. 6—8) сравнивались алгоритмы № 2 на GPU и алгоритм № 1 на CPU (табл. 3).

В этих экспериментах выявилась зависимость скорости рендеринга от сложности и выбранного ракурса сцены для "параллельного" алгоритма на GPU. При приближении камеры к сцене скорость обработки растет. Для случая CPU такая зависимость практически отсутствует. Это связано с тем, что основной объем вычислений при трассировке лучей сосредоточен в расчете начальных параметров трассировки, а спуск по дереву от корневого узла к листьям — процедура вычислительно несложная, но требующая большого числа проверок и ветвлений. В результате, для CPU скорость рендеринга определяется, главным образом, числом лучей, т. е. расчет начальных параметров всех лучей является более затратным, чем непосредственная трассировка лучей до граничных вокселей. Для GPU, напротив, расчет начальных параметров трассировки для всех лучей — процедура малозатратная, поскольку хорошо соответствует его многопоточной архитектуре. В то же время проход от родительского бокса к подбоксам, требующий проверок и ветвлений, является более затратным. Ветвление на CUDA, в силу специфики GPU, реализовано таким образом, что все лучи одного *warp* ждут, пока

Рис. 6. Сцена "Bucky":
а — ракурс 1; *б* — ракурс 2; *в* — ракурс 3

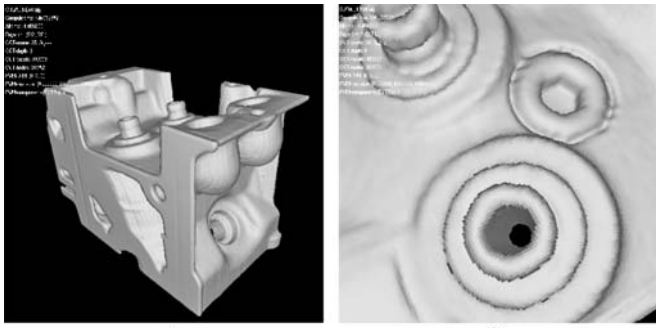


Рис. 7. Сцена "Engine":
а — ракурс 1; *б* — ракурс 2

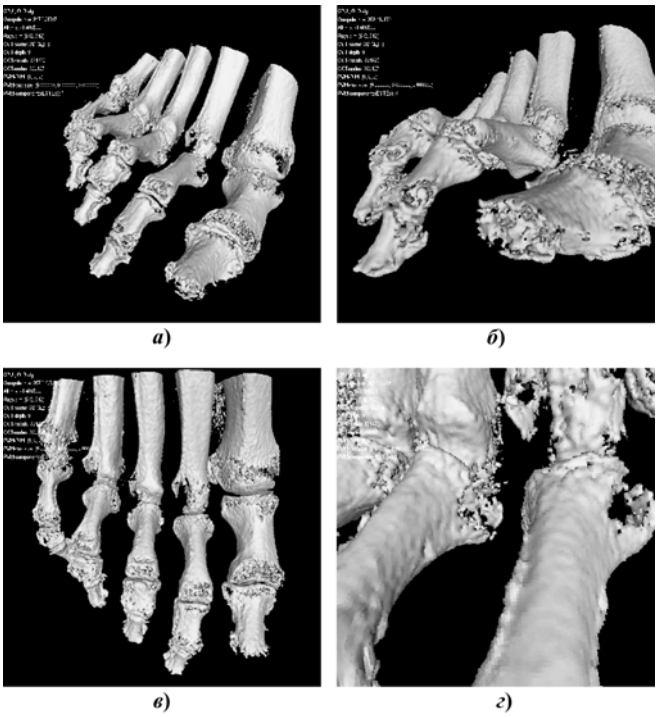


Рис. 8. Сцена "Foot":
а — ракурс 1; *б* — ракурс 2; *в* — ракурс 3; *г* — ракурс 4

закончится обработка самого "сложного" луча. То есть для GPU наиболее затратной вычислительной частью является проход по узлам до граничных вокселей, где сконцентрированы все ветвления алгоритма. Этим объясняется сложная зависимость скорости рендеринга на GPU от глубины дерева (глубже дерево — больше ветвлений), от сложности и ракурса (больше проверяемых подбоксов до пересечения луча с объектом — больше ветвлений). Другими словами, при высокой когерентности лучей (много лучей попадает в один или соседние воксели) эффективность алгоритма для GPU повышается за счет меньшего числа условных ветвлений.

Следует также одновременно отметить и определенные преимущества исходного базового алгоритма для CPU:

- хорошая производительность, сравнимая с производительностью параллельных реализаций на GPU, независимо от сложности и ракурса сцены;
- эффективность распараллеливания применительно к традиционной многопроцессорной архитектуре.

Заключение

В работе предложена реализация на GPU средствами языка CUDA параллельного алгоритма трассировки лучей в октантном дереве, описывающем воксельную сцену. Алгоритм сконструирован с учетом как особенностей программирования на языке CUDA, так и специфики организации параллельных вычислений на архитектуре графической платы. Вычислительные эксперименты с графической платой GeForce 9600GT показали, что за счет параллельной обработки на GPU возможно ускорение в несколько раз по отношению к производительности на CPU. При этом максимальная эффективность распараллеливания (более чем на порядок) достигается в ситуациях с высокой когерентностью лучей трассировки (приближение к сцене, неглубокие октантные деревья). Однако узким местом

предложенного алгоритма, ограничивающим получение более весомого ускорения, являются ветвления, необходимые для трассировки октантного дерева. В перспективе можно ожидать появления новых архитектур (например FERMI), более адаптированных к циклам и условным ветвлениям, что позволит существенно увеличить эффективность подобного рода алгоритмов. Получаемый за счет графического ускорителя потенциал "быстрых" вычислений может рассматриваться не только как альтернатива обработки данных на CPU, но и как дополнительная возможность организации комбинированной обработки данных CPU + GPU в приложениях с использованием октантной структуры данных.

Список литературы

1. **Бобков В. А., Роньшин Ю. И., Мельман С. В.** Визуализация воксельных сцен // Информационные технологии. 2005. № 6. С. 16—19.

2. **Engel K., Hadwiger M., Kniss J., Rezk-Salama C., Weiskopf D.** Real-time Volume Fraphics. AK-Peters 2006.

3. **Lefohn A. E.** Glift: Generic Data Structures for Graphics Hardware // Dissertation Submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science in the Office of Graduate Studies, of the University of California, 2006.

4. **Kruger J., Westermann R.** Acceleration techniques for GPU-based volume rendering // Proc. of IEEE Visualization. 2003. P. 287—292.

5. **Scharsach H.** Advanced GPU raycasting // In Central European Seminar on Computer Graphics. 2005. P. 69—76.

6. **Vollrath J. E., Schafhitzel T., Ertl T.** Employing complex GPU data structures for the interactive visualization of adaptive mesh refinement data // Eurographics. IEEE VGTC Workshop on Volume Graphics. 2006. P. 55—58.

7. **Boada I., Navazo I., Scopigno R.** Multiresolution volume visualization with a texture-based octree // The Visual Computer. 2001. N 17(3). P. 185—197.

8. **Gobbetti E., Marton F., Guitian J. A. I.** A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets // The Visual Computer. July 2008. Vol. 24, N 7—9. P. 797—806.

9. **Grassin C., Neyret F., Lefebvre S., Eisemann E., Voxels G.** Ray-guided streaming for efficient and detailed voxel rendering // Symposium on Interactive 3D Graphics. Proc. of the 2009 symposium on Interactive 3D graphics and games. 2009. P. 15—22.

10. **Budge B. C., Anderson J. C., Garth C., Joy K. I.** A straightforward CUDA implementation for interactive ray-tracing // Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium. 9—10 Aug. 2008. P. 178.

11. **Knoll A.** Ray traversal of octree point clouds on the GPU // Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium. 9—10 Aug. 2008. P. 182.

УДК 004.4'41

С. В. Полуян, аспирант,
Южный федеральный университет,
Ростов-на-Дону,
e-mail: steka@front.ru

Уточнение графа информационных связей с помощью анализа псевдонимов

Рассмотрено создание алгоритма, позволяющего на этапе компиляции построить граф информационных связей с учетом информации о псевдонимах переменных. Этот алгоритм программно реализован в "Диалоговом высокоуровневом оптимизирующем распараллеливателе программ".

Ключевые слова: *граф информационных связей, анализ псевдонимов, распараллеливание*

Введение

Данная работа посвящена уточнению информационных зависимостей программ на графе информационных связей [1, §1.2] за счет учета информации о псевдонимах переменных. Результаты этой работы используются в "Диалоговом высокоуровневом оптимизирующем распараллеливателе" (ДВОР) [2].

Под зависимостью в данной работе понимается так называемая *информационная зависимость по памяти* (memory-based dependence) [3, § 1.1.3], т. е. такая зависимость, которая существует между вхождениями переменных, если они обращаются к одной и той же ячейке памяти. Для получения информации о зависимостях используются специальные методы, такие как НОД-тест и неравенства Банержи [4, §3.3.3], Омега-тест [3, §1.3.3], методы П. Фотрье (P. Feautrier) [5] и В. В. Воеводина [6, §4.2]. Эти методы определяют зависимости между вхождениями переменных при определенных значениях индексных выражений, однако они не учитывают зависимости, обусловленные наличием в программах *псевдонимов* (aliases) [7, §1.6.7].

Псевдонимы — это переменные, являющиеся разными именами одной и той же ячейки памяти. Псевдонимы характерны для таких языков программирования, как C, Fortran 90, Pascal и других, в которых существуют указатели [8, §5.10].

Для сбора информации о псевдонимах в ДВОР реализован алгоритм *анализа псевдонимов* (alias analysis) [9, §10]. Этот анализ используется в современных компиляторах, обфускаторах [10] и тестогенераторах, но лишь для уточнения анализа потоков данных (data-flow analysis) [7, §9.2]. Так, в наборе компиляторов GCC [11] анализ псевдонимов выполняется на основе SSA-формы [9, §8.11], а его

результаты используются в анализе потоков данных перед выполнением оптимизирующих преобразований. Или, например, в анализе запутывающих преобразований программ А. В. Чернова [10] анализ псевдонимов применяется для корректного запутывания графа потока управления программы.

Особенностью данной работы является использование результатов анализа псевдонимов для уточнения графа информационных связей, а не анализа потоков данных. Задача уточнения графа информационных связей имеет большое значение в ДВОР, так как этот граф используется в распараллеливающих преобразованиях программ, таких как перестановка фрагментов кода, переименование переменных, разбиение циклов и так далее. В свою очередь анализ псевдонимов в ДВОР также имеет свои особенности:

- позволяет получить результаты для исходной, а не преобразованной (например к SSA-форме) программы, так как для анализа информационных зависимостей нужна информация о псевдонимах для исходной программы;
- является универсальным для разных языков программирования, так как реализован на мультиязыковом внутреннем представлении, называемом в ДВОР Reprise [12], в узлах которого содержатся высокоуровневые конструкции языков программирования, а его структура не зависит от входного языка;
- реализован так, чтобы информация о псевдонимах легко обновлялась в процессе использования в ДВОР трансформирующих программу преобразований;
- хранит информацию о псевдонимах в хэш-таблице так, что в момент анализа информационных зависимостей не нужно выполнять никаких дополнительных вычислений.

Методика уточнения графа информационных связей

Графом информационных связей называется ориентированный граф, вершинами которого являются вхождения переменных, а между вхождениями *u* и *v* существует дуга, направленная от *u* к *v*, если вхождение *v* зависит от вхождения *u*. В этом определении вхождением переменной называется всякое появление этой переменной в тексте программы. Причем, если при обращении к переменной меняется состояние ячейки памяти, то такое вхождение называется генератором, остальные вхождения называются использованиями. Таким образом, дуги графа информационных связей бывают четырех типов в зависимости от типов инцидентных им вершин:

1) если *u* является генератором, а *v* — использованием, то зависимость называется *потоковой зависимостью* (flow dependence);

2) если *u* является использованием, а *v* — генератором, то зависимость называется *антизависимостью* (antidependence);

3) если *u* и *v* являются генераторами, то зависимость называется *выходной зависимостью* (output dependence);

4) если *u* и *v* являются генераторами, то зависимость называется *входной зависимостью* (input dependence).

Перед уточнением графа информационных связей в ДВОР проводится анализ псевдонимов. Он позволяет для каждого узла графа, являющегося вхождением указателя, сформировать множество его возможных значений.

В процессе уточнения графа информационных связей к нему добавляются новые дуги:

- соединяющие вхождение разыменнованного указателя с вхождением переменной, если существует зависимость между этим вхождением переменной и возможным значением указателя;
- соединяющие вхождения разыменнованных указателей между собой, если между возможными значениями указателей существуют зависимости.

В качестве примера рассмотрим следующий фрагмент программы на языке С:

```
int i, A[102], *p;
...
p = &A[0];
for (i = 0; i < 100; i = i + 1)
  A[i + 1] = *(p + i + 2) + 1;
```

Граф информационных связей, автоматически построенный в ДВОР с помощью неравенств Банержи и НОД-теста без учета информации о псевдонимах, для этого фрагмента программы имеет вид, представленный на рис. 1.

Как видно, на графе нет дуг информационных зависимостей, мешающих распараллеливанию этого фрагмента кода, но, добавив к нему информацию о псевдонимах, получаем следующий граф (рис. 2).

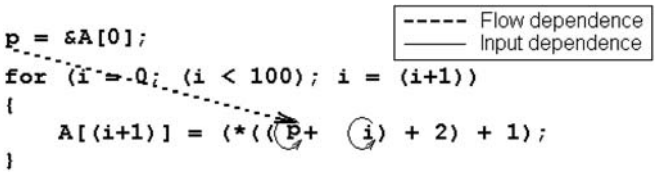


Рис. 1. Граф информационных связей без учета псевдонимов

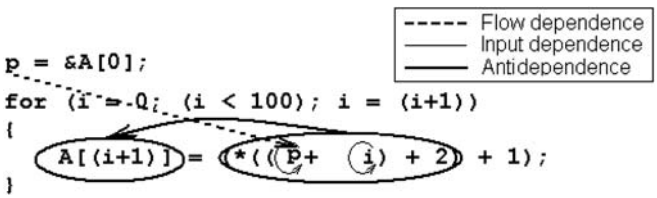


Рис. 2. Уточненный граф информационных связей

Можно видеть, что на графе появилась новая дуга антизависимости, ведущая от использования `*(p + i + 2)` к генератору `A[(i + 1)]`, мешающая распараллеливанию данного фрагмента кода. Таким образом, не обнаружив этой дуги при простом анализе информационных зависимостей, существует опасность ошибочно применить распараллеливающие преобразования.

Анализ псевдонимов

В "Диалоговом высокоуровневом оптимизирующем распараллеливателе программ" реализован анализ псевдонимов, основанный как на анализе указателей (*points-to analysis*) [13], так и на анализе типов переменных (*type-based alias analysis*) [14]. Следует отметить, что зачастую анализом псевдонимов называется лишь один из этих двух видов анализов [7], однако в данной работе под анализом псевдонимов понимается совокупность анализа указателей и анализа типов.

Для сбора информации о псевдонимах используется понятие *абстрактной ячейки памяти* [15]. Абстрактная ячейка памяти — это поименованная область памяти, к которой возможен доступ из программы. В качестве представления абстрактной ячейки памяти была выбрана пара:

<Имя, Индекс>,

где "Имя" — имя переменной, а "Индекс" — индекс в массиве.

На основе понятия абстрактной ячейки памяти реализован механизм анализа псевдонимов при использовании арифметических операций с указателями. Хотя арифметические операции с указателями имеют смысл в основном при работе со структурными данными, реализован механизм, работающий со всем множеством абстрактных ячеек памяти программы. В качестве демонстрации использования арифметических операций с указателями рассмотрим пример, в котором переменные размещены в памяти последовательно друг за другом (табл. 1).

Таблица 1

Сбор информации о псевдонимах

Текст программы на языке С	Информация о псевдонимах
`int a, m[10], b, x, *p;`	
`p=&m[2];`	`<m, 2>`
`if (*p > 0)`	`<m, 2>`
`  p=p-1;`	`<m, 1>`
`else`	
`  p+=7;`	`<m, 9>`
`x=*p;`	`<m, 1>; <m, 9>`
`p++;`	`<m, 2>; <b, 0>`
`p=m-1;`	`<a, 0>`

Таблица 2

Использование специальных имен

Фрагмент программы на языке С	Абстрактная ячейка памяти
`char *str;` `str = malloc( _MAX_PATH );`	`<DYNAMIC_MEMORY, 0>`
`int *p, *u;` `p=u;`	`<UNKNOWN_VALUE, 0>`
`int *q=NULL;`	`<NULL_VALUE, 0>`
`int v, *w, m[10];` `w=&m[v];`	`<m, UNKUOWN_OFFSET>`

Для единообразия работы с памятью множество имен переменных было дополнено следующими специальными именами:

- DYNAMIC_MEMORY — для памяти, выделенной одной из процедур захвата памяти (malloc(), calloc(), realloc()...);
- UNKNOWN_VALUE — для переменных, значения которых неопределенны;
- NULL_VALUE — для переменных, инициализированных 0 или макросом NULL.

При использовании специальных имен в Reprise к ним автоматически добавляется уникальное значение, чтобы получить уникальное имя.

Понятие "Индекс" абстрактной ячейки памяти было также расширено добавлением следующего значения — UNKNOWN_OFFSET, которое используется в тех случаях, когда получить числовой индекс невозможно. Такой индекс означает, что указатель может быть псевдонимом любого элемента массива.

Приведен пример использования специальных имен (табл. 2).

Анализ типов переменных

Анализ типов переменных позволяет точно определить, что два указателя не являются псевдонимами, если они указывают на переменные разных типов. Однако в языке С существуют и исключения. Так, например, указатель на тип char может быть псевдонимом для переменной, имеющей тип, отличный от char. В качестве примера рассмотрим следующие описания переменных:

```
int *i;
double *d, m[10];
char *p;
```

В этом примере переменная i не может быть псевдонимом переменных d или m, в то же время переменные d и m могут быть псевдонимами. А вот переменная p может быть псевдонимом любой из описанных выше переменных.

В тех ситуациях, когда анализ типов допускает, что переменные могут быть псевдонимами, применяется анализ указателей.

Анализ указателей

Задача анализа указателей заключается в вычислении для каждой точки программы, содержащей указатель, множества фрагментов памяти, адреса которых может принимать указатель в этой точке программы. Вообще говоря, задача статического анализа указателей является алгоритмически неразрешимой [9, §10], но существуют алгоритмы [13, 16], позволяющие построить приближенные множества объектов, на которые потенциально могут указывать указатели.

Алгоритмы анализа указателей можно разделить на два типа — решающие внутрипроцедурную и межпроцедурную части задачи анализа указателей.

Внутрипроцедурная часть анализа указателей

Внутрипроцедурная часть анализа основана на обходе *графа потока управления* (control flow graph) [17]. Следует отметить, что вершинами графа потока управления в ДВОР являются все операторы фрагмента программы. Эта особенность позволяет ускорить и упростить обновление графа при трансформациях фрагмента кода в процессе применения преобразований программ.

Во внутрипроцедурной части анализа используется итерационный алгоритм, начинающий свою работу с первой вершины графа потока управления процедуры. Разбирая узел графа, алгоритм определяет места записей в указатели и вычисляет их значения.

Основную сложность на внутрипроцедурном уровне представляют циклы. Чтобы вычислить значения указателей, находящихся внутри циклов, строится *дерево вложенности циклов*, в узлах которого содержится информация о циклах, а в листьях находятся выражения. В качестве примера построим дерево вложенности циклов для следующего фрагмента кода:

```
for (i = 0; i < N; i = i + 1)
  for (j = 0; j < N; j = j + 1)
  {
    if (m[i] [j] > 0) m[i][j] = 0;
    for (k = 0; k < N; k = k + 1)
      m[i][j] = m[i][j] + (*(p + i*N +
      k))*(*(q + k*N + j));
  }
```

Дерево вложенности циклов для этого фрагмента кода будет иметь вид, приведенный на рис. 3.

Далее вычисленные значения распространяются по графу потока управления (рис. 4).

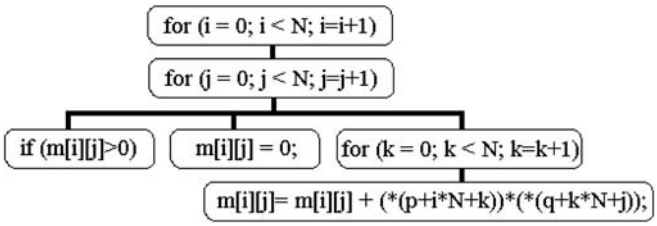


Рис. 3. Дерево вложенности циклов

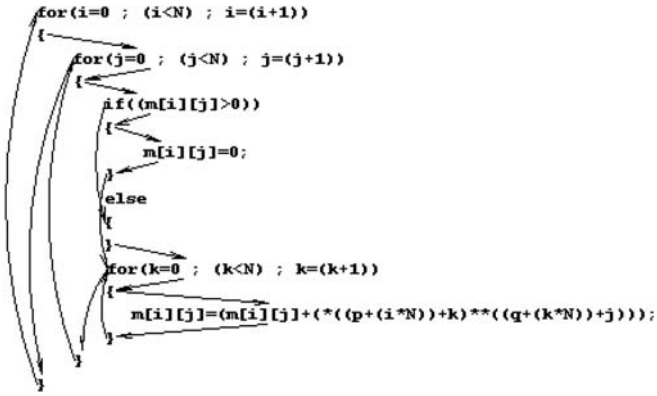


Рис. 4. Граф потока управления, автоматически построенный в ДВОР

В итоге, итерационный алгоритм завершит свою работу после того, как во всех узлах графа значения указателей окажутся вычисленными.

Для хранения значений указателей на внутрипроцедурном уровне используется *хэш-таблица вхождений указателей*. Эта таблица связывает с каждым вхождением переменной указательного типа множество абстрактных ячеек памяти, на которые может указывать данная переменная. В отличие от представления значений указателей диаграммами двоичных решений (*binary decision diagrams*) [16], использование хэш-таблицы вхождений указателей позволяет в момент анализа информационных зависимостей не выполнять никаких дополнительных вычислений.

Межпроцедурная часть анализа указателей

В межпроцедурной части анализа используются как *контекстно-чувствительный* (context-sensitive) [16], так и *контекстно-нечувствительный* (context-insensitive) [18] алгоритмы. Чувствительный к контексту алгоритм позволяет провести анализ указателей для каждого вызова процедуры с учетом контекста этого вызова, что существенно повышает точность анализа, но этот алгоритм является *NP*-трудным [16].

Для ускорения анализа используются шаблоны, позволяющие при необходимости применять нечувствительную к контексту версию алгоритма. Так, например, существенную проблему для контекстно-чувствительного алгоритма представляют рекур-

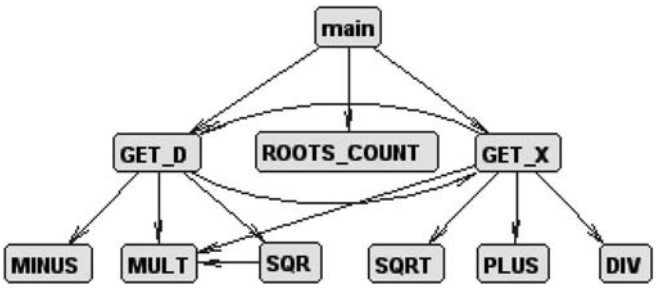


Рис. 5. Граф вызовов, автоматически построенный в ДВОР

сивные вызовы процедур. Для определения рекурсивных вызовов процедур используется *граф вызовов* (call graph) [7, §12.1.1], наличие петель на котором указывает на их присутствие (рис. 5).

При реализации контекстно-чувствительного алгоритма используются *записи активаций* (activation record) [19, §6] вызываемых процедур. Для проведения анализа указателей требуются следующие поля записей активаций:

- *фактические параметры* — список фактических параметров процедуры указательного типа с множеством значений;
- *возвращаемое значение* — множество значений указателя для результата, имеющего указательный тип;
- *связь по данным* — значения нелокальных переменных указательного типа на момент вызова процедуры.

На межпроцедурном уровне информация об указателях связывается с соответствующим узлом графа вызовов.

Заключение

На данный момент разработано множество преобразований программ для распараллеливающих компиляторов, но часть из них способна корректно работать лишь с переменными элементарных типов данных [8, §5.2], в то время как появление указателей в программах делает эти преобразования способными нарушить корректность программ. Описанный в данной работе алгоритм уточнения информационных зависимостей позволяет не допустить некорректное применение таких преобразований.

Работа поддержана ФЦП "Научные и научно-педагогические кадры инновационной России" на 2009—2013 годы, ГК 02.740.11.0208.

Список литературы

1. **Штейнберг Б. Я.** Распараллеливание программ для суперкомпьютеров с параллельной памятью и открытая распараллеливающая система: дис. ... д-ра техн. наук. Ростов н/Д, 2004. 343 с.

2. **Диалоговый** высокоуровневый оптимизирующий распараллеливатель программ (ДВОР). URL: http://www.ops.rsu.ru.

3. **Шульженко А. М.** Исследование информационных зависимостей программ для распараллеливающих преобразований: дис. ... канд. техн. наук. Ростов н/Д, 2006. 202 с.

4. **Allen R., Kennedy K.** Optimizing compilers for Mordern Architectures // Morgan Kaufmann Publisher, Academic Press. USA, 2002. 790 p.

5. **Feautrier P.** Dataflow analysis of scalar and array references // International Journal of Parallel Programming. February 1991. N 29(1). P. 23—52.

6. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 599 с.

7. **Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д.** Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. М.: Вильямс. 2008. 1184 с.

8. **Себеста Р. У.** Основные концепции языков программирования, 5-е изд.: Пер. с англ. М.: Вильямс, 2001. 672 с.

9. **Muchnick S.** Advanced compiler design and implementation. Morgan Kaufmann, 3rd ed., 1997.

10. **Чернов А. В.** Анализ запутывающих преобразований программ // Труды Института системного программирования / Под ред. В. П. Иванникова. М.: ИСП РАН, 2002.

11. **GNU** Compiler Collection (GCC). URL: http://gcc.gnu.org.

12. **Петренко В. В.** Внутреннее представление REPRISE распараллеливающей системы // РАСО'2008. Труды четвертой международной конференции "Параллельные вычисления и задачи управления". Москва, 27—29 октября 2008 г.

13. **Дроздов А. Ю., Владиславлев В. Е.** Межпроцедурный анализ указателей // Информационные технологии. Приложение № 2. 2005. 24 с.

14. **Diwan A., McKinley K. S., Moss J. E. B.** Type-based alias analysis // Proc. of the ACM SIGPLAN 1998 conference on Programming language design and implementation. Montreal, Quebec, Canada. June 17—19, 1998. P. 106—117.

15. **Yair Sade, Mooly Sagiv, Ran Shaham.** Optimizing C multithreaded memory management using thread-local storage // In Compiler Construction. 2005. P. 137—155.

16. **Whaley J., Lam M. S.** Cloning-based context-sensitive pointer alias analysis using binary decision diagrams // In SIGPLAN Conference on Programming Language Design and Implementation, 2004.

17. **Нис З. Я.** Анализ потока управления в открытой распараллеливающей системе // Искусственный интеллект. 2005. № 3. С. 461—464.

18. **Ruf E.** Context-insensitive alias analysis reconsidered // In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, June 1995. P. 13—31.

19. **Appel A. W., Ginsburg M.** Modern Compiler Implementation in C. Cambridge University Press. 1998.

УДК 004.056.55

В. В. Коробицын,
канд. физ.-мат. наук, доц., зав. каф.,
С. С. Ильин, аспирант,
Омский государственный университет
им. Ф. М. Достоевского
e-mail:nemossi@mail.ru

Реализация симметричного шифрования по алгоритму ГОСТ-28147 на графическом процессоре с использованием технологии CUDA

Предлагаются реализации базовой криптографической операции алгоритма симметричного шифрования ГОСТ-28147 на графическом процессоре с использованием технологии NVIDIA CUDA. Проводится сравнение с реализациями шифрования на видеокартах, использующими прикладные интерфейсы DirectX и OpenGL. Определяются наиболее быстродействующий вариант реализации и параметры функционирования системы, обеспечивающие максимально полное использование ресурсов графического процессора.

Ключевые слова: *симметричное шифрование, CUDA, параллельная обработка данных, графические процессоры*

Введение

Современные массивно-многоядерные процессоры, к которым можно отнести графические процессоры последних поколений, ориентированы на повышение эффективности решения задач за счет распараллеливания. Значительное превосходство графических процессоров над центральными процессорами по числу универсальных вычислительных ядер объясняет высокий уровень пиковой производительности, достигаемый при их использовании в ряде практических задач. Более привлекательное по сравнению с центральными процессорами соотношение уровня производительности графических процессоров к стоимости и потребляемой ими энергии делает их использование экономически целесообразным в случаях, когда вычислительную задачу удается эффективно перенести на GPU.

Одна из потенциальных возможностей использования GPU — выполнение на них криптографических преобразований. За последние несколько лет предпринимались попытки реализовать алгоритм AES на GPU различных поколений. Первые реализации использовали интерфейсы программирования трехмерной графики Direct3D и OpenGL [1]. Задача шифрования представлялась в терминах обработки массивов пикселей изображения, так что процесс шифрования заключался в вычислении цветов точек растрового изображения. В ранних реализациях шифрования на GPU [2, 3] не использовались возможности программирования шейдеров Direct3D или OpenGL, что приводило к трудностям в реализации и низкой производительности. Но даже тогда была подмечена возможность использования GPU для организации безопасной передачи графической информации при работе на терминалах и в приложениях видеоконференцсвязи. Более высокий уровень производительности был получен [4] с использованием программируемых шейдеров и специфичных возможностей, предоставляемых интерфейсом OpenGL, и объяснялся он меньшим числом вызовов функций прикладного API и более низкоуровневым и эффективным заданием операций, выполняемых процессорами GPU. Авторы также занимались реализацией симметричного шифрования с использованием GPU и интерфейсов Direct3D и OpenGL [5]. В их работе в качестве алгоритма для блочного шифра был выбран стандарт ГОСТ-28147 [6].

На графических процессорах предыдущего поколения возникали трудности с реализацией таких операций, как целочисленное сложение, выполнение операции XOR, реализация циклических сдвигов, логических побитовых операций. Эти трудности были связаны с отсутствием поддержки целочисленных операций самими графическими процессорами и с ограничениями графических API. Современные графические процессоры имеют полную поддержку арифметических и логических операций с 32-битными целыми числами, что значительно упрощает процедуру переноса криптографического кода на GPU, изначально предназначенного для CPU. Появление новых средств программирования позволило создать более производительные реализации шифрования на GPU.

С применением графических процессоров компании AMD и технологии CTM (close-to-metal)

было реализовано шифрование по алгоритмам DES и AES [7]. Компания NVIDIA выпустила технологию CUDA [8] в 2007 г. и внесла аппаратные изменения в свои процессоры для упрощения программируемости своих чипов, так что данное решение не является полностью программным. Теперь для запуска программ на графическом процессоре нет необходимости использовать интерфейсы программирования трехмерной графики. Более того, теперь программисты могут писать программы на стандартном языке C/C++ с минимумом изменений. Использование технологии CUDA также открывает доступ к специфическим возможностям CUDA-совместимых процессоров. С применением технологии CUDA были получены еще более высокие скорости шифрования по алгоритму AES [9, 10].

В данной работе авторы фокусируются на реализации базового криптографического преобразования по алгоритму симметричного шифрования ГОСТ-28147 на графическом процессоре с использованием технологии CUDA. Основной целью данной работы было исследование эффективности CUDA-реализации по сравнению с имеющейся реализацией на OpenGL и Direct3D. Главными задачами, которые ставили авторы, были поиск наиболее скоростной реализации функции шифрования на GPU и выявление условий, в которых возможно эффективное практическое использование GPU. Сама задача симметричного шифрования не предоставляет возможностей для распараллеливания. Возможность распределения нагрузки между ядрами GPU возникает в случае обработки достаточно большого числа блоков открытого текста. Поэтому необходимо установить параметры, при которых достигается наиболее полная загруженность GPU, и то, как эти параметры влияют на практическую применимость GPU для шифрования.

Программная модель CUDA

Для программиста вычислительная система CUDA состоит из хоста (система с обычным центральным процессором, например от Intel) и подключенного к нему одного или нескольких CUDA-совместимых устройств, содержащих большое число вычислительных блоков, работающих параллельно.

Программа CUDA состоит из нескольких фаз, которые могут выполняться как на *хосте* (CPU), так и на *устройстве* (GPU). Фазы, не поддающиеся распараллеливанию, исполняются на CPU, а расчеты с высокой степенью параллельности на уровне данных исполняются на GPU. Сама программа представляет собой обычный код на языке C/C++ с небольшим набором новых ключевых слов.

Порции кода, предназначенные для GPU, представляют собой функции, с некоторым набором переменных, задаваемых программистом. Такие функции называются *ядрами* (англ. *kernel*), их код исполняется одновременно большим числом потоков. У программистов CUDA есть возможность указать, сколько именно требуется запустить потоков и то, как они будут организованы. Потоки запускаются для выполнения ядра блоками, т. е. программист указывает, сколько блоков потоков запустить и сколько потоков содержит каждый блок. Потоки внутри блока могут иметь одно-, двух- или трехмерную организацию. Например, блок может содержать просто 256 потоков или 16 × 16 потоков, организованных в двумерную сетку. В функции ядра доступны специальные переменные, позволяющие идентифицировать каждый отдельный поток. Зная номер потока, можно выделить порцию данных, которую он должен обработать, определить место в памяти, куда записать результат, а также выбрать нужную ветку кода для исполнения.

Блочное шифрование на GPU с использованием CUDA

Задача реализации базового криптографического преобразования алгоритма ГОСТ-28147 на графическом процессоре с использованием CUDA заключается в разработке функции-ядра и замере скорости ее выполнения. Основное внимание данной работы было уделено оптимизации самой операции шифрования, поэтому все запускаемые на GPU потоки шифровали одинаковые по объему порции данных одним и тем же ключом.

После отправки блоков открытого текста на видеокарту на ней запускалось B одномерных блоков потоков по M потоков в каждом. Число блоков B было кратно числу мультипроцессоров в используемой видеокарте (27 для NVIDIA GeForce 260 GTX). Каждый поток обрабатывал P блоков открытого текста.

В функции-ядре, исполняемой каждым запущенным потоком, сначала вычислялся индекс потока, и определялось смещение в массиве данных, с которого данный поток начнет шифрование, после чего выполнялся цикл из P итераций, в этом цикле очередной блок данных считывался из памяти GPU, шифровался и записывался в выходной массив в памяти GPU.

При замерах времени шифрования не учитывалось время, затрачиваемое на пересылку данных из системной памяти на видеокарту и обратно.

Практическое применение такого режима шифрования весьма ограничено, но, тем не менее, позволяет оценить возможности использования GPU для шифрования блоков открытого текста как в

режимах со сцеплением блоков, так и в параллельных режимах, подобных ECB или CTR.

Реализация алгоритма шифрования ГОСТ-28147 на GPU

Стандарт ГОСТ-28147 определяет алгоритм криптографического преобразования для соответствующего блочного шифра. Сам шифр представляет собой сеть Фейстеля с 32 раундами и оперирует блоками по 64 бита. Размер ключа составляет 256 бит. Основные операции криптографического преобразования следующие: сложение по модулю 2^{32}; преобразование в S-блоках; циклический сдвиг влево; операция XOR.

Современные графические процессоры поддерживают целочисленную арифметику и побитовые логические операции. Эти операции выполняются с такой же скоростью, как и арифметические операции над числами с плавающей запятой, т. е. за 1 такт. Таким образом, для реализации одного раунда криптографического преобразования необходимо определиться с тем, как запрограммировать преобразования в S-блоках.

В алгоритме используются восемь S-блоков для преобразования 32-битного числа. Каждый из S-блоков применяется к отдельным четырем битам входного числа. На вход S-блока подается 4-битное число и на выходе получается также 4-битное число, а само преобразование описывается таблицей подстановки. Если на вход i-го S-блока подается 4-битное число x, то результат применения i-го S-блока будем обозначать $S_0^i(x)$. Для задания одного 4-битного S-блока необходимо заполнить массив из 16 4-битных элементов $S_0^k(j) \in \{0, 1, ..., 15\}, j \in \{0, 1, ..., 15\}$, $k = 0, 1, ..., 7$. Если ввести обозначение для 32-битного слова $X = (x_0, x_1, ..., x_7)$, состоящего из восьми 4-битных частей $x_0, x_1, ..., x_7$, то преобразование этого 32-битного числа восемью S-блоками можно записать в виде $S(X) = (S_0^0(x_0), S_0^1(x_1), ..., S_0^7(x_7))$.

Применение S-блоков может быть непосредственно запрограммировано с использованием восьми массивов, хранящих таблицы замен. Но для этого потребуется 8 раз извлекать отдельные 4-битные составляющие 32-битного слова, использовать их для выборки из таблиц и "упаковывать" полученные значения в новое 32-битное слово. Помимо большого числа битовых операций, затрачиваемых на побитовые сдвиги и маскирование, потребуется выполнить восемь операций доступа к таблицам. Поэтому было решено объединить S-блоки попарно, сделав четыре 8-битных S-блока. Таблицы замен для таких S-блоков уже содержат 256 8-битных элементов. И хотя выборки из меньших по размеру таблиц подстановок потенциально могут быть более быстрыми, вдвое большее их число и дополнительные инструкции для упаковки/распаковки битов делают их использование неэффективным.

Объединенные S-блоки определяются следующим образом: $S_g^k(16i + j) = S_0^{2k}(j) + 16 S_0^{2k+1}(i)$, где $i, j \in \{0, 1, ..., 15\}$, $k = 0, 1, 2, 3$. Если ввести обозначения для отдельных байтов 32-битного числа $Y = (y_0, y_1, y_2, y_3)$, где y_k соответствуют отдельным байтам слова, то преобразование 32-битного слова Y с использованием объединенных S-блоков можно записать следующим образом: $S(Y) = (S_g^0(y_0), S_g^1(y_1), S_g^2(y_2), S_g^3(y_3))$. Для вычисления 32-битного числа в программном коде потребуется объединить четыре байта в одной переменной: $S(Y) = S_g^0(y_0) | (S_g^1(y_1) \ll 8) | (S_g^2(y_2) \ll 16) | (S_g^3(y_3) \ll 24)$. Операций побитового сдвига можно избежать, если вместо 8-битных чисел использовать в таблицах 32-битные элементы и хранить в них 8-битные значения, заранее сдвинутые на необходимое число бит влево: $S_{32}^k(x) = S_g^k(x) \ll (8k)$, $x = 0, 1, ..., 255$, $k = 0, 1, ..., 3$. Тогда применение S-блоков будет описываться выражением: $S(Y) = S_{32}^0(y_0) | S_{32}^1(y_1) | (S_{32}^2(y_2) | S_{32}^3(y_3)$.

Использование 32-битных ячеек в таблицах замен позволяет совместить операцию применения S-блоков и последующий циклический сдвиг влево. Для этого достаточно циклически сдвинуть влево хранящиеся в таблицах значения: $S_{rcl}^k(x) = RCL_{11}(S_{32}^k(x))$, $x = 0, 1, ..., 255$, $k = 0, 1, ..., 3$. Если использовать четыре отдельных таблицы для хранения S_{rcl}^k, то преобразование 32-битного Y в S-блоках вместе с последующим циклическим сдвигом влево на 11 бит может быть вычислено путем выполнения четырех табличных подстановок и трех операций логического ИЛИ: $RCL_{11}(S(Y)) = S_{rcl}^0(y_0) | S_{rcl}^1(y_1) | S_{rcl}^2(y_2) | S_{rcl}^3(y_3)$. В некоторых реализациях четыре таблицы S_{rcl}^k, $k = 0, 1, 2, 3$ были совмещены в одну S_{rcl4}, значения элементов которой вычислялись по формуле $S_{rcl4}(x) = S_{rcl}^0(x) | S_{rcl}^1(x) | S_{rcl}^2(x) | S_{rcl}^3(x)$, $x = 0, 1, ..., 255$. Использование такого упакованного представления S-блоков приводит к необходимости выполнения дополнительных побитовых операций для извлечения полезных битов, а полное преобразование числа Y имеет вид: $RCL_{11}(S(Y)) =$

$= (S_{rcl4}(y_0)\ \&\ 7F800)|(S_{rcl4}(y_1)\ \&\ 7F80000)|(S_{rcl4}(y_2)\ \&\ \&\ F8000007)|(S_{rcl4}(y_3)\ \&\ 7F8)$.

Основной вопрос связан с тем, где и в какой форме расположить табличные данные *S*-блоков. При процессорной реализации такого вопроса не возникает, так как в этом случае данные таблиц подстановок размешаются в оперативной памяти в виде массива. При реализации на GPU непосредственное размещение массивов в глобальной памяти устройства неэффективно, так как у глобальной памяти очень высокая латентность, особенно при случайном доступе к табличным данным, который характерен для процедуры шифрования.

Для размещения таблиц возможны три варианта с использованием: кэша констант, текстурного кэша и разделяемой памяти мультипроцессоров.

Использование кэша констант. Массивы для констант определяются почти так же, как и обычные массивы в языке C/C++. Эти массивы заполняются путем копирования данных из системной памяти, после чего к этим массивам могут обращаться потоки GPU. Для кэша констант были использованы два вида таблиц для *S*-блоков, описанные выше, а именно S^k_{rcl} и упакованная таблица S_{rcl4}.

Использование текстурного кэша. Текстурный кэш позволяет обращаться к области глобальной памяти путем вызова специальных функций оперирования с текстурами. Использование текстурного кэша заключается в выделении части глобальной памяти GPU, заполнении ее данными с CPU и связывании ее с текстурной переменной, которая затем используется в операциях чтения из текстуры. Текстуры можно представлять себе как массивы, которые могут быть одной, двух- или трехмерными. При этом элементами массива могут быть как отдельные числа, так и векторы (до четырех составляющих). Данные текстур располагаются в линейном пространстве глобальной памяти GPU, что необходимо учитывать при связывании области памяти с текстурной переменной.

В данной работе были использованы три вида текстур:

- одномерная текстура с размерами 256 × 1, состоящая либо из 32-битных однокомпонентных элементов (при хранении упакованной таблицы S_{rcl4}), либо из 4-компонентных векторов, имеющих четыре 32-битные составляющие (для хранения четырех таблиц S^k_{rcl}, $k = 0, 1, 2, 3$);
- двумерная текстура с размерами 16 × 16; двумерность текстуры усложняет адресацию при считывании данных и увеличивает число инструкций в коде, однако именно для локальных двумерных выборок и оптимизирован текстурный кэш, поэтому производительность может оказаться выше, чем для линейных текстур, что уже было отмечено в работах [4, 5];
- двумерная текстура с размерами 256 × 256 и однокомпонентными 32-битными элементами. Элемент текстуры с индексом (i, j) заполняется следующим образом:

$$T(i, j) = S^0_{rcl}(i)|S^1_{rcl}(j)|S^2_{rcl}(i)|S^3_{rcl}(j),$$

что позволяет вычислить преобразование числа Y в виде

$$RCL_{11}(S(Y)) = (T(y_0, y_1)\ \&\ 7FFF8)|(T(y_2, y_3)\ \&\ F80007FF).$$

Использование текстуры большего размера позволяет уменьшить общее число операций доступа к таблицам, но текстура увеличенного объема может не полностью поместиться в текстурный кэш, что существенно замедлит чтение данных из такой текстуры [4].

Использование разделяемой памяти. Таблицы подстановок можно разместить также в разделяемой памяти. Потоки в начале своего исполнения должны скопировать табличные данные из глобальной памяти в разделяемую. Процесс копирования *S*-блоков из системной памяти в разделяемую легко распараллеливается между потоками одного блока. Например, если таблица подстановок состоит из 1024 элементов и запускается блок из 256 потоков, то каждый поток должен скопировать по 4 элемента в соответствующие ячейки разделяемой памяти, после чего необходимо будет выполнить синхронизацию потоков, и тогда табличные данные будут готовы для использования всеми потоками блока.

Проблемы возникают при попытке разместить таблицы S^k_g для 8-битных *S*-блоков в разделяемой памяти. Для размещения четырех таблиц из 256 8-битовых элементов требуется не менее 1 Кбайта памяти. Ввиду того, что разделяемая память разбита на банки и при доступе к ее ячейкам возникают конфликты, самым лучшим вариантом было бы размещение 16 копий табличных данных, по одной копии на каждый из банков. Однако сделать это невозможно, так как часть разделяемой памяти использует GPU для служебных целей, и все 16 Кбайт не доступны для программ CUDA. В итоге, в разделяемой памяти размещались два вида таблиц, такие же как и в случае с использованием константной памяти. Таблицы размещались в памяти линейно, занимая в ней все 16 банков в равной степени.

Дополнительной мерой увеличения быстродействия, использованной во всех реализациях, была стандартная операция развертывания циклов для раундов, увеличившая производительность на 10 %.

Результаты численного эксперимента

В тестах на измерение пиковой скорости шифрования имитировалось шифрование большого числа сообщений одинаковой длины в режиме ECB. Число B блоков CUDA было равно 27, число M потоков в блоке — 256, а число P блоков — 32. Пиковые скорости шифрования, наблюдаемые для различных реализаций, перечислены в таблице (символ * используется для указания того факта, что помеченный вариант работает с упакованной таблицей S_{rcl4}).

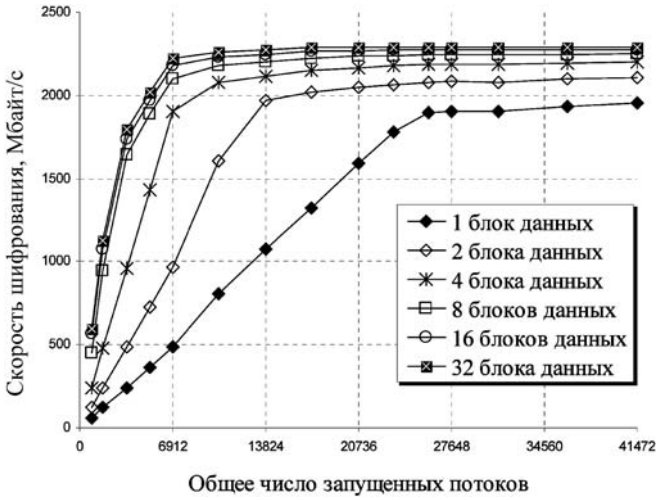
Для сравнения реализаций шифрования на GPU с использованием CUDA и OpenGL/Direct3D, в таблицу внесены скорости шифрования, измеренные при запуске программы, использованной авторами в работе [5].

Самой быстрой является реализация с использованием четырех таблиц S^k_{rcl}, размещаемых в разделяемой памяти. С этим вариантом программы были проведены дополнительные тесты в целях изучения зависимости скорости шифрования от общего числа потоков и оценки возможностей применения GPU для обработки сообщений с различными режимами сцепления блоков. В этих тестах имитировалось шифрование N сообщений фиксированной длины P. Общее число потоков N варьировалось в пределах от 864 (запуск 27 блоков по 32 потока в каждом) до 41472 (запуск 81 блока по 512 потоков в каждом). Во всех случаях число запускаемых блоков было кратно числу мультипроцессоров, а число M потоков в блоке варьировалось в пределах от 32 до 512. Число P менялось в диапазоне от 1 до 32 в геометрической прогрессии. Результаты тестов представлены на рисунке.

Из графиков на рисунке видно, что скорость обработки коротких сообщений некоторое время растет линейно с ростом числа запущенных потоков. Такой характер роста может быть связан с некоторым фиксированным временем, затрачиваемым на запуск потоков. Для оценки этого времени был выполнен отдельный тест, в котором исполнялась

Скорости шифрования на GPU

Вариант реализации	Скорость шифрования, Мбайт/с
Разделяемая память	2308
Разделяемая память*	1936
Линейная текстура	553
Линейная текстура*	556
Текстура 16 × 16	623
Текстура 16 × 16*	1758
Текстура 256 × 256	408
Константная память	779
Константная память*	384
Direct3D 9.0c	384
OpenGL 2.1	660
CPU, Intel Q9400@3500 МГц, одно ядро	53



Зависимость скорости шифрования на GPU от общего числа запущенных потоков при обработке сообщений разной длины

пустая функция-ядро с одной операцией возврата и замерялось время работы GPU. Оказалось, что время запуска сетки потоков в описанных тестах практически не зависит от числа потоков. Эксперименты показали, что при запуске N потоков, обрабатывающих по P блоков, верхняя граница скорости шифрования может быть оценена выражением $Spd_{enc} = 69{,}95 \cdot 10^{-3} NP$ Мбайт/с. Для используемой видеокарты это сравнимо с добавлением $33 \cdot 10^6 (NP)^{-1}$ тактов к шифрованию одного блока данных. Число же тактов, необходимое для самого шифрования одного блока данных, было оценено числом 990. Поэтому для небольших значений N и P наблюдается существенное ограничение скорости шифрования и линейный характер роста.

Для более длинных сообщений с увеличением числа потоков происходит стремительный рост производительности, которая уже ограничивается скоростью непосредственно шифрования и зависит от того, насколько хорошо процессор справляется с диспетчеризацией потоков и скрытием латентности памяти. Можно сделать вывод, что эффективное использование GPU возможно для сообщений, имеющих длину не менее 16 64-битовых блоков или 128 байт. Для эффективного переключения потоков необходимо запускать не менее 256 потоков на один мультипроцессор.

Выводы

В работе были представлены несколько реализаций базового криптографического шифрования по алгоритму ГОСТ-28147 на графических процессорах, поддерживающих архитектуру CUDA. Использование специфических возможностей, предоставляемых архитектурой CUDA, позволило создать более производительную реализацию криптографического преобразования по сравнению с

программами, основанными на вызовах OpenGL или Direct3D. Пиковая скорость шифрования на GPU при обработке больших пакетов данных превышает скорость ассемблерной реализации на CPU с использованием одного ядра более, чем в 40 раз. Реализация шифрования на GPU с использованием разделяемой памяти может быть взята за основу для построения криптографической системы, поддерживающей сообщения произвольной длины, множество ключей и произвольные режимы сцепления блоков шифротекста.

Список литературы

1. **Brown P.** The OpenGL Graphics System: A Specification (Version 2.1). 2006-06-30. Silicon Graphics, 2006. 380 p.
2. **Cook D., Keromytis A.** CryptoGraphics / Exploiting Graphics Cards For Security. Berlin: Springer-Verlag, 2006. 139 p.
3. **Cook D. L., Ioannidis J., Keromytis A. D., Luck J.** CryptoGraphics: Secret Key Cryptography Using Graphics Cards // Topics in Cryptology, The Cryptographers' Track at the RSA Conference 2005 (CT-RSA 2005). LNCS. Vol. 3376. Berlin, Heidelberg: Springer-Verlag, 2005. P. 334—350.
4. **Harrison O., Waldron J.** AES Encryption Implementation and Analysis on Commodity Graphics Processing Units // 9th Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007). LNCS. Vol. 4727. Berlin, Heidelberg: Springer-Verlag, 2007. P. 209—226.
5. **Коробицын В. В., Ильин С. С.** Реализация симметричного шифрования по алгоритму ГОСТ-28147 на графическом процессоре // Информационные технологии. 2005. № 10. С. 46—51.
6. **ГОСТ 28147—89.** Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. — Введ. 1990-01-01. М.: Изд-во стандартов, 1989.
7. **Yang J., Goodman J.** Symmetric Key Cryptography on Modern Graphics Hardware // 13th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2007). LNCS. Vol. 4833. — Berlin, Heidelberg: Springer-Verlag, 2007. P. 249—264.
8. **NVIDIA CUDA.** URL: http://www.nvidia.com/object/cuda_home_new.html
9. **Bos J. W., Osvik D. A., Stefan D.** Fast Implementations of AES on Various Platforms. Cryptology ePrint Archive. Report 2009/501. November 2009. URL:http://eprint.iacr.org.
10. **Manavski S. A.** CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography // IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 24—27 November 2007.

УДК 004.056.57

Р. М. Алгулиев, чл.-корр. НАНА,
д-р техн. наук, проф., директор,
С. А. Назирова, диссертант, науч. сотр.,
Институт информационных технологий НАНА,
г. Баку
e-mail:sbunyadova@gmail.com

Об одном подходе к формированию и реализации политики борьбы со спамом с учетом требований прав человека

Предлагается механизм разработки политики борьбы со спамом, учитывающий многочисленные субъективные оценки пользователей, структур, даже государств на отнесение той или иной поступившей корреспонденции к категории спам. Фильтрация на верхних уровнях ведется на основании запросов пользователей нижнего уровня. При этом не нарушается Всеобщая декларация прав человека. Предложенная система очень гибкая, так как без каких-либо ограничений принимает все мнения пользователей в формировании и осуществлении своей политики борьбы со спамом. Предлагается пошаговый алгоритм рассмотренного подхода.

Ключевые слова: *спам, фильтрация электронной почты, права человека, политика борьбы со спамом, многоуровневая архитектура*

Введение

Глобальное информационное пространство дает человеку возможность получать и распространять информацию в мировом масштабе. Все большее значение приобретают информационные права личности, связанные с этим процессом, их реализация и защита.

Права человека свободно искать, получать и распространять информацию закреплены в международных актах, провозглашающих права человека в области информации и телекоммуникаций [1—4]. Информационное общество должно обеспечивать правовые и социальные гарантии того, что каждый гражданин в любом пункте, в любое время может получить любую информацию, необходимую для его деятельности и решения стоящих перед ним проблем [5, 6].

Другой очень важной проблемой глобального информационного пространства является спам. Согласно исследованиям компании "Postini" в настоящее время 90 % трафика электронной почты занимает спам [7]. Более того, в последнее время спам приобретает новые функции, которые используются спамерами как способы распространения информации криминального характера, целью которых являются мошенничество, фишинг, фальсификация, "черный PR", реклама контрафактных или поддельных товаров, информационная

атака и т. п. Киберпреступность в настоящее время является одной из наиболее серьезных угроз национальной безопасности в информационной сфере. В связи с этим резкое увеличение спама криминального характера вызывает естественную озабоченность.

Сложность борьбы со спамом проявляется в том, что в этом процессе следует учитывать многочисленные субъективные оценки, подходы и взгляды пользователей, структур, даже государств на отнесение той или иной поступившей корреспонденции к категории спам. Некоторые страны под видом борьбы со спамом ограничивают свободу на распространение и приобретение информации в Интернет-среде, тем самым нарушая принципы демократии и прав человека. Вместе с тем, в настоящее время во многих странах не приняты нормативно-правовые акты, т. е. отсутствует какая-либо политика, предназначенная для борьбы со спамом. Данная ситуация создает благоприятную среду для неблаговидной деятельности спамеров.

Однако какой бы ни была эта политика борьбы со спамом, она должна основываться исключительно на базовых нормах и принципах прав человека. Поэтому прежде всего необходимо учитывать отношения пользователя к той или иной корреспонденции. Кроме того, ситуация осложняется еще тем, что мнение пользователя не является стабильным, постоянным. Оно меняется в зависимости от настроения пользователя и ряда других субъективных обстоятельств. Поэтому какой-нибудь жесткий, статический подход при фильтрации от нежелательных писем в какие-то моменты может нарушить нормы и принципы прав человека. В этой связи необходим динамический подход к борьбе со спамом, который мог бы учесть изменчивое отношение самих пользователей по мере его появления в процессе просмотра электронной почты.

В существующих индивидуальных и корпоративных антиспам-системах фильтры обучаются, как правило, на ограниченном числе сообщений, поступающих только конкретному пользователю или конкретному провайдеру [8, 9]. Вследствие этого не обеспечивается качественная фильтрация спама и остается нерешенной задача совместной фильтрации с вовлечением отдельных пользователей и Интернет-провайдеров. Качество фильтрации может быть повышено за счет применения комплексных иерархических и многопользовательских систем фильтрации, обеспечивающих полномасштабное участие пользователей в процессе выявления ошибок фильтрации и соответствующей настройке фильтров на каждом уровне [10]. Но в настоящее время в мире отсутствуют правовые механизмы борьбы со спамом, за исключением некоторых деклараций и конвенций.

Предлагаемый подход

Рассматривается централизованная система фильтрации от нежелательной корреспонденции на уровне координации работы всех функционирующих в стране сервис-провайдеров и совместной фильтрации от спама с вовлечением пользователей этой системы и Интернет-провайдеров. Этот механизм может реализовываться на уровне сервис-провайдеров, которые динамически вместе с клиентами-пользователями могут составлять и обновлять "черные списки" Е-почты. Интернет-провайдеры оперативно могут делегировать или удалять данные, передавать "черные списки" в распоряжение ССП (сетевой сервис-провайдер, NSP), который обеспечивает сервис-провайдеров Интернет-трафиком. Как видно, такое совместное, согласованное взаимодействие пользователей и провайдеров на национальном уровне, с одной стороны, обеспечивает эффективную борьбу со спамом, основанную на принципах прав человека, с другой — в целом обеспечивает "экологичность" информационной среды.

Слабым местом данного подхода является появление возможности определения личных отношений пользователей в течение длительного времени к тем или иным контентам, которые могут расцениваться как нарушения других принципов прав человека касательно неприкосновенности личной жизни. Поэтому соответствующие государственные структуры подготавливают требования к обеспечению защиты персональных данных пользователей.

В любом случае каждый пользователь должен быть информирован о существовании данной опасности. После этого он сам принимает решение об участии в реализации рассматриваемой политики борьбы со спамом.

Рассматривается единая иерархическая система для каждого уровня, для которой должна быть сформирована своя политика борьбы со спамом, и на ее основе реализуется фильтрация спама на каждом уровне (см. рисунок на четвертой стороне обложки).

Предлагаемая система имеет многослойную иерархическую структуру, состоящую из трех уровней: государственного, корпоративного и персонального [10]. Каждый уровень, в свою очередь, имеет свою политику борьбы со спамом, которая определяется совокупностью признаков, содержащихся в спам-сообщениях на серверах сервис-провайдеров этого уровня. На узлах системы находятся конечные пользователи и сервис-провайдеры. Сервис-провайдеры, корпоративные почтовые серверы и клиентские компьютеры являются вертикальными составляющими предлагаемой иерархической системы. Запросы на фильтрацию

от нежелательной корреспонденции отправляются с узлов нижних уровней к узлам верхних.

На каждом уровне многослойной иерархической системы имеются серверные узлы, в которых собирается база спам-сообщений, приходящих от узлов нижних уровней или же от обычных узлов того же уровня. Фильтрация от нежелательной корреспонденции может проводиться на любом из этих уровней, однако предлагаемая методика подразумевает фильтрацию, которая ведется на верхнем уровне — на уровне сервис-провайдеров, в то время как информация для обработки поступает с нижних уровней. Фильтрация ведется сверху вниз, а база спам-шаблонов и правил обеспечивается снизу наверх. Базы спам-шаблонов предлагается формировать рапортами пользователей о наличии спама в полученной корреспонденции.

При горизонтальном обмене информации на самом верхнем уровне сервис-провайдеры могут быть объединены в определенные целевые группы. Таким образом, происходит обмен информации между сервис-провайдерами и пользователями.

Там, где есть обмен информации, должна быть стандартизация языка общения между информационными узлами. Стандартизированный язык, определяющий поведение узлов при передаче данных, назовем протоколом обмена информацией о спаме между узлами, далее именуемым как SIEP (Spam Information Exchange Protocol — протокол обмена информацией о спаме). Этот протокол создается добавлением новых полей к существующему SMTP-протоколу, которые будут отражать набор правил взаимодействия узлов, описывать синтаксис сообщения, имена элементов данных, операции управления и состояния. С помощью расширенного протокола можно будет фиксировать корреспонденцию, объявленную пользователем как спам, а также информацию для статистики — время поступления письма пользователю; время отправки пользователем рапорта; IP-адрес сервера, с которого поступила нежелательная корреспонденция; информация о пользователе и т. д.

Формализация предлагаемого подхода

Введем некоторые обозначения:

N — число уровней предлагаемой многоуровневой архитектуры;

J_i — число узлов на i-м уровне, на которых должны быть размещены серверы, $i = \overline{0, N}$;

K_{j_i} — число узлов на i-м уровне, которые прикреплены к серверным узлам j_i, $i = \overline{0, N}$ $j_i = \overline{1, J_i}$.

Так как предложенная система предполагается динамичной и обучаемой, а базы спам-шаблонов постепенно должны пополняться новыми шаблонами, вводится параметр времени $t \in T$;

$s^z_{k_{j_i}}(t)$ — z-е сообщение, делегированное узлом k_{j_i} как спам в момент времени t, где $z \in Z$, $t \in T$, $k_{j_i} = \overline{1, K_{j_i}}$;

$s^z_{k_{j_i}}(t+1)$ — z-е сообщение, делегированное узлом k_{j_i} как спам в момент времени $t+1$, где $z \in Z$, $t \in T$, $k_{j_i} = \overline{1, K_{j_i}}$;

$Q_{j_i}(s^z_{k_{j_i}}(t))$ — общее число ситуаций делегирования серверному узлу j_i о спамности сообщения $s^z_{k_{j_i}}(t)$ на время t, $z \in Z$, $j_i = \overline{1, J_i}$, $i = \overline{0, N}$, $t \in T$;

$U_{j_i}(t)$ — множество спам-шаблонов серверного узла j_i в момент времени t:

$$U_{j_i}(t) = \{s^{z_1}_{k_{j_i}}(t),\ s^{z_2}_{k_{j_i}}(t),\ ...,\ s^{z_l}_{k_{j_i}}(t)\}, \qquad (1)$$

где $j_i = \overline{1, J_i}$, $i = \overline{0, N}$, $z \in Z$, $t \in T$, $k_{j_i} = \overline{1, K_{j_i}}$, $l = \overline{1, L}$;

$U_{j_i}(t+1)$ — множество спам-шаблонов серверного узла j_i в момент времени $t+1$, определяемое следующим образом:

$$U_{j_i}(t) = \{s^{z_1}_{k_{j_i}}(t),\ s^{z_2}_{k_{j_i}}(t),\ ...,\ s^{z_l}_{k_{j_i}}(t),\ s^{z_{l+1}}_{k_{j_i}}(t)\}, \qquad (2)$$

где $j_i = \overline{1, J_i}$, $i = \overline{0, N}$, $z \in Z$, $t \in T$, $k_{j_i} = \overline{1, K_{j_i}}$.

Спам-сообщение будет фильтроваться для узла k_{j_i}, если удовлетворится следующее равенство:

$$Q_{j_i}(s^z_{k_{j_i}}(t)) = K_{j_i}. \qquad (3)$$

Значит, система будет фильтровать сообщение для узла k_{j_i} только и только в том случае, если это сообщение будет признано спамом достаточным числом узлов, прикрепленных к серверному узлу j_i.

Множество спам-шаблонов уровня i в момент времени t равно пересечению множеств спам-шаблонов узлов j_i в момент времени t:

$$U_i(t) = \bigcap_{j_i=1}^{J_i} U_{j_i}(t), \qquad (4)$$

где $i = \overline{0, N}$, $j_i = \overline{1, J_i}$, $t \in T$.

То есть это множество состоит из тех спам-шаблонов, которые в тот момент делегированы как нежелательное сообщение всеми пользователями того уровня.

Политику борьбы со спамом $P^s_{j_i}(t)$ серверного узла j_i в момент времени t можно представить как вектор, составляющими которого являются политики борьбы со спамом каждого обычного узла:

$$P^s_i(t) = \{P^s_{j_1}(t),\ P^s_{j_2}(t),\ P^s_{j_3}(t),\ ...,\ P^s_{J_i}(t)\}, \quad (5)$$

где $i = \overline{0, N}$, $j_i = \overline{1, J_i}$, $t \in T$.

Политика борьбы со спамом узла j_i определяется множеством спам-шаблонов этого узла:

$$P^s_{j_1}(t) \Rightarrow U_{j_i}(t), \quad (6)$$

где $i = \overline{0, N}$, $j_i = \overline{1, J_i}$, $t \in T$.

Политика борьбы со спамом уровня i определяется множеством спам-шаблонов этого уровня:

$$P^s_{i_i}(t) \Rightarrow U_i(t), \quad (7)$$

где $i = \overline{0, N}$, $j_i = \overline{1, J_i}$, $t \in T$.

В представленной системе допускается возможность обратно отозвать (восстановить) сообщение, ранее отмеченное как спам. В этом случае сообщение $s^z_{k_{j_i}}(t)$, делегированное узлом k_{j_i} как спам на время t, удаляется из множества спам-шаблонов $U_{j_i}(t)$. Соответственно меняются общая база спам-шаблонов $U_i(t + 1)$ и политика борьбы со спамом $P^s_i(t + 1)$ уровня $i = \overline{0, N}$. Динамический алгоритм системы позволит восстанавливать состояние динамической системы в темпе реального времени (по ходу процесса), используя входную информацию о системе в текущие дискретные моменты времени. Как мы видим, несмотря на то, что фильтрация в нашей системе идет сверху вниз, на самом деле предложенная система управляется снизу вверх.

Комплекс решений включает в себя специализированное приложение для контекстного сравнения, сопоставления корреспонденции со спам-шаблонами и прогонки корреспонденции через определенные фильтры, которые прибавляют или убавляют вероятность идентификации этой корреспонденции как спам, и предоставления выбора для дальнейшей манипуляции над этим письмом (попадания этого письма в Indox или в Junk E-mail (папка для спама) или же отказа в доставке конечному пользователю). Сервис-провайдер предлагает конечному пользователю либо получить корреспонденцию, идентифицированную как спам, либо отказаться от нее вообще. Во втором случае сервис-провайдер перестанет в дальнейшем пересылать письма с таким же содержанием.

Предложенный подход базируется исключительно на интересах пользователей, а не каких-то государственных структур и организаций.

Отметим, что спам-шаблоны, собранные на базах, классифицируются и параметризируются. По этим параметрам можно будет также определять источник спама и устанавливать тематическую зависимость от географической (например, какие тематики преобладают в спам-сообщениях, отправленных из определенных стран). Таким образом, система будет способна выявлять целенаправленные информационные атаки, если таковые будут происходить. Анализируя источники спам-сообщений, находящихся в базах спам-шаблонов, можно определить и раскрыть организованные спамерские группировки.

Кроме того, система может содержать свои "слабые" спам-фильтры, работающие сверху вниз, основанные на формальных признаках электронных сообщений [11].

Для ясности алгоритмизации предложенного подхода опишем пошаговый алгоритм обмена почты предложенной системы фильтрации спама.

Шаг 1. Пользователь проверяет почту.

Шаг 2. Полученное письмо проверяется на элемент спамности.

Шаг 3. Легитимная корреспонденция попадает в папку "входящие".

Шаг 4. Корреспонденция, отмеченная как спам, делегируется серверу для последующей обработки.

Шаг 5. Серверная часть анализирует письмо и выявляет индекс спамности, который характеризируется общим числом операций делегирования данного письма.

Шаг 6. Письмо проверяется на соответствие спам-шаблонам из базы.

Шаг 7. Если соответствие не обнаружено, письмо добавляется в базу данных и ему присваивается начальный индекс спамности.

Шаг 8. Если соответствие обнаружено, то индекс спамности сопоставляется с общим числом пользователей.

Шаг 9. Если индекс спамности письма меньше числа пользователей, то индекс увеличивается на единицу.

Шаг 10. В противном случае письмо блокируется как спам.

Шаг 11. В случае, если пользователь изменит свое мнение о письме, ранее делегированном как спам, и обратно отзовет его, то в дальнейшем это письмо будет приходить в почтовый ящик пользователя.

Шаг 12. Индекс спамности отозванного письма-шаблона в базе данных уменьшается на единицу.

Заключение

Таким образом, данный подход, который основывается на Всеобщей декларации прав человека и имеет универсальный характер, может быть применен во всех странах мира. Реализация этого подхода практически не составляет особых трудностей, кроме мер по повышению информационной культуры Интернет-пользователей в каждой стране, чтобы те активно принимали участие в борьбе со спамом как на корпоративном, так и на государственном уровне. Если все пользователи будут иметь определенный уровень знаний по информационной культуре и безопасности, то всенародная борьба со спамом приведет к экологичности информационного общества страны.

Список литературы

1. **Окинавская** Хартия глобального информационного общества от 22 июля 2000 г. // Международное право. 2000. № 3. С. 410.
2. **Convention** on Cybercrime, Budapest, 23.11.2001. URL: http://conventions.coe.int/Treaty/Commun/QueVoulezVous.asp?NT=185&CL=ENG
3. **Алгулиев Р., Джабраилова З.** Азербайджан — соорганизатор Тунисского саммита в создании информационного общества // Ренессанс. Баку: 15 ноября 2005 (на азерб.) // Renessans. Baku: 15-Noyabr-2005.
4. **London** Action Plan on International Spam Enforcement Cooperation, October 11, 2004. URL: http://www.londonactionplan.org/?q=node/1
5. **Алгулиев Р. М., Назирова С. А.** Проблемы информационной экологии и пути их решения // Известия Азербайджанского национального аэрокосмического агентства: Информатика и проблемы управления, 2008. Т. XI. № 3. С. 48—55.
6. **Всеобщая** декларация прав человека (принята и провозглашена резолюцией 217A (III) Генеральной Ассамблеи от 10 декабря 1948 года). URL: http://www.un.org/russian/documen/declarat/declhr.html
7. **Postini** Spam Bulletin. Spam trends and security best practices. URL: http://postini.com/webdocs/rel_notes/announce/bulletin_spam.html
8. **Баранов П. А.** Обзор средств борьбы со спамами и E-mail вирусами // Проблемы информационной безопасности. Компьютерные системы. 2004. № 1. С. 44—50.
9. **Pelletier L., Almhana J., Choulakian V.** Adaptive Filtering of SPAM // Proceedings of the Second Annual Conference on Communication Networks and Services Research. IEEE. 2004. P. 530—537.
10. **Алгулиев Р. М., Назирова С. А.** Архитектура иерархической интеллектуальной общегосударственной системы борьбы со спамом // Информационные технологии. 2006. № 8. С. 32—36.
11. **Nazirova S.** New Anti Spam methods / PCI'2008 — the Second International Conference Problems of Cybernetics and Informatics, Baku, September 10—12. 2008. Vol. I. P. 89—92. URL: http://pci2008.science.az/1/23.pdf

УДК 004.27

А. Е. Мамченко, канд. техн. наук, доц.,
В. Г. Першеев, канд. техн. наук, доц.,
Московский государственный
университет путей сообщения (МИИТ)
e-mail: vss.miit@gmail.com

О кодах представления чисел с фиксированной точкой (запятой) в компьютерах и вычислительных системах

Рассматриваются используемые в вычислительной технике представления чисел с фиксированной точкой: беззнаковое представление, прямой, обратный и дополнительный коды, смещенный код и смещенный код с отрицательным нулем. Простейшие действия над числами в этих кодах обсуждаются без доказательств.

Ключевые слова: *компьютер, числа, фиксированная точка, коды представления целых чисел*

Введение

Представление чисел в вычислительной аппаратуре и алгоритмы их обработки, объединяемые общим термином "машинная арифметика", были и будут оставаться краеугольным камнем организации и реализации вычислительных машин и систем. Понятно, что эти вопросы являются весьма важными в подготовке не только специалистов по разработке компьютеров, но и представляют интерес для широкого круга лиц, специализирующихся в области информационных систем и технологий. Хотя алгоритмы кодирования чисел и выполнение основных операций над ними разрабатывались со времени появления первых ЭВМ, с развитием вычислительной техники возник ряд новых подходов, обеспечивающих простоту представления и высокую скорость реализации арифметических операций. При этом некоторые методологические вопросы остаются неосвещенными в современной литературе. Они и рассматриваются ниже.

В настоящее время цифровая вычислительная техника базируется на способе представления чисел, использующем *позиционные системы счисления*. При этом числа представляются в виде наборов символов (цифр), записываемых в определенном порядке. Последний задается *разрядной сеткой* — указателем количества, положения и номеров используемых для представления числа позиций (разрядов). Веса отдельных разрядов определяются как p^j, где p — основание системы счисления, а j — номер разряда в сетке. Цифрами обычно являются 0, 1, ..., $p-1$, хотя, в принципе, могут быть и другие.

Количественный эквивалент чисел, записанных в разрядной сетке (рис. 1), определяется полиномом

$$X = x_{n-1}p^{n-1} + x_{n-2}p^{n-2} + \ldots + x_1p^1 + x_0p^0 + \\ + x_{-1}p^{-1} + x_{-2}p^{-2} + \ldots + x_{-m}p^{-m}. \qquad (1)$$

Здесь использованные в сетке разряды пронумерованы от $(-m)$ до $n-1$, а x_j — цифра в разряде j. Граница между разрядами 0 и (-1) считается отмеченной *точкой* (*запятой*) для удобства задания различных сеток.

Формы рассмотренного выше представления чисел могут быть разными. Чаще других используется *естественная форма* (представление с *фиксированной точкой*) и *форма с плавающей точкой*. Естественная форма предполагает, что числа представляются в виде наборов цифр, положение которых относительно точки в разрядной сетке неизменно. Например, точка может считаться зафиксированной после последнего (младшего) разряда в наборе цифр (такое часто встречающееся представление называется *целочисленным* (рис. 2)).

При фиксации точки перед старшим разрядом в наборах цифр получаем разрядные сетки для представления правильных дробей.

n-1	*n*-2	…	1	0	-1	-2	…	-*m*
x_{n-1}	x_{n-2}	…	x_1	x_0	x_{-1}	x_{-2}	…	x_{-m}

Рис. 1. Разрядная сетка с ($n + m$) разрядами (позициями)

а)

3	2	1	0
x_3	x_2	x_1	x_0

б)

7	6	5	4	3	2	1	0
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

Рис. 2. Два примера целочисленных форматов:
а — тетрада; *б* — байт

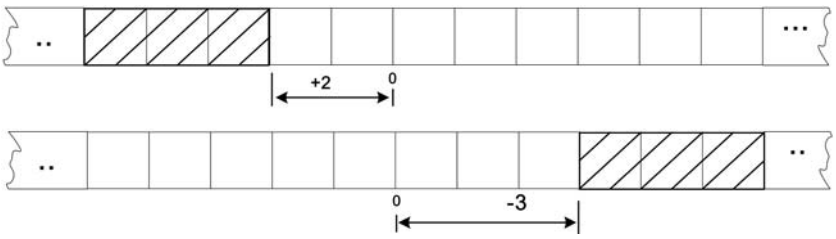


Рис. 3. К пояснению представления чисел с плавающей точкой

Формат представления чисел в форме с плавающей точкой задает два параметра: цифры, записываемые в разрядной сетке, и их местоположение (иначе *порядок*) относительно точки в ней (рис. 3).

В соответствии с принципом относительности можно говорить, что точка сдвигается (перемещается, плавает) относительно зафиксированного набора цифр. Отсюда название формы представления и определение числа по выражению

$$X = M_x P^{Q_x}.$$

В этом выражении M_x — число, которое после умножения на P^{Q_x} (смещение по сетке на величину порядка Q_x) станет соответствовать числу X в естественной форме представления. Величина M_x называется *мантиссой* и обычно фиксируется в диапазоне $1/p \leqslant |M_x| < 1$ (в последнее время чаще $1 \leqslant |M_x| < p$).

Таким образом, используя форму представления чисел с плавающей точкой, мы должны задать два числа — мантиссу и порядок. Каждое из них представляется в естественной форме, что позволяет говорить о фундаментальной значимости кодирования чисел в форме с фиксированной точкой. Будем помнить, что рассматриваемые величины, как правило, имеют знак.

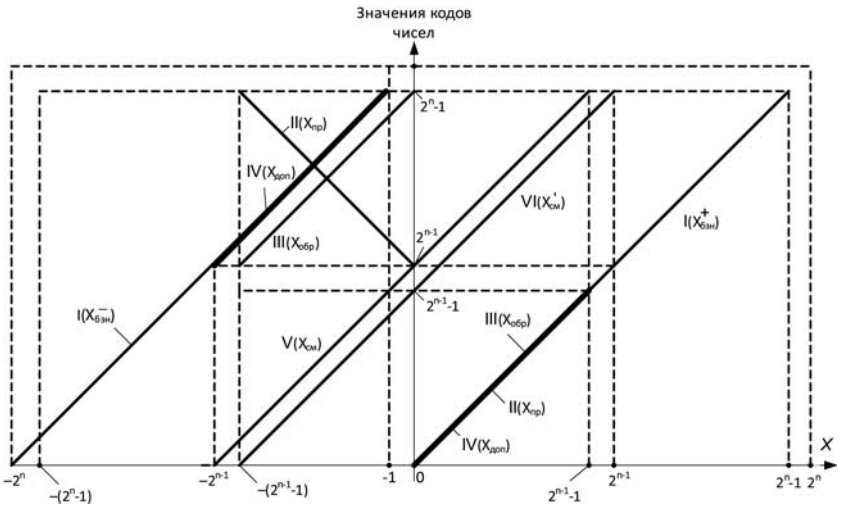


Рис. 4. Графики кодов двоичных целых чисел со знаками

В дальнейшем эти величины будем считать изначально представленными своими модулями в *двоичной* системе счисления ($p = 2$), а знак величины — заданным отдельно. Код модуля будем считать совпадающим с записью модуля числа в рассматриваемой разрядной сетке. Если формат (сетка) содержит больше разрядов, чем необходимо для записи модуля, "лишние" разряды заполняются нулями. При нехватке разрядов для представления модуля требуемой величины или с требуемой точностью формат должен быть заменен на другой — большей разрядности.

Коды представления двоичных целых чисел со знаком

Говоря о числах с фиксированной точкой в общем случае, имеют в виду смешанные дроби, представляемые в разрядной сетке с n разрядами для записи целой части числа и с m разрядами для дробной. Чаще других используются сетки для целых чисел ($m = 0$ и вес младшего разряда равен 2^0) и для смешанных дробей ($m \neq 0$ и $n \geqslant 1$). Последний случай подразумевает описание мантисс чисел с плавающей точкой.

Дальнейшее изложение касается целых чисел, что позволяет упростить изложение, не умаляя, в конечном счете, его общности.

Обычно рассмотрение представления чисел в вычислительной технике начинают с так называемой *беззнаковой формы*, в которой знаки чисел задаются программой их обработки. Все n разрядов кода здесь несут информацию только о модуле числа:

$$X_{\text{бзн}} = \begin{cases} |X| & \text{при } 2^n - 1 \geqslant X \geqslant 0; \\ 2^n - |X| & \text{при } -1 \geqslant X \geqslant -2^n. \end{cases} \qquad (2)$$

Из формулы (2) видно, что положительные числа $X^{+}_{\text{бзн}} = |X|$ (при $0 \leqslant X \leqslant 2^n - 1$) изображаются *модулями*, а отрицательные — *дополнениями* модуля до 2^n: $X^{-}_{\text{бзн}} = 2^n - |X|$ при $-2^n \leqslant X \leqslant -1$. Отметим, что диапазон представляемых чисел несимметричен: максимальное по модулю отрицательное число (-2^n) имеет модуль, на единицу больший максимального положительного числа $2^n - 1$.

Нуль в беззнаковой форме представляется единственным образом как положительная величина с нулевым кодом.

Формуле (2) соответствует график на рис. 4, обозначенный как I с ветвями для $X^{+}_{\text{бзн}}$ и $X^{-}_{\text{бзн}}$. Отмечаем, что код $X_{\text{бзн}}$ всегда является положительной

величиной независимо от представляемой величины X^+ и X^-.

Дополнение может вычисляться по следующему простому правилу: разряды модуля числа инвертируются, и к младшему разряду инверсии прибавляется единица. В качестве примера (для $n = 4$) укажем, что кодом 0000 представляется число $X = 0$, таким же кодом 0000 представляется число $X = -16$, задаваемое дополнением.

Недостатком беззнакового представления является то, что по коду числа нельзя определить знак представленной величины. От этого недостатка свободны *прямой*, *обратный* и *дополнительный* коды. Прямой код для n-разрядного формата определяется как

$$X_{\text{пр}} = \begin{cases} |X| & \text{при } 2^{n-1} - 1 \geqslant X \geqslant 0; \\ 2^{n-1} + |X| & \text{при } 0 \geqslant X \geqslant -(2^{n-1} - 1). \end{cases} \quad (3)$$

Этой формуле на рис. 4 соответствует график II ($X_{\text{пр}}$). Из него видно, что имеются два представления нуля: 00...00 для +0 и 10...00 для −0, что неудобно. Отмечаем, что по сути все прямые коды — положительные величины, ветви их графиков располагаются в I и IV квадрантах плоскости координат. Положительные числа имеют ноль в старшем $(n - 1)$-м разряде кода, а отрицательные — единицу в том же разряде. Это позволяет говорить о левом, $(n - 1)$-м, разряде как о знаковом и по его значению различать прямые коды положительных и отрицательных чисел (рис. 5).

Простота задания и естественность представления чисел прямыми кодами обеспечивает последним широкое применение в ЭВМ и системах (о недостатках см. ниже).

Использовавшийся в первых ЭВМ *обратный код* представлял числа со знаком так:

$$X_{\text{обр}} = \begin{cases} |X| & \text{при } (2^{n-1} - 1) \geqslant X \geqslant 0; \\ 2^n - 1 - |X| & \text{при } 0 \geqslant X \geqslant -(2^{n-1} - 1). \end{cases} \quad (4)$$

Положительные числа изображаются в двоичном виде своими модулями, а отрицательные — дополнениями модуля до 2^{n-1}. Так как число $2^n - 1$ представляется всеми единицами (n единиц), то при вычитании из него двоичного модуля $|X|$ получаются инверсии разрядов чисел, т. е. $\overline{|X|}$. В старшем разряде кода получается единица, так как в $(n - 1)$-м разряде модуля располагается нуль, ибо $|X| < 2^{n-1}$ (рис. 6).

При представлении чисел обратными кодами удобно, как и в случае прямых кодов, говорить о старшем, $(n - 1)$-м, разряде как о знаковом (плюс — ноль, минус — единица) и различать по нему коды положительных и отрицательных чисел. Отметим также, что диапазон представляемых чисел симметричен, а нуль, как и в случае прямых кодов, имеет два представления:

$$\boxed{+0} = \underbrace{00...00}_{n \text{ нулей}} \text{ и } \boxed{-0} = \underbrace{11...11}_{n \text{ единиц}}.$$

Дополнительный код является в настоящее время основным для представления в ЭВМ целых чисел:

$$X_{\text{доп}} = \begin{cases} |X| & \text{при } 2^{n-1} - 1 \geqslant X \geqslant 0; \\ 2^n - |X| & \text{при } -1 \geqslant X \geqslant -2^{n-1}. \end{cases} \quad (5)$$

Если $2^n - |X|$ представить как $(2^n - 1) - |X| + 1$, то появляется простое правило вычисления $X_{\text{доп}}$ для отрицательных чисел: инвертировать разряды модуля числа и прибавить к младшему разряду полученного кода единицу.

Для чисел, представленных дополнительным кодом (как и в случае прямого и обратного кодов), удобно использовать понятие знакового разряда, так как благодаря выбранной области представления значений X левый (знаковый) разряд для кодов положительных чисел всегда нуль, а для отрицательных — единица (рис. 7).

График $X_{\text{доп}}(X)$ представлен на рис. 4 — прямые IV ($X_{\text{доп}}$). Из него видно, что для положительных чисел $X_{\text{доп}}$ совпадает с графиками $X_{\text{пр}}$ и $X_{\text{обр}}$, а также с начальной частью графика для $X^+_{\text{бзн}}$. Для отрицательных чисел (от -2^{n-1} до -1) имеет место совпадение с графиком для $X^-_{\text{бзн}}$ в том же диапазоне.

Кроме рассмотренных в ЭВМ при представлении порядков в числах с плавающей точкой используются *смещенные* коды. Их получают прибав-

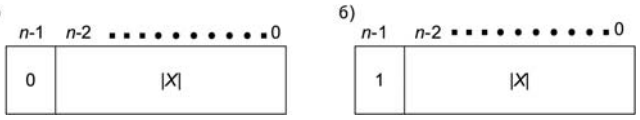


Рис. 5. Представление чисел со знаком прямыми кодами:
а — положительные числа; *б* — отрицательные числа

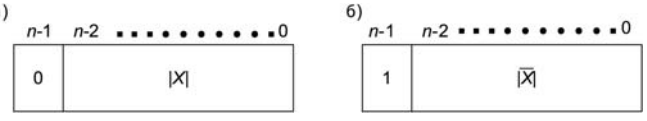


Рис. 6. Представление чисел со знаком обратными кодами:
а — положительные числа; *б* — отрицательные числа

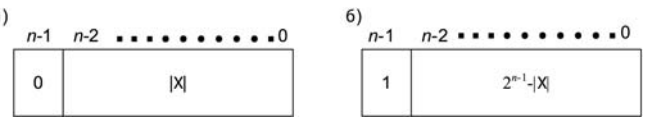


Рис. 7. Представление чисел со знаком дополнительными кодами:
а — положительные числа; *б* — отрицательные числа

лением к числу со знаком константы, называемой *смещением*. Исторически первым появился смещенный код, определяемый как

$$X_{\text{см}} = X + 2^{n-1} \text{ для } -2^{n-1} \leqslant X \leqslant 2^{n-1} - 1. \quad (6)$$

График $X_{\text{см}}(X)$ представлен на рис. 4 прямой $V(X_{\text{см}})$.

Смещенный код можно вычислять и так:

$$X_{\text{см}} = \begin{cases} 2^{n-1} + |X| & \text{для } 0 \leqslant X \leqslant 2^{n-1} - 1; \\ 2^{n-1} - |X| & \text{для } 2^{n-1} \leqslant X \leqslant -1. \end{cases}$$

При этом оказывается, что для $X \geqslant 0$ код имеет вид, представленный на рис. 8, *а*; для $X < 0$ в старшем разряде кода появляется нуль (рис. 8, *б*).

Это позволяет судить о знаке числа по значению старшего разряда кода. Нетрудно также заметить, что $X_{\text{см}}$ можно получать из $X_{\text{доп}}$ (и наоборот) инверсией старшего разряда. Это дает возможность в ряде случаев упрощать понимание различных свойств кодов.

В смещенном коде с отрицательным нулем используется другое смещение:

$$X'_{\text{см}} = X + 2^{n-1} - 1 \text{ для } -(2^{n-1} - 1) \leqslant X \leqslant 2^{n-1}. \quad (7)$$

Его график на рис. 4 представляется прямой VI ($X'_{\text{см}}$). Рассматриваемый код можно представить и так:

$$X'_{\text{см}} = \begin{cases} 2^{n-1} + (|X|) - 1 & \text{для } 1 \leqslant X \leqslant 2^{n-1}; \\ 2^{n-1} - 1 - |X| & \text{для } -(2^{n-1} - 1) \leqslant X \leqslant 0. \end{cases} \quad (8)$$

При этом оказывается, что смещенные коды с отрицательным нулем имеют вид, представленный на рис. 9.

Таким образом, старший разряд кода, как и в случае смещенного кода, говорит о знаке числа, относя, однако, нуль к отрицательным числам.

В заключение в иллюстративных целях приведем таблицу соответствия кодов целых чисел и самих чисел для $n = 4$.

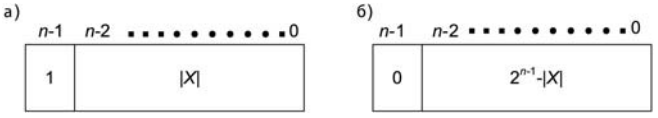


Рис. 8. Представление чисел со знаком смещенными кодами:
а — положительные числа; *б* — отрицательные числа

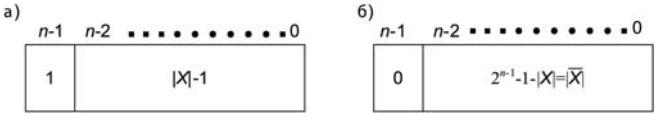


Рис. 9. Представление чисел смещенными кодами с отрицательным нулем:
а — положительные числа; *б* — отрицательные числа

Таблица соответствия чисел и кодов для $n = 4$

	Код числа	Число в коде						
		$X^{+}_{\text{бзн}}$	$X^{-}_{\text{бзн}}$	$X_{\text{пр}}$	$X_{\text{обр}}$	$X_{\text{доп}}$	$X_{\text{см}}$	$X'_{\text{см}}$
0	0000	0	−16	+0	+0	0	−8	−7
1	0001	1	−15	1	1	1	−7	−6
2	0010	2	−14	2	2	2	−6	−5
3	0011	3	−13	3	3	3	−5	−4
4	0100	4	−12	4	4	4	−4	−3
5	0101	5	−11	5	5	5	−3	−2
6	0110	6	−10	6	6	6	−2	−1
7	0111	7	−9	7	7	7	−1	−0
8	1000	8	−8	−0	−7	−8	0	1
9	1001	9	−7	−1	−6	−7	1	2
10	1010	10	−6	−2	−5	−6	2	3
11	1011	11	−5	−3	−4	−5	3	4
12	1100	12	−4	−4	−3	−4	4	5
13	1101	13	−3	−5	−2	−3	5	6
14	1110	14	−2	−6	−1	−2	6	7
15	1111	15	−1	−7	−0	−1	7	8

Свойства кодов и простейшие операции над числами в кодах

Особый интерес представляют ответы на вопросы о свойствах и возможностях кодов чисел и о том, как выполнять простейшие операции над числами, представленными теми или иными кодами. Все последующие рассмотрения проводятся без аргументирующих доказательств, а читателю предоставляется возможность самостоятельной проверки приведенных алгоритмов.

1. Определение модуля числа

Беззнаковое представление положительных чисел, прямой код для $X \geqslant 0$, обратный код для $X \geqslant 0$ и дополнительный код для $X > 0$

Здесь модулем числа являются все его разряды.

Беззнаковое представление отрицательных чисел и дополнительный код для $X < 0$

Для определения модуля числа все разряды кодов инвертируются, и к полученному коду прибавляется единица. Следует обратить внимание, что максимальное значение модуля в случае беззнакового кода равно 2^n. Оно не размещается в n-разрядной сетке и требует еще одного разряда. Его вид 10...00.

Прямой код для $X \leqslant 0$

Единичное значение старшего (знакового) разряда заменяется на 0.

Обратный код для $X \leqslant 0$

Все разряды кода инвертируются

Смещенный код

При X ⩾ 0 (об этом в смещенном коде говорит единичное значение разряда x_{n-1}) необходима замена x_{n-1} на 0 (инверсия x_{n-1}). В случае смещенного кода отрицательного числа (X < 0) проводят инвертирование всех разрядов кода, кроме старшего x_{n-1}, и прибавление 1.

Пример для $n = 4$: $X_{\text{см}} = 0101 \rightarrow 0010 + 1 = 0011$. $X = 3$ (см. таблицу).

Смещенный код с отрицательным нулем

При $X > 0$ (об этом в данном коде говорит единичное значение разряда x_{n-1}) необходима замена разряда x_{n-1} на ноль (иначе, инверсия этого разряда) и прибавление 1 к младшему разряду. В случае смещенного кода отрицательного числа и нуля ($X \leqslant 0$, $x_{n-1} = 0$) выполняется инвертирование всех разрядов, кроме старшего x_{n-1}.

2. Определение четности числа

Беззнаковое представление, прямой код, дополнительный код, смещенный код

Признаком четности числа, представленного такими кодами, выступает нулевое значение младшего разряда кода.

Обратный код

При $X \geqslant 0$ признаком четности числа является нулевое значение младшего разряда кода. При $X \leqslant 0$ таким признаком выступает единичное значение младшего разряда кода.

Смещенный код с отрицательным нулем

Признаком четности числа, представленного кодом $X_{\text{см}}$, является единичное значение младшего разряда кода.

3. Расширение разрядной сетки

В практике ЭВМ нередки случаи, когда n-разрядный код числа требуется представить кодом с большей разрядностью $-k + n$ (рис. 10).

Беззнаковое представление при $X \geqslant 0$, обратный код при $X \geqslant 0$ и дополнительный код при $X \geqslant 0$

Для получения расширенного кода k его разрядов заполняются нулями, т. е. к исходному коду слева приписывается k нулей.

Беззнаковое представление при $X < 0$, обратный код при $X \leqslant 0$ и дополнительный код при $X < 0$

Расширенный $(k + n)$-разрядный код образуется приписыванием к исходному коду слева K единиц.

Прямой код

Здесь значение разряда x_{n-1} перемещается в $(k + n - 1)$-й разряд, а в $k - 1$ разрядов справа от последнего заносятся нули согласно рис. 11.

Смещенный код и смещенный код с отрицательным нулем

Здесь правила образования расширенного кода одинаковы для обоих кодов: старший разряд x_{n-1}

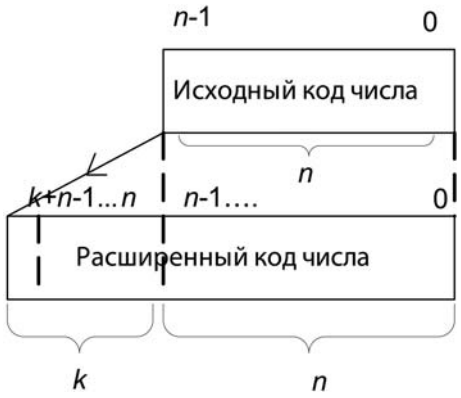


Рис. 10. Исходная и расширенная разрядные сетки (исходный и расширенный коды)

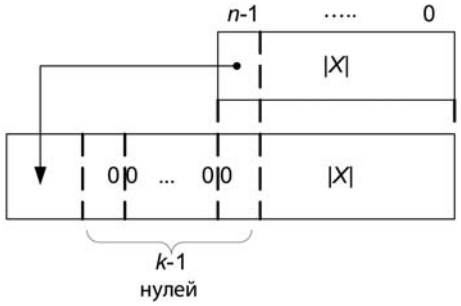


Рис. 11. Образование расширенного прямого кода

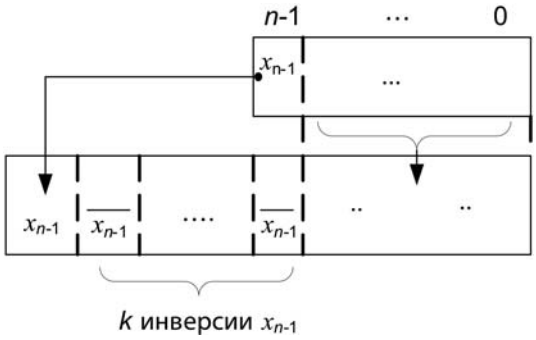


Рис. 12. К образованию расширенного смещенного кода

исходного кода перемещается в $(k + n - 1)$-й разряд, а в позициях $(k + n - 2)...(n - 1)$ помещаются инверсии x_{n-1} (рис. 12).

Пример. Смещенный код числа $X = -3$ при $n = 4$ равен $X_{см} = 0101$ (см. таблицу). При расширении разрядной сетки до 8 (k = 4) по вышеприведенному правилу получаем $X_{см.расш} = 01111101$.

4. Сравнение чисел по их кодам

Сравнивая два числа на больше, меньше, равно, можно выполнять вычитание этих чисел, пользуясь правилами алгебраического сложения для рассматриваемых кодов, а затем анализировать полученную разность. Однако можно сравнивать и иначе — с помощью процедуры вычитания кодов, рассматриваемых как целые положительные величины (модули). Такой подход наиболее удобен при представлении чисел смещенными кодами (как с положительным, так и с отрицательным нулем). При использовании этих кодов монотонный рост значений представляемых величин сопровождается монотонным ростом значений их кодов (графики на рис. 4 значений функции $X_{см}$ и $X'_{см}$ наглядно иллюстрируют это). В результате $X_{1см} \geqslant X_{2см}$ означает, что $X_1 \geqslant X_2$. Отсюда следует возможность сравнения чисел путем сравнения их смещенных кодов.

Интересно отметить, что благодаря этому, поставив в формате чисел с плавающей точкой код смещенного порядка перед кодом модуля мантиссы, мы получаем возможность сравнения нормализованных чисел с плавающей точкой путем сравнения их как кодов целых чисел. При использовании прямого, обратного и дополнительного кодов, а также при беззнаковом представлении сравнение кодов осуществляется путем предварительного выяснения знаков сравниваемых величин одного знака. При разных знаках чисел положительное рассматривается как большее. Для прямого и обратного кода необходимо дополнительно учитывать двойственное представление нуля.

5. Нахождение кода половинного числа

Беззнаковое представление чисел

Код половинного числа $\left(\frac{X}{2}\right)_{бзн}$ по коду числа $X_{бзн}$ получают сдвигом вправо всех разрядов кода и занесением в старший разряд x_{n-1} константы. Последняя равна нулю при $X_{бзн} = X^{+}_{бзн}$ и единице при $X_{бзн} = X^{-}_{бзн}$. Точное значение кода числа $\frac{X}{2}$ имеет место при делении пополам четных чисел. При нечетных X результат получается с недостатком.

Обратный код

Код половинного числа получают сдвигом вправо всех разрядов кода, включая знаковый. При $X \geqslant 0$ в разряд x_{n-1} заносится нуль, при $X \leqslant 0$ — единица (такой сдвиг иногда называют модифицированным сдвигом). Точное значение результата имеет место для четных X. При нечетных $X > 0$ результат получается с недостатком, а для $X < 0$ — с избытком.

Прямой код

Код половинного числа получают сдвигом вправо цифровых разрядов кода; знаковый разряд в искомом коде остается без изменения. Точное значение результата получается для четных X. Для нечетных X модуль половинного числа получается с недостатком.

Дополнительный код

При $X \geqslant 0$ вправо сдвигаются все разряды, включая знаковый, в который заносится 0. При $X < 0$ выполняется сдвиг вправо всех разрядов кода, включая знаковый, в который заносится единица. Для нечетных X половинное значение всегда получается с недостатком.

Смещенный код

Получение кодов половинного числа из числа X, представленного смещенным кодом и смещенным кодом с отрицательным нулем, осуществляется по одному и тому же правилу: происходит сдвиг вправо всех разрядов кода с записью инвертированного значения x_{n-1} в позицию $n - 2$ и сохранение значения старшего разряда x_{n-1} (рис. 13).

Точный результат получается так же, как и ранее для четных чисел X; для нечетных чисел величина $\frac{X}{2}$ вычисляется с недостатком для смещенного кода и с избытком для смещенного кода с отрицательным нулем.

Рис. 13. К получению $\left(\frac{X}{2}\right)_{см}$ из $X_{см}$ и $\left(\frac{X}{2}\right)'_{см}$ из $X'_{см}$

6. Нахождение кода удвоенного числа

При удвоении числа, а также при суммировании двух чисел результат может получить значение, выходящее за пределы диапазона значений используемого кода. В этом случае невозможно размещение полученного результата в используемой разрядной сетке, что квалифицируется как возникновение переполнения последней.

За переполнением разрядной сетки необходимо следить, так как, размещая результат, переполняющий разрядную сетку, мы получаем искаженный результат.

Беззнаковое представление чисел

Здесь код $(2X)_{\text{бзн}}$ получают путем сдвига всех разрядов кода $X_{\text{бзн}}$ влево с занесением нуля в разряд 0. Если при сдвиге $X^{+}_{\text{бзн}}$ из старшего разряда $n-1$ выходит 0, то результат правилен; если 1, то имеет место переполнение. При сдвиге $X^{-}_{\text{бзн}}$, наоборот, выходящая единица говорит о получении правильного результата, а нуль — о переполнении разрядной сетки.

Прямой код

Удвоение получается сдвигом влево всех разрядов кода, кроме знакового. Знаковый разряд остается неизменным, а в правый разряд, освобождающийся при сдвиге, заносится 0. Перенос из старшего цифрового разряда никуда не идет, но его единичное значение говорит о том, что результат удвоения переполняет разрядную сетку.

Дополнительный код

Удвоение получается сдвигом влево всех разрядов кода. В правый разряд, освобождающийся при сдвиге, заносится 0. Несовпадение значений переносов из старшего разряда и в старший говорит о том, что результат удвоения переполняет разрядную сетку. При этом положительный результат будет представлен модулем, а отрицательный — дополнением, но старший разряд не будет знаковым (полученный результат здесь будет представлен не в дополнительном коде, а в беззнаковом коде).

Обратный код

Подобно дополнительному коду удвоение получается сдвигом влево всех разрядов кода. Однако в правый разряд, освобождающийся при сдвиге, заносится не константа 0, а то значение, которое перед сдвигом находилось в старшем (знаковом) разряде. Такой сдвиг называют левым циклическим сдвигом. Несовпадение значений переносов из старшего разряда и в старший говорит о возникновении переполнения разрядной сетки. При этом положительный результат будет представлен модулем, а отрицательный — поразрядным дополнением (инверсией) модуля, но старший (левый) разряд не будет знаковым.

Смещенный код

Удвоение реализуется в два шага следующим образом. Сначала выполняется сдвиг влево всех разрядов кода с занесением нуля в правый разряд, освобождающийся при сдвиге. Если при сдвиге имеет место несовпадение значений переносов из старшего разряда и в старший разряд результата, то последний имеет значение, не переполняющее разрядную сетку. Для окончательного формирования результата необходимо инвертировать значение старшего разряда. При совпадении значений переносов результат удвоения переполняет разрядную сетку и не может быть представлен n-разрядным смещенным кодом. В этом случае результат может быть представлен $(n+1)$-разрядным кодом, старший, т. е. $(n+1)$-й, разряд совпадает со значением переноса, получившимся при сдвиге исходного кода. Последующие n разрядов кода результата совпадают с кодом, полученным при сдвиге.

Смещенный код с отрицательным нулем

Действия в данном случае подобны тем, которые выполняются для смещенных кодов. Единственное отличие заключается в том, что в освобождающийся при сдвиге правый разряд заносится не 0, а 1.

7. Суммирование (сложение) двух чисел

При суммировании двух чисел X и Y, как и при удвоении одного числа, возможно переполнение разрядной сетки, когда сумма $X+Y$ выходит за диапазон представления чисел. Понятно, что переполнение возникает только при сложении чисел одного знака.

Беззнаковое представление

При суммировании чисел, представленных в беззнаковой форме, возможны следующие ситуации:

$$\text{а) } X^{+}_{\text{бзн}} + Y^{+}_{\text{бзн}};\ \text{б) } X^{+}_{\text{бзн}} + Y^{-}_{\text{бзн}};$$

$$\text{в) } X^{-}_{\text{бзн}} + Y^{+}_{\text{бзн}};\ \text{г) } X^{-}_{\text{бзн}} + Y^{-}_{\text{бзн}}.$$

В ситуациях а) и г) возможно переполнение результата, а в б) и в) — необходимо определить, модулем или дополнением будет представлена вычисленная сумма.

Признаком переполнения при сложении модулей (ситуация а)) является единичное значение переноса из старшего разряда суммы C_n (от carry).

При нулевом его значении модуль суммы, т. е. $(X + Y)^{+}_{\text{бзн}}$, размещается в разрядной сетке.

При сложении дополнений (ситуация г)) признаком переполнения, наоборот, выступает нулевое значение такого переноса. При $C_n = 1$ сумма $(X + Y)^{-}_{\text{бзн}}$ размещается в разрядной сетке.

Ситуации б) и в) равноценны. Если при сложении $C_n = 0$, то сумма отрицательна и является дополнением $(X + Y)^{-}_{\text{бзн}}$. При $C_n = 1$ результатом является модуль $(X + Y)^{+}_{\text{бзн}}$.

Прямой код

Здесь сложение чисел выполняется достаточно непросто. При одинаковых знаках чисел X и Y выполняют сложение кодов $X_{\text{пр}}$ и $Y_{\text{пр}}$, включая знаковые разряды. Переполнение обнаруживается по единичному значению переноса в знаковый разряд (т. е. $C_{n-1} = 1$) либо по образовавшейся в знаковом разряде единице. Так получается модуль суммы, к которому приформировывается знак.

Сложнее обстоит дело при сложении чисел с разными знаками, т. е. при вычитании одного кода из другого. Обычно это выполняют так: прямой код отрицательного числа заменяют дополнительным и руководствуются правилами сложения чисел в дополнительных кодах (см. ниже). В случае образования отрицательного результата осуществляется переход от дополнительного кода к прямому (при положительном результате прямой и дополнительный код совпадают). Переполнение разрядной сетки тут не возникает.

Обратный код

Суммирование чисел в обратных кодах выполняется так: коды складываются по правилам двоичного сложения, включая знаковые разряды: $X_{\text{обр}} + Y_{\text{обр}}$. Если перенос из знакового разряда C_n равен 0, проверяется признак переполнения OVE (его образование рассматривается ниже). При OVE = 1 имеет место переполнение разрядной сетки. При OVE = 0 переполнения нет и $X_{\text{обр}} + Y_{\text{обр}}$ есть $(X + Y)_{\text{обр}}$. Если перенос C_n из знакового разряда равен 1, то он складывается с $X_{\text{обр}}$ и $Y_{\text{обр}}$: $X_{\text{обр}} + Y_{\text{обр}} + 1$.

Такое сложение с единицей переноса из старшего (знакового) разряда называют *циклическим*. После него проверяется признак переполнения OVE. Последний есть функция переноса из старшего разряда суммы C_n и переноса в знаковый разряд ее C_{n-1}:

$$\text{OVE} = C_n \overline{C}_{n-1} \vee \overline{C}_n C_{n-1}. \qquad (9)$$

Единичное значение этой булевой функции говорит о наличии переполнения, нулевое — об отсутствии последнего.

Дополнительный код

При сложении чисел, представленных дополнительным кодами, образуется сумма $X_{\text{доп}} + Y_{\text{доп}}$. Значения переноса из старшего (знакового) разряда C_n и переноса в знаковый разряд C_{n-1} определяют значение признака переполнения OVE в соответствии с формулой (9). Простота образования кода суммы чисел и однозначное представление нуля являются теми достоинствами, которые и объясняют распространенность дополнительного кода в большинстве ЭВМ и систем. Следует отметить, что в случае возникновения переполнения положительная сумма будет представлена модулем, а отрицательная — дополнением, однако старший (левый) разряд не будет знаковым (полученный результат будет представлен не в дополнительном коде, а в беззнаковом коде).

Смещенный код

Сложение чисел осуществляется по следующему правилу:

- смещенные коды чисел X и Y складываются друг с другом;
- сравниваются значения переноса из старшего разряда C_n и старшего $(n-1)$-го разряда суммы. Если они одинаковы, то фиксируется переполнение. Если различны, то переполнения нет, и смещенный код суммы получают, проинвертировав старший разряд.

Смещенный код с отрицательным нулем

Сложение чисел осуществляется по следующему правилу:

- получают сумму $(X)'_{\text{см}} + (Y)'_{\text{см}} + 1$;
- сравнивают значение переноса из старшего разряда C_n и значение старшего разряда суммы. Если они одинаковы, то фиксируется факт переполнения разрядной сетки. Если различны, то переполнение отсутствует. Смещенный код суммы получают, инвертируя старший разряд суммы.

8. Изменение знака числа

Для каждого положительного числа X существует единственное отрицательное $-X$, которое дополняет X до нуля: $X + (-X) = 0$. Нуль, строго говоря, единственен, но в компьютерных формах записи чисел со знаком он часто кодируется как число со знаком "+" или "−". В некоторых случаях (прямой и обратный коды) нуль имеет два представ-

ления. С учетом вышесказанного понятно, что процедура изменения знака числа, не равного нулю, может рассматриваться как переход от задания числа с модулем $|X|$ и некоторым знаком к числу с таким же модулем и противоположным знаком.

Беззнаковое представление

Результат получается путем инвертирования всех разрядов кода и последующего подсуммирования единицы к младшему разряду инверсии.

Несимметричность диапазонов представления положительных и отрицательных величин не позволяет для любого операнда осуществить смену знака. Невозможна смена для $X^+ = 0$ и для $X^- = -2^n$. Обнаружить факт невозможности смены знака позволяет анализ значения переноса из старшего разряда при подсуммировании единицы к инверсии. Единичное значение переноса говорит о невозможности фиксации результата.

Дополнительный код

Как и для беззнакового представления, получают инверсию всех разрядов, к которой подсуммируется единица к младшему разряду. Если исходный операнд $X = 0$, то его вид не изменяется из-за единственности представления нуля. Если исходный операнд $X = -2^{n-1}$, то результат операции переполняет разрядную сетку. Обнаружить переполнение можно по несовпадению значений переносов из старшего разряда суммы и в старший разряд.

Обратный код

Результат получается инверсией всех разрядов кода, причем такая обработка допустима для любого операнда.

Смещенный код

Действия подобны выполняемым для дополнительного кода. Отличие лишь в том, что обнаруживать переполнение удобно по единичному значению переноса из старшего разряда суммы.

Смещенный код с отрицательным нулем

Результат может быть получен путем инвертирования всех разрядов исходного кода и последующего подсуммирования кода, содержащего единицы во всех разрядах. При значении исходного операнда $X = 2^{n-1}$ результат операции переполняет разрядную сетку, что может обнаружить по нулевому значению переноса из старшего разряда суммы.

9. Вычитание чисел

Вычитание $X - Y$ реализуется однотипно при разных кодах для представления чисел со знаком. Первый операнд X остается в исходном виде, второй операнд Y инвертируется, а затем формируется сумма кодов X, $\overline{Y}$ и C, где C — константа 0 или 1. Процедуры суммирования и определения свойств результата совпадают с рассмотренными выше при суммировании двух чисел.

Беззнаковое представление
Дополнительный код
Смещенный код

Разность $(X - Y)$ получается суммированием кодов X, $\overline{Y}$ и единицы в младшем разряде суммы $(C = 1)$. Признак переполнения определяется так же, как и при получении $(X + Y)$. При использовании смещенного кода у результата суммирования инвертируется старший разряд суммы. При обработке чисел в беззнаковом виде анализ результата выполняется так, как будто выполнялось сложение X и Y, у которого знак изменен на противоположный.

Важно помнить, что суммировать X с предварительно полученным кодом $(\overline{Y}) + 1$ недопустимо (код $(\overline{Y}) + 1$ задает величину $(-Y)$, но, как известно, не всегда $(-Y)$ может помещаться в рассматриваемой разрядной сетке. Поэтому предварительное получение величины $(-Y)$ может заблокировать возможность получения $X - Y$).

Прямой код
Обратный код

Здесь разность получается как сумма операндов X и $(-Y)$. При этом код Y может быть изменен перед выполнением суммирования.

Смещенный код с отрицательным нулем

Разность $(X - Y)$ получается путем суммирования кодов X и $\overline{Y}$ $(C = 0)$. Признак переполнения тот же, что и в процедуре суммирования двух чисел. У результата суммирования, не переполняющего разрядную сетку, старший результат должен быть инвертирован.

Список литературы

1. **Ричардс Р.** Арифметические операции на цифровых вычислительных машинах. М.: ИЛ, 1957. 325 с.
2. **Карцев М. А.** Арифметика цифровых машин. М.: Наука, 1969. 597 с.
3. **Поспелов Д. А.** Арифметические основы вычислительных машин дискретного действия. М.: Высшая школа. 1970. 308 с.
4. **Каган Б. М.** Электронные вычислительны машины и системы. М.: Энергоатомиздат, 1991. 592 с.

УДК 004.272.2

Д. В. Тельпухов, аспирант, инж.-исслед.,
Институт проблем проектирования
в микроэлектронике РАН,
e-mail:Nofrost@inbox.ru

Построение обратных преобразователей модулярной логарифметики для устройств цифровой обработки сигналов

Дается определение нового аппарата модулярной логарифметики. Изучается актуальная проблема реализации одной из основных немодульных операций — перевода чисел из LG-кода модулярной логарифметики в традиционную систему счисления. Описан высокоэффективный метод конвейерного преобразования из двоичного представления в LG-код, ориентированный на задачи цифровой обработки сигналов.

Ключевые слова: *модулярная логарифметика, LG-код, обратный преобразователь, цифровая обработка сигналов, конвейерная структура*

Введение

Отличительной чертой задач цифровой обработки сигналов (ЦОС) является поточный характер обработки больших объемов данных в реальном масштабе времени, требующий от технических средств высокой производительности и возможности интенсивного обмена с внешними устройствами. Это достигается в настоящее время благодаря специфической архитектуре процессоров ЦОС, называемой базовой архитектурой. Базовая архитектура — это совокупность характерных особенностей процессора, направленная на повышение его производительности и отличающая процессоры ЦОС от микросхем других типов. В значительной мере базовая архитектура обусловлена как широким использованием конвейерного режима работы [1], так и параллельными структурами обработки информации [2].

Идея конвейерной обработки заключается в выделении отдельных этапов или стадий выполнения общей операции. Причем данные, обработанные на каждой стадии, передаются на следующую, одновременно с поступлением новой порции входных данных. Получаем очевидный выигрыш в скорости обработки за счет совмещения прежде разнесенных во времени операций. Предположим, что в операции можно выделить пять микроопераций, каждая из которых выполняется за одну единицу времени. Если есть одно неделимое последовательное устройство, то 100 пар аргументов оно обработает за 500 единиц. Если каждую микрооперацию выделить в отдельный этап (или иначе говорят — ступень) конвейерного устройства, то на пятой единице времени на разной стадии обработки такого устройства будут находиться первые пять пар аргументов, а весь набор из ста пар будет обработан за $5 + 99 = 104$ единицы времени — ускорение по сравнению с последовательным устройством почти в пять раз (по числу ступеней конвейера).

В то время как конвейеризация алгоритмов ЦОС достигается разработкой различных конвейерных архитектур, нужный уровень *параллелизма* может быть достигнут благодаря использованию аппарата модулярной арифметики. Модулярная арифметика позволяет проводить распараллеливание трактов обработки данных без какого-либо изменения существующих технологий. Однако такие важные операции машинной арифметики, как деление, формирование признака переполнения, округление, перевод из одной системы счисления в другую, а также алгоритмы декодирования системы самокоррекции являются последовательно-параллельными. Перечисленные операции принято называть немодульными. Немодульные операции являются основным тормозящим фактором как в модулярной арифметике, так и в модулярной логарифметике, являющейся расширением индексной арифметики поля $GF(p)$ на сингулярную точку (т. е. нулевую точку поля $GF(p)$). Основными немодульными операциями являются операции перевода чисел из традиционной системы счисления в LG-код модулярной логарифметики и обратно. Способы построения прямых преобразователей модулярной логарифметики были рассмотрены в статье [3], в то время как данная работа посвящена вопросам построения преобразователей из LG-кода модулярной логарифметики в позиционный двоичный код.

Система остаточных классов и индексные вычисления

Система остаточных классов [4] основана на отношении конгруэнтности, которое определяется следующим образом. Два целых числа a и b сравнимы по модулю m, если m делит разность a и b без остатка. Обычно это обозначают как $a \equiv b (\mathrm{mod}\, m)$. Таким образом, например, $10 \equiv 7 (\mathrm{mod}\ 3)$, $10 \equiv 4 (\mathrm{mod}\ 3)$, $10 \equiv 1 (\mathrm{mod}\ 3)$ и $10 \equiv -2 (\mathrm{mod}\ 3)$. Число m называется **модулем**, и будем считать, что значения модуля исключают единицу.

Если q и r — частное и остаток, соответственно, от деления целого a на модуль m так, что $a = qm + r$. Тогда, по определению, мы имеем $a \equiv r (\mathrm{mod}\ m)$. Число r называется **вычетом** числа a по модулю m

и, как правило, обозначается $r = |a|_m$. Набор из m наименьших значений {0, 1, 2, ..., $m - 1$}, которые может принять вычет r, называется полной системой наименьших вычетов по модулю m. Будем считать, что r принимает только эти значения, если обратное не оговорено заранее.

Предположим, у нас есть набор $\{m_1, m_2, ..., m_N\}$ N положительных попарно взаимно простых модулей. Пусть M — произведение модулей. Тогда любое число $X < M$ может быть представлено единственным образом в системе остаточных классов, как набор остатков по соответствующим модулям $\{|X|_{m_i} : 1 \leqslant i \leqslant N\}$. Арифметические операции над двумя операндами X и Y определены как $Z = X \circ Y$, где $X = (x_1, x_2, ..., x_r)$, $Y = (y_1, y_2, ..., y_r)$, $Z = (z_1, z_2, ..., z_r)$, и $z_i = x_i \circ y_i$ для $i = 1,..., r$. Символ $\circ$ обозначает любую из модульных операций сложения, вычитания или умножения по соответствующим модулям. Из предыдущего определения видно, что операции реализуются параллельно по каждому остаточному каналу, независимо друг от друга. Этот врожденный параллелизм и арифметика без переносов между остаточными каналами обеспечивают ускорение во время арифметических действий.

При построении модулярных умножителей часто пользуются индексным или дискретно-логарифмическим представлением операндов и заменяют операцию модулярного умножения операцией модулярного сложения. Отображение операции умножения двух ненулевых вычетов u_k и u_j по простому модулю m на операцию сложения по модулю $(m - 1)$ осуществляется по формуле

$$|u_k \cdot u_j|_m \Leftrightarrow w^{|v_k + v_j|_{m-1}}, \qquad (1)$$

где w — первообразный корень кольца; v_m — "дискретный логарифм" вычета $u \in Z_p^*$ по основанию w.

Дальнейшее развитие парадигмы индексных вычислений приводит к идее полностью отказаться на уровне модульных операций от аддитивных вычислений и перейти к мультипликативным (т. е. логарифметическим) [5]. Для достижения этой цели необходимо расширить область определения на сингулярную (нулевую) точку, тогда модулярная логарифметика кольца вычетов по составному модулю N позволяет реализовать все арифметические операции над элементами кольца вычетов посредством дискретного логарифмирования.

Умножение двух ненулевых вычетов, как было показано ранее (1), заменяется сложением дискретных логарифмов, в то время как их сложение в логарифметике выполняется по правилу

$$|x + y|_p = |w^{\log_w |x|_p}(1 + w^{|\log_w |y|_p - \log_w |x|_p|_{p-1}})|_p = \\ = |w^{\log_w |x|_p + J_w(|\log_w |y|_p - \log_w |x|_p|_{p-1})})|_p,$$

где $J_w(|u|_{p-1}) = \log_w |1 + w^{|u|_{p-1}}|_p$ — так называемый логарифм Якоби [6].

Способы построения обратных преобразователей модулярной логарифметики

Важным аспектом при построении модулярных вычислителей является выбор базисных модулей. Помимо прочего, этот выбор влияет на архитектуру немодульных устройств, в частности прямых и обратных преобразователей. Некоторые авторы предлагают использовать наборы модулей специального вида, для того чтобы получить более простые и эффективные архитектуры обратных преобразователей, например модули вида $(2^n - 1, 2^n, 2^n + 1)$. Естественно, этот подход сокращает возможности модулярной арифметики (максимальные преимущества достигаются при использовании большого числа небольших модулей). Фактически n растет вместе с требуемым динамическим диапазоном и, соответственно, производительность всего модулярного процессора понижается. Вместе с тем, высокоскоростные маломощные мультимедийные приложения требуют устройств ЦОС с большим динамическим диапазоном и хорошей гранулярностью разбиения (этот аспект относится к оптимизации разрядности модулей). Очевидно, эти требования не могут быть полностью удовлетворены с помощью модулей специального вида. Для того чтобы преодолеть эти проблемы, в большинстве приложений требуются произвольные наборы модулей. Выбор конкретных наборов может быть связан с оптимизацией разрядности модулей, а также с минимизацией бивалентного дефекта [7].

Приложения модулярной логарифметики ограничены кругом вычислительно сложных задач, с преобладающим числом модульных операций сложения, и в особенности умножения, и небольшим числом операций округления и сравнения по величине. В свете этого, наилучшей областью применения аппарата модулярной логарифметики представляется цифровая обработка сигналов [8]. Планируется использовать модулярные логарифмические блоки для ускоренной реализации отдельных задач в составе устройств ЦОС. Исходя из сказанного выше, можно обозначить три возможных месторасположения разрабатываемого модулярного блока относительно аналого-цифровых и цифроаналоговых преобразователей (рис. 1).

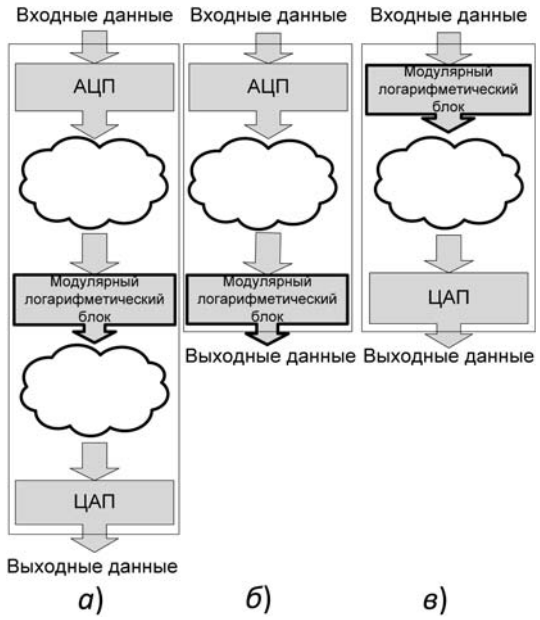


Рис. 1. Возможные месторасположения разрабатываемого модулярного блока

В первом случае (рис. 1, *а*), когда модулярный логарифметический блок находится внутри тракта обработки информации, в качестве интерфейса он должен использовать преобразователи из двоичного представления в *LG*-код и обратно. Однако, если данный блок находится в начале тракта обработки (рис. 1, *в*), на который подается аналоговый сигнал, или же выполняет заключительную операцию перед выводом сигнала в аналоговой форме (рис. 1, *б*), то появляется возможность использовать преобразователи аналог—*LG*-код и *LG*-код—аналог, соответственно. В настоящее время в ИППМ РАН ведутся разработки таких преобразователей. Они позволяют сократить накладные расходы на последовательную реализацию двух преобразователей аналого-двоичного и двоично-модулярного путем их интеграции. В данной статье мы более подробно коснемся случаев (рис. 1, *а* и *в*), а именно, рассмотрим возможные структуры построения обратных преобразователей модулярной логарифметики и предложим несколько высокоэффективных методов, ориентированных на задачи ЦОС.

После прохождения некоторых данных через прямой преобразователь и выполнения определенного набора операций над ними в модулярном логарифметическом устройстве результаты попадают на обратный преобразователь. Под обратным преобразователем понимается устройство, переводящее данные из логарифметического представления остатков обратно в двоичную систему счисления. Вообще говоря, в СОК процесс обратного преобразования — это одна из наиболее трудоемких операций и является ограничивающим фактором для более широкого использования модулярной арифметики. Обратный преобразователь модулярного логарифметического устройства состоит из преобразователя индексного представления остатков в классическое представление и преобразователя СОК в двоичную систему счисления.

Остановимся подробнее на второй составляющей преобразователя, так как получение остатков из индексов реализуется просто с использованием просмотровых таблиц.

Все многообразие архитектур обратных преобразователей базируется на китайской теореме об остатках и на преобразовании в полиадический код. Все другие методы являются просто комбинацией первых двух. Устройства на базе китайской теоремы об остатках обладают большим параллелизмом, чем устройства на базе перевода в полиадический код, однако необходимость выполнения операции мультиоперандного суммирования по потенциально большому модулю сводит на нет это преимущество. К тому же возможность конвейерной реализации обратных преобразователей на базе полиадической системы счисления делает этот метод наиболее подходящим для задач ЦОС.

Построение обратного преобразователя модулярной логарифметики на основе полиадической системы счисления

Полиадическая система счисления является позиционной системой счисления со смешанным основанием. Она является взвешенной системой счисления, что облегчает выполнение операций (таких как отношение порядка), которые являются трудновыполнимыми в системе остаточных классов. Пусть основания полиадической системы счисления r_N, r_{N-1}, ..., r_1, тогда любое число X может быть единственным образом представлено в форме:

$$X \cong (z_N, z_{N-1}, z_{N-2}, ..., z_1), \text{ где } 0 \leqslant z_i < r_i,$$

$$X = z_N r_{N-1} r_{N-2} \ldots r_1 + \ldots + z_3 r_2 r_1 + z_2 r_1 + z_1. \quad (2)$$

Ограничение на z_i гарантирует однозначное представление числа. Покажем, как получить разряды z_i и таким образом преобразовать число в традиционное двоичное представление.

Для того чтобы обеспечить взаимосвязь между системой со смешанным основанием и представлением в остатках $(x_N, x_N - 1, \ldots, x_1)$ по базисным модулям $m_1, m_2, \ldots, m_N$, возьмем основания полиадической системы, равными модулям системы остаточных классов. Таким образом

$$X = z_N m_{N-1} m_{N-2} \ldots m_1 + \ldots + z_3 m_2 m_1 + z_2 m_1 + z_1.$$

Возьмем обе части уравнения по модулю m_1:

$$|X|_{m_1} = z_1 = x_1.$$

Для того чтобы найти z_2, перепишем уравнение в следующем виде:

$$X - z_1 = z_N m_{N-1} m_{N-2} \ldots m_1 + \ldots + z_3 m_2 m_1 + z_2 m_1.$$

Теперь возьмем обе части этого уравнения по модулю m_2:

$$|X - z_1|_{m_2} = |z_2 m_1|_{m_2}.$$

Умножение обеих частей уравнения на $|m_1^{-1}|_{m_2}$ приводит к

$$\|m_1^{-1}|_{m_2}(X - z_1)|_{m_2} = |z_2|_{m_2} = z_2, \text{ так как } z_2 < m_2.$$

Таким образом, поскольку $|X - z_1|_{m_2} = \left||X|_{m_2} - |z_1|_{m_2}\right|_{m_2} = |x_2 - z_1|_{m_2}$ $(z_1 < m_2)$, то уравнение для z_2 будет выглядеть следующим образом:

$$z_2 = \|m_1^{-1}|_{m_2}(x_2 - z_1)|_{m_2}.$$

Применяя такой же алгоритм для нахождения z_3, получим

$$z_3 = \|(m_2 m_1)^{-1}|_{m_3}(x_3 - (z_2 m_1 + z_1))|_{m_3}.$$

Продолжая этот процесс, значения разрядов полиадической системы z_i можно получить из остатков:

$$z_1 = x_1;$$

$$z_2 = \|m_1^{-1}|_{m_2}(x_2 - z_1)|_{m_2};$$

$$z_3 = \|(m_1 m_2)^{-1}|_{m_3}(x_3 - (z_2 m_1 + z_1))|_{m_3};$$

$$\vdots$$

$$z_N = \|(m_1 m_2 \ldots m_{N-1})^{-1}|_{m_N}(x_n - (z_{N-1} m_{N-2} \ldots \ldots m_1 + \ldots + z_2 m_1 + z_1))|_{m_N}.$$

Инверсные произведения базовых модулей, используемые в алгоритме, являются константами, могут быть заранее рассчитаны и хранятся в памяти устройства.

Последний набор формул четко демонстрирует, что метод, основанный на переводе в полиадическую систему счисления, является полностью последовательным: для вычисления z_i необходимы значения $z_{i-1} \ldots z_1$. Вместе с тем в методе, основанном на китайской теореме об остатках, частные суммы $X_j s$ (аналог $z_j s$) могут вычисляться параллельно.

Однако метод перевода в полиадическую систему счисления также позволяет несколько распараллелить структуру за счет некоторых модификаций алгоритма. Воспользуемся правилом

$$\left||m_i^{-1}|_{m_k}|m_j^{-1}|_{m_k}\right|_{m_k} = |(m_i m_j)^{-1}|_{m_k}. \qquad (3)$$

Теперь мы можем переписать уравнения для всех значений z_i, $i = 3 \ldots N$:

$$z_3 = \left|\|m_2^{-1}|_{m_3}|m_1^{-1}|_{m_3}\right|_{m_3}(x_3 - (z_2 m_1 + z_1))|_{m_3} =$$

$$= \|m_2^{-1}|_{m_3}(|m_1^{-1}|_{m_3}(x_3 - z_1) - z_2)|_{m_3};$$

$$z_4 = \|m_3^{-1}|_{m_4}(|m_2^{-1}|_{m_4}(|m_1^{-1}|_{m_4}(x_4 - z_1) - z_2) - z_3)|_{m_4};$$

$$\vdots$$

$$z_N = \|m_{N-1}^{-1}|_{m_N}(|m_{N-2}^{-1}|_{m_N}(\ldots|m_2^{-1}|_{m_N}(|m_1^{-1}|_{m_N} \times$$

$$\times (x_N - z_1) - z_2)\ldots) - z_{N-1})|_{m_N}.$$

Рассмотрим для примера процесс вычисления z_3. При использовании прямого метода, как мы уже заметили, потребуется последовательно вычислять значения z_1, z_2 и затем z_3. Однако анализируя модифицированные уравнения, можно наблюдать некоторый параллелизм: вычисление $|m_1^{-1}|_{m_3}(x_3 - z_1)$ может быть начато, как только будет доступно число z_1. Еще одно отличие модифицированного метода заключается в количестве мультипликативных инверсий, используемых в алгоритме.

Архитектура обратного преобразователя на основе полиадического представления для пяти модулей представлена на рис. 2. В данной архитектуре предполагается использование двухвходовой памяти, реализующей произведение разности двух чисел на константу, в роли которой выступает мультипликативное инверсное по соответствующему модулю. Также в схеме присутствует обычная па-

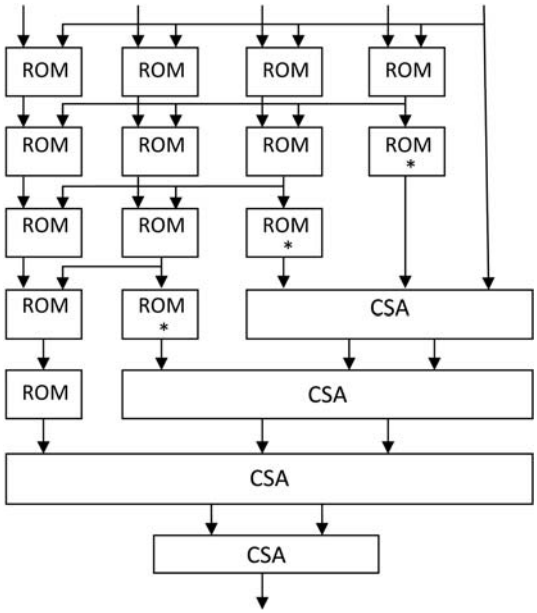


Рис. 2. Обратный преобразователь на основе полиадического представления: ROM (read-only memory) — одно- и двухвходовые ячейки памяти; CSA (Carry-Save-Adder) — сумматор с сохранением переноса

Результаты синтеза обратных преобразователей

Используемые наборы модулей	Динамический диапазон	Тактовая частота, МГц	Число занятых макроячеек (Alut)	Число регистров (Reg)
{2, 3, 5}	30	>500	15	18
{2, 3, 5, 7}	210	>500	32	36
{2, 3, 5, 7 11}	2310	481,46	88	70
{2, 3, 5, 7, 11, 13}	30 030	440,92	162	107
{2, 3, 5, 7, 11, 13, 31}	930 930	315,16	459	159
{2, 3, 5, 7, 11, 13, 31, 61}	56 786 730	241,02	1373	240
{2, 3, 5, 7, 11, 13, 31, 61, 103}	5 849 083 190	162,60	3956	322
{31, 61, 103}	194 773	193,57	1807	57

мять, реализующая операцию умножения, и дерево сумматоров с сохранением переноса для перевода из системы счисления со смешанным основанием в стандартное двоичное представление. Важно отметить, что использование памяти позволяет объединить получение вычета по индексу с нахождением $\left|m_i^{-1}\right|_{m_j}(x_j - z_i)$ и таким образом исключить из архитектуры преобразователь индексного представления остатков в классическое представление.

В завершении статьи представим результаты некоторых исследований спроектированных обратных преобразователей модулярной логарифметики. Используя методологию, описанную в работе [9], был реализован автоматизированный генератор функциональных описаний, позволяющий создавать *RTL*-описания обратных преобразователей для заданных наборов модулей на языке *verilog*. Синтез проводился с помощью САПР *Altera Quartus II* для ПЛИС *Stratix_II EP2S15F484C3*. Анализировались аппаратные затраты, характеризующиеся числом занятых регистров и макроячеек в ПЛИС, а также тактовая частота работы устройства как показатель временных затрат на реализацию обратного преобразования. Результаты исследования сведены в таблицу. Анализируя полученные данные, можно сделать несколько выводов:

- При наращивании динамического диапазона путем последовательного добавления базисных модулей аппаратные затраты увеличиваются в соответствии с разрядностью добавляемого модуля.
- Частота, как и предполагалось, в большей степени зависит от значения самого большого модуля, в то время как число тактов зависит от числа модулей. Это объясняется конвейерной структурой преобразователя.
- Сравнивая временные оценки устройств ЦОС, полученные в тех же условиях, можно сделать вывод о том, что полученная структура обратного преобразователя позволяет строить модулярные устройства, превосходящие по производительности свои двоичные аналоги.

Заключение

В рамках проведенного исследования был описан аппарат модулярной логарифметики, являющийся расширением индексной арифметики поля $GF(p)$ на сингулярную точку. Также были рассмотрены различные способы построения обратных преобразователей модулярной логарифметики и предложен конвейерный метод, ориентированный на задачи ЦОС. Кроме того, был построен автоматизированный генератор функциональных представлений для реализации *RTL*-описаний обратных преобразователей модулярной логарифметики и проведены некоторые исследования полученных моделей. Анализ полученных результатов показал эффективность и гибкость данного метода для использования его в составе модулярных устройств, реализующих процедуры ЦОС.

Список литературы

1. **He S., Torkelson M.** A New Approach to Pipeline FFT Processor // Proc. 10th International Parallel Processing Symposium (IPPS'96). 1996. P. 766—770.
2. **Шпаковский Г. И.** Параллельные микропроцессоры для цифровой обработки сигналов и медиаданных. Минск: БГУ, 2000. 196 с.
3. **Амербаев В. М., Тельпухов Д. В., Балака Е. С., Константинов А. В.** Методы построения прямых преобразователей модулярной логарифметики, ориентированных на ЦОС // Проблемы разработки перспективных микро- и наноэлектронных систем — 2010. Сб. научных трудов / Под общ. ред. А. Л. Стемпковского. М.: ИППМ РАН, 2010. С. 374—377.
4. **Акушский И. Я., Юдицкий Д. И.** Машинная арифметика в остаточных классах. М.: Сов. радио, 1968. 440 с.
5. **Arnold M. G.** Residue Logarithmic number System. Theory and Implementation // Proceedings Computer Arithmetic, 2005. (ARITH-17 2005). 17th IEEE Symposium. 2005. P. 186—205.
6. **Лидл Р., Нидеррайтер Г.** Конечные поля: в 2 т. / Под общ. ред. В. И. Нечаева. М.: Мир, 1988.
7. **Амербаев В. М., Константинов А. В., Тельпухов Д. В.** Бивалентный дефект модулярных кодов // Проблемы разработки перспективных микро- и наноэлектронных систем — 2008: Сб. научных трудов / Под общ. ред. А. Л. Стемпковского. М.: ИППМ РАН, 2008. С. 462—466.
8. **Amos Omondi, Premkumar A. B.** Residue Number Systems: Theory and Impemetation. London: Imperial College Press, 2007. 310 p.
9. **Корнилов А. И., Семенов М. Ю., Ласточкин О. В., Калашников В. С.** Методология проектирования специализированных вычислителей на основе автоматизированной генерации технологически независимых IP-блоков // Проблемы разработки перспективных микроэлектронных систем — 2005. Сб. научных трудов / Под общ. ред. А. Л. Стемпковского. М.: ИППМ РАН, 2005. С. 487—492.

УДК 519.25+004.9

А. Г. Трофимов, канд. техн. наук, доц.,
e-mail:atrofimov@list.ru
В. И. Скругин, аспирант,
e-mail:goodthings@yandex.ru
Национальный исследовательский
ядерный университет "МИФИ", г. Москва

Метод выделения динамических паттернов в задаче классификации многомерных временных рядов

Рассматривается задача классификации многомерных динамических данных. Излагается метод формирования вектора характерных признаков многомерного сигнала, основанный на выделении динамических паттернов. Приводятся результаты экспериментальных исследований метода при решении прикладной задачи распознавания сигналов электроэнцефалограмм.

Ключевые слова: *многомерный временной ряд, классификация, динамический паттерн, количественная электроэнцефалография*

Введение

Задачи анализа и классификации многомерных динамических данных получили широкое распространение в различных областях науки и техники, среди которых медицинская диагностика, распознавание речи, идентификация динамических систем, эконометрика и др. Существующие методы цифровой обработки сигналов преимущественно ориентированы на анализ и обработку одномерных временных рядов, в то время как многомерным динамическим данным уделяется существенно меньшее внимание. Методы анализа и классификации, ориентированные на одномерные сигналы, зачастую оказываются неэффективными для многомерных данных или вообще не могут быть обобщены на многомерный случай. Как правило, эти методы обработки сводятся к использованию спектрально-корреляционных преобразований или построению авторегрессионных моделей [1—3].

Построение классификатора временных рядов предполагает выполнение следующих этапов:

- *Предобработка сигналов*. На этом шаге осуществляется фильтрация сигналов от шумов, устранение артефактов или другие операции, позволяющие на ранних этапах отсеять неинформативные данные, используя априорно известную информацию о природе сигнала.
- *Выделение характерных признаков сигнала*. Целью данного этапа является переход от исходного пространства динамических данных к пространству многомерных статических данных. Как только такой переход сделан, задача классификации сигналов сводится к известной задаче классификации многомерных данных.
- *Классификация в пространстве характерных признаков сигналов.* На этом этапе могут использоваться классические алгоритмы классификации многомерных статических данных, например, метод дискриминантных функций, нейросетевой аппарат и др.

Наибольшее внимание в данной статье уделяется проблеме выделения характерных признаков сигнала для классификации.

Динамическая система, продуцирующая многомерный временной ряд, может находиться в различных функциональных состояниях или работать в различных режимах. Работа системы в том или ином функциональном состоянии находит отражение в наблюдаемых сигналах в виде паттернов. Как правило, моментам смены функционального состояния системы соответствуют моменты изменения динамических характеристик наблюдаемого временного ряда, т. е. моменты смены паттернов.

В данной статье предлагается метод расчета характерных признаков многомерных сигналов, в том числе нестационарных, основанный на выделении паттернов, соответствующих различным функциональным состояниям динамической системы, продуцирующей временной ряд. Характеристики этих паттернов используются в качестве признаков для классификации.

Рассматривается практическое применение предлагаемого метода для анализа и классификации многомерных нестационарных сигналов электроэнцефалограмм (ЭЭГ). Классификатор сигналов ЭЭГ может являться основой телекоммуникационных систем нового поколения — интерфейсов "мозг—компьютер" — систем, позволяющих осуществлять управление внешними техническими устройствами непосредственно с помощью электрической активности нейронов мозга, минуя мышцы [4].

1. Постановка задачи

В связи с тем, что обработка данных осуществляется преимущественно в рамках классических ЭВМ, будем рассматривать либо сигналы, имеющие дискретную природу, либо дискретизованные. Значение многомерного сигнала в момент времени t является вектором вида

$$x(t) = (x_1(t), x_2(t), ..., x_L(t)) \in X,\ t = \overline{1, T}, \quad (1)$$

где $t = \overline{1, T}$ — номер такта дискретного времени; T — число дискретных отсчетов; L — размерность сигнала; $x_l(t)$ — значение l-го компонента сигнала $\{x(t),\ t = \overline{1, T}\}$ в момент времени t; $l = \overline{1, L}$, $X \in R^L$ — множество значений временного ряда.

Пусть в результате P наблюдений получено множество многомерных временных рядов в общем случае различной длины, каждый из которых отнесен к одному из K классов. Таким образом, каждый класс представлен конечным числом сигналов. Обозначим P_k — число наблюдений, относящихся к k-му классу, $k = \overline{1, K}$. Таким образом, общее число рассматриваемых временных рядов $P = \sum_{k=1}^{K} P_k$.

Значение временного ряда, относящегося к k-му классу, полученное в результате p-го наблюдения в момент времени t, будем обозначать

$$x^{k,p}(t) = (x_1^{k,p}(t), x_2^{k,p}(t), ..., x_L^{k,p}(t)) \in X,$$
$$t = \overline{1, T_p^k},\ p = \overline{1, P_k},\ \mathrm{k} = \overline{1, K}, \quad (2)$$

где T_p^k — число дискретных отсчетов p-го сигнала k-го класса.

Классификатор сигналов запишем как оператор, сопоставляющий сигналу $\{x(t),\ t = \overline{1, T}\}$ номер класса k, к которому он относится:

$$C[\{x(t),\ t = \overline{1, T}\}] = k,\ k \in \{1, 2, ..., K\}. \quad (3)$$

Задача построения классификатора временных рядов, таким образом, сводится к задаче нахождения оператора C на основе наблюдений вида (2), для которых результат действия этого оператора, т. е. номер класса, известен. Практически все методы классификации временных рядов предполагают сведение задачи классификации динамических данных к задаче классификации статических данных. Для такого сведения необходимо осуществить переход от временного ряда $\{x(t),\ t = \overline{1, T}\}$ к некоторому вектору признаков z размерности M, характеризующему этот временной ряд. Обозначая оператор этого перехода Ψ, запишем:

$$\Psi[\{x(t),\ t = \overline{1, T}\}] = z,\ z \in R^M. \quad (4)$$

Основная проблема состоит в том, что не существует единого алгоритма такого перехода и в каждом конкретном случае исследователь сталкивается с необходимостью поиска оператора Ψ и пространства признаков для эффективной классификации. Особенно остро эта проблема встает для случая нестационарных временных рядов.

Учитывая формулы (3) и (4), можно считать, что задача классификации временных рядов состоит в поиске операторов Ψ и c:

$$\begin{cases} \Psi[\{x(t),\ t = \overline{1, T}\}] = z,\ z \in R^M; \\ c[z] = k,\ k \in \{1, 2, ..., K\}. \end{cases} \quad (5)$$

Отметим, что задача нахождения этих операторов на основе результатов наблюдений является неоднозначной. Неточность в оценке оператора Ψ может быть компенсирована за счет увеличения сложности оператора c, что, однако, может привести к ухудшению обобщающих способностей классификатора.

Качество построенного классификатора напрямую связано с его обобщающими способностями, которые могут быть оценены по результатам классификации данных тестовой выборки, не использовавшихся при построении операторов Ψ и c.

2. Алгоритм выделения характерных признаков многомерного сигнала на основе анализа динамических паттернов

2.1. Выделение моментных характерных признаков сигнала. Оператор Ψ в модели (5) ставит в соответствие временному ряду $\{x(t),\ t = \overline{1, T}\}$ некоторый вектор характерных признаков z, не зависящий от времени t. Это означает, что все интересующие особенности динамики значений временного ряда на рассматриваемом интервале времени должны быть интегрированы в конечное множество статических показателей. Эти характерные признаки могут быть рассчитаны как по исходным значениям временного ряда, так и по некоторым производным показателям, возможно, лучше отражающим эти особенности.

Введем оператор φ, определенный на множестве рассматриваемых временных рядов вида (1) и возвращающий в каждый момент времени t, $t = \overline{1, T}$, вектор, характеризующий рассматриваемый временной ряд в этот момент:

$$\varphi[\{x(\tau),\ \tau = \overline{1, T}\},\ t] = v(t),\ v(t) \in V,\ t = \overline{1, T}. \quad (6)$$

Вектор $v(t)$ будем называть вектором *моментных характерных признаков* (МХП).

Цель перехода от пространства X значений исходного временного ряда к пространству МХП V состоит во введении в рассмотрение дополнительных параметров исходного временного ряда, характеризующих его динамические свойства в каждый момент времени. Выбор конкретного типа

оператора φ осуществляется с учетом специфики анализируемых данных. Так, для сигналов гармонической природы в качестве элементов вектора МХП $v(t)$ могут быть выбраны значения спектральной плотности мощности, рассчитанные по результатам оконного преобразования Фурье с центром окна в момент времени t; в задачах выделения специфических графоэлементов элементами вектора $v(t)$ могут являться коэффициенты непрерывного вейвлет-преобразования. В простейшем частном случае $v(t) = x(t)$, $t = \overline{1, T}$.

2.2. Понятие статического и динамического паттерна временного ряда. В связи с тем, что элементы вектора МХП $v(t)$ связаны с динамическими особенностями сигнала $x(t)$ в момент времени t, предположим, что некоторые области в пространстве МХП V соответствуют отдельным функциональным состояниям динамической системы, продуцирующей временной ряд.

Пусть пространство V некоторым образом разбито на Q областей S_1, ..., S_Q таких, что

$$\begin{cases} \bigcup\limits_{q=1}^{Q} S_q = V; \\ S_{q_1} \cap S_{q_2} = \varnothing,\ q_1 = \overline{1, Q},\ q_2 = \overline{1, Q},\ q_1 \neq q_2. \end{cases} \quad (7)$$

Эти области далее будем называть *статическими паттернами* (СП). Будем говорить, что статический паттерн S_q *активен* в момент времени t, если $v(t) \in S_q$. Таким образом, каждый сигнал $\{x(t), t = \overline{1, T}\}$ вида (1) может быть представлен как последовательность номеров активных статических паттернов $\{q(t), t = \overline{1, T}\}$.

С течением времени активные статические паттерны для сигнала $\{x(t), t = \overline{1, T}\}$ некоторым образом сменяют друг друга. Долю времени, в течение которого СП S_q был активным, будем называть *индексом активности p* статического паттерна S_q:

$$p(S_q) = \frac{T(S_q)}{T},\ q = \overline{1, Q}, \quad (8)$$

где $T(S_q)$ — число тактов дискретного времени, в течение которого СП S_q был активным.

Классификацию временного ряда будем проводить по результатам анализа индексов активностей СП, присутствовавших в сигнале. Так, среди множества СП могут быть найдены паттерны, специфичные для временных рядов одного класса и неспецифичные для временных рядов другого класса. Эта специфичность паттерна может выражаться в существенном различии значений индексов активности для сигналов разных классов.

Однако, как показывают эксперименты и как показано в работах [5, 6], в ряде случаев классификация сигналов лишь на основе индексов активности СП не оказывается эффективной. Связано это с тем, что сигналы различных классов могут иметь одни и те же СП с примерно одинаковыми индексами активности. Различие же между классами сигналов проявляется в последовательности смены этих статических паттернов.

Обобщим понятие индекса активности (8) на цепочку статических паттернов $\Sigma_{q_1, \ldots, q_r} = (S_{q_1}, \ldots, S_{q_r})$ длины r, которую далее будем называть *динамическим паттерном* (ДП) r-го *порядка*:

$$p(\Sigma_{q_1, \ldots, q_r}) = p(S_{q_1}, \ldots, S_{q_r}) = \frac{T(S_{q_1}, \ldots, S_{q_r})}{T - r + 1},$$
$$q_i \in \{1, \ldots, Q\},\ i = \overline{1, r}, \quad (9)$$

где $T(S_{q_1}, \ldots, S_{q_r})$ — число вхождений цепочки $(q_1, \ldots, q_r)$ в последовательность номеров СП $\{q(t), t = \overline{1, T}\}$, соответствующую рассматриваемому сигналу $\{x(t), t = \overline{1, T}\}$. Индекс активности динамического паттерна фактически представляет собой относительную частоту встречаемости соответствующей последовательности переходов между статическими паттернами на рассматриваемом интервале времени.

2.3. Алгоритм формирования вектора характерных признаков сигнала. Введем понятие специфичности ДП Σ по отношению к паре классов (k_1, k_2), $k_1 \in \{1, \ldots, K\}$, $k_2 \in \{1, \ldots, K\}$, $k_1 \neq k_2$. Под специфичностью ДП Σ будем понимать степень успешности классификации значений индексов активности этого ДП на обучающих сигналах, относящихся к классам k_1 и k_2, с помощью некоторого классификатора. В качестве *показателя специфичности* ДП Σ по отношению к паре классов (k_1, k_2), таким образом, выберем точность работы классификатора:

$$I_{k_1, k_2}(\Sigma) = \frac{1}{2}\left(\frac{n_{k_1}}{N_{k_1}} + \frac{n_{k_2}}{N_{k_2}}\right),$$
$$k_1 \in \{1, \ldots, K\},\ k_2 \in \{1, \ldots, K\},\ k_1 \neq k_2, \quad (10)$$

где n_{k_1} и n_{k_2} — число правильно классифицированных примеров классов k_1 и k_2 соответственно по значению индекса активности ДП Σ, а N_{k_1} и N_{k_2} — общее число примеров соответствующих классов в обучающей выборке. Возможные значения показателя $I_{k_1, k_2}(\Sigma)$ лежат в отрезке [0; 1]. Значение $I_{k_1, k_2}(\Sigma) = 1$ означает разделимость классов k_1 и k_2 с помощью выбранного классификатора.

Наиболее простым классификатором индексов активности p динамического паттерна Σ на два класса k_1 и k_2 является линейный классификатор:

$$c[p(\Sigma)] = \begin{cases} k_1, \text{ если } p(\Sigma) \leqslant p_0, \\ k_2 \text{ в ином случае} \end{cases}$$
$$\text{или} \qquad (11)$$
$$c[p(\Sigma)] = \begin{cases} k_2, \text{ если } p(\Sigma) \leqslant p_0, \\ k_1 \text{ в ином случае,} \end{cases}$$

где значение разделяющего порога p_0 выбирается таким образом, чтобы обеспечить максимальное значение критерия (10).

Предполагая, что индекс ДП с максимальной специфичностью в наибольшей степени характеризует тип классифицируемого сигнала, будем использовать его в качестве характерного признака $z_{k_1k_2}$ сигнала $\{x(t), t = \overline{1, T}\}$ для его классификации на два класса k_1 и k_2:

$$z_{k_1k_2} = p(\Sigma^*_{k_1k_2}), \qquad (12)$$

где $\Sigma^*_{k_1k_2}$ — наиболее специфичный ДП для классов k_1 и k_2 с точки зрения критерия (10).

Задача линейной классификации сигналов на $K > 2$ классов может быть сведена к построению ряда линейных классификаторов на два класса вида (11), каждый из которых осуществляет классификацию в пространстве соответствующих признаков $z_{k_1k_2}$, $k_1 = \overline{1, K-1}$, $k_2 = \overline{k_1+1, K}$. Составим из этих признаков вектор z:

$$z = (z_{12}, \ldots, z_{K-1,K}). \qquad (13)$$

Именно этот вектор предполагается использовать в качестве вектора характерных признаков сигнала $\{x(t), t = \overline{1, T}\}$ для его классификации. Размерность вектора z не превышает $K^*(K-1)/2$.

2.4. Алгоритм оптимизации специфичности динамических паттернов. Значения показателей I_{k_1, k_2} для наиболее специфичных ДП $\Sigma^*_{k_1k_2}$, $k_1 = \overline{1, K}$, $k_2 = \overline{1, K}$, $k_1 \neq k_2$, определяются статическими паттернами $S_{q_1}, \ldots, S_{q_r}$. Поставим задачу разбиения пространства V на статические паттерны $S_1, \ldots, S_Q$ таким образом, чтобы максимизировать точность работы выбранного классификатора в пространстве признаков z. В связи с тем, что точность классификации на K классов связана с точностью попарной классификации каждой пары классов, введем следующий показатель точности:

$$I = \left(\prod_{k_1=1}^{K-1} \prod_{k_2=k_1+1}^{K} I_{k_1, k_2}(\Sigma^*_{k_1k_2}) \right)^{2/K(K-1)}. \qquad (14)$$

Значение показателя I, равное 1, будет означать разделимость всех пар классов в пространстве характерных признаков z с помощью выбранного классификатора.

Задача поиска оптимальных статических паттернов состоит в разбиении пространства признаков МПХ V на Q областей $S_1, \ldots, S_Q$ таким образом, чтобы максимизировать значение точности классификации (14) при ограничениях (7). Эту задачу можно свести к задаче разбиения множества векторов МХП сигналов обучающей выборки, каждый из которых представляет собой точку в пространстве V, на Q групп. В связи с тем, что число точек в пространстве V может достигать десятков или даже сотен тысяч, перед решением сократим ее размерность. Для этого проведем предварительную кластеризацию данных в пространстве V на N кластеров ($N \gg Q$), используя стандартные методы кластеризации, например, метод k-средних или самообучающиеся карты Кохонена [7, 8]. Далее найденные кластеры будем называть *микросостояниями*.

Таким образом, рассматриваемая оптимизационная задача сводится к разбиению N кластеров на Q групп (к задаче формирования Q статических паттернов из N микросостояний) таким образом, чтобы максимизировать критерий (14). Для решения этой комбинаторной задачи могут быть применены различные эвристические методы. В данной статье для ее решения предложено использовать генетические алгоритмы (ГА) [9].

2.5. Классификация в пространстве характерных признаков. После выбора пространства характерных признаков z следующим шагом является построение собственно классификатора данных $c[z]$ в этом пространстве. В связи с тем, что при формировании вектора z уже был выбран и использован классификатор для разделения пар классов по значениям индексов ДП, на данном этапе для разделения K классов тип классификатора целесообразно сохранить. Так, при использовании линейного классификатора (11) в расчетах показателя (10) классификатор $c[z]$ также целесообразно выбирать линейным.

Для оценки качества классификации будем использовать показатель

$$J = \frac{1}{K} \sum_{k=1}^{K} \frac{n_k}{N_k}, \qquad (15)$$

где n_k — число верно распознанных примеров K-го класса при предъявлении N_k сигналов этого класса. Значение показателя (15), равное 1, означает правильную классификацию всех предъявленных на вход классификатора примеров.

3. Результаты экспериментальных исследований

3.1. Описание исходных данных. Экспериментальные исследования предложенных алгоритмов классификации многомерных временных рядов проводились на сигналах электроэнцефалограмм (ЭЭГ), использовавшихся в III Соревновании по решению задач распознавания нейрофизиологиче-

ских данных (*BCI Competition III*) и предоставленных Технологическим университетом *Graz* (Австрия). Каждая ЭЭГ регистрировалась при воображении испытуемым одного из $K = 4$ типов движения: левой рукой (тип L), правой рукой (тип R), ногой (тип F) и языком (тип T).

В результате экспериментов собрано по 45 сигналов ЭЭГ каждого типа с частотой дискретизации $\omega_д = 80$ Гц. Запись ЭЭГ велась по 60 каналам, из которых были отобраны $L = 4$ в позициях C_3, C_4, C_z, PO_z. Все сигналы были разбиты на обучающую (60 %), валидационную (20 %) и тестовую (20 %) выборки, объемы которых соответственно равны $P_{обуч} = 0{,}6 \cdot 180 = 108$, $P_{валид} = 0{,}2 \cdot 180 = 36$, $P_{тест} = 0{,}2 \cdot 180 = 36$.

3.2. Расчет векторов моментных характерных признаков. В связи с тем, что электрическая активность мозга имеет гармоническую природу [10], для формирования МХП ЭЭГ предложено использовать непрерывное вейвлет-преобразование (НВП) с материнской вейвлет-функцией Морле, имеющей аналитический вид [11]

$$\psi(t) = \exp(-t^2/2)\exp(i2\pi t).$$

Расчет вейвлет-коэффициентов сигнала $\{x_l(t), t = \overline{1, T}\}$ в l-м канале ЭЭГ проводили в узлах дискретной сетки на плоскости (a, b) по формуле

$$w_{ij}^{l} = \frac{1}{\sqrt{a_i}} \sum_{t=0}^{T-1} x_l(t) \int_{t\Delta T}^{(t+1)\Delta T} \psi^*\left(\frac{\tau - b_j}{a_i}\right) d\tau,\ l = \overline{1, L},$$

где ΔT — интервал дискретности; a_i, b_j — параметры масштаба и временного сдвига соответственно; $\psi(t)$ — материнская вейвлет-функция, оператор * означает комплексное сопряжение.

Вектор МХП ЭЭГ $\nu(t)$ составлен из средних значений энергии сигнала каждого из каналов ЭЭГ в заранее заданных частотных диапазонах $\Omega_1, ..., \Omega_\mu$ в момент времени t:

$$\nu(t) = (\nu_1(t), ..., \nu_L(t)),\ \nu_l(t) = (\nu_{l1}(t), ..., \nu_{l\mu}(t)),$$
$$l = \overline{1, L},$$
$$\nu_{lm}(t) = \frac{1}{|\Omega_m|} \sum_{i \in \Omega_m} |w_{it}^{l}|^2,\ l = \overline{1, L},\ m = \overline{1, \mu}.$$

Исследовали пять частотных диапазонов: 8—10, 10—12, 13—15, 16—20 и 20—24 Гц, что соответствует диапазонам альфа- и бета-ритма. Размерность вектора МХП $\nu(t)$ равна μL, где μ — число выбранных частотных полос ($\mu = 5$), L — число каналов ($L = 4$).

Векторы МХП, рассчитанные для сигналов обучающей выборки, были кластеризованы на $N = 200$ кластеров (микросостояний) с помощью метода k-средних [7].

3.3. Формирование статических и динамических паттернов. Для решения задачи оптимизации критерия (14) с ограничениями (7) использован генетический алгоритм [9]. В связи с тем, что каждое микросостояние должно входить ровно в один статический паттерн, сформируем хромосому следующим образом:

$$g = (g_1, ..., g_N),\ g_i \in \{1, ..., Q\},\ i = \overline{1, N},$$

где g_i — i-й ген хромосомы g, представляющий собой целое число в диапазоне от 1 до Q. Значение $g_i = q$ означает, что микросостояние s_i принадлежит статическому паттерну S_q. Каждая хромосома, таким образом, однозначно определяет разбиение множества микросостояний $\{s_1, ..., s_N\}$ на статические паттерны $S_1, ..., S_Q$. Генетические операции над такими хромосомами определяются аналогично генетическим операциям для бинарных генетических алгоритмов.

После срабатывания критерия останова (по числу эпох) для хромосомы, показывающей лучший результат на обучающей выборке ($I_{обуч}$), рассчитывались значения показателя (14) на валидационной ($I_{валид}$) и тестовой ($I_{тест}$) выборках. В результате исследований установлено, что приемлемые значения $I_{обуч}$, $I_{валид}$, $I_{тест}$ наблюдаются при $Q = 10$. На рис. 1 приведены графики зависимости этих показателей от номера запуска генетического алгоритма при $Q = 10$.

По результатам многократного запуска ГА проводился выбор оптимальной хромосомы по критерию максимума значения показателя $I_{валид}$ на валидационной выборке. Для такой хромосомы значения $I_{обуч} = 0{,}94$, $I_{валид} = 0{,}89$, $I_{тест} = 0{,}81$.

Были использованы следующие параметры генетического алгоритма. Длина хромосомы $N = 200$, численность популяции — 100, алгоритм формирования родительского пула — по методу рулетки, кроссинговер — равномерный с вероятностью обмена генами 0,5, элитная стратегия селекции, вероятность мутации — 0,05, число поколений — 500.

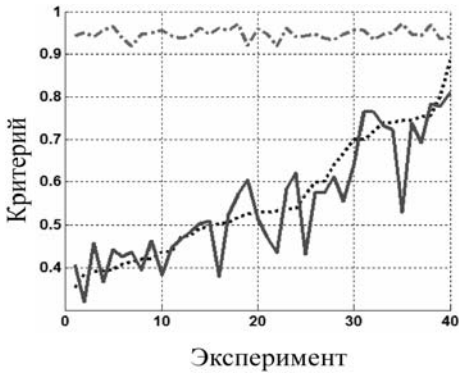


Рис. 1. Графики зависимости показателя специфичности (14) на обучающей (штрих-пунктирная линия), валидационной (пунктирная линия) и тестовой (сплошная линия) выборках от номера запуска генетического алгоритма. Номера запусков отсортированы по значению $I_{валид}$

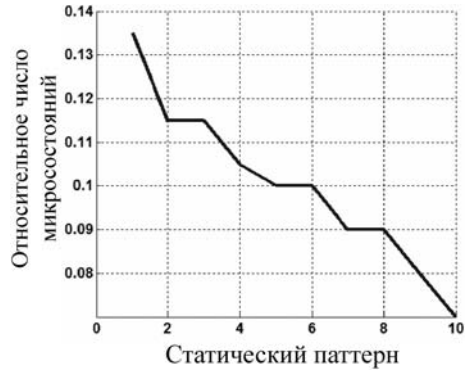


Рис. 2. Распределение числа микросостояний по статическим паттернам для оптимальной хромосомы. По оси абсцисс отложен номер статического паттерна, по оси ординат — отсортированное относительное число микросостояний. Общее число микросостояний $N = 200$, число статических паттернов $Q = 10$

Таблица 1

Наиболее специфичные ДП 2-го порядка и значения показателей специфичности

Классы	ДП	$I_{\text{обуч}}$	$I_{\text{тест}}$
'*LR*'	(4,3)	0,9259	0,7778
'*LF*'	(6,6)	0,9814	0,9444
'*LT*'	(6,6)	0,9259	0,9444
'*RF*'	(6,6)	0,9814	0,9444
'*RT*'	(6,6)	0,9259	0,8889
'*FT*'	(10,7)	0,9074	0,7778

Таблица 2

Доля числа правильно распознанных примеров каждого класса обученным нейросетевым классификатором на обучающей и тестовой выборках

Классы	$I_{\text{обуч}}$	$I_{\text{тест}}$
'*L*'	0,5926	0,5556
'*R*'	0,8889	0,4444
'*F*'	0,8148	0,5556
'*T*'	0,8519	0,7778

На рис. 2 приведено распределение числа микросостояний по статическим паттернам, рассчитанное для найденного разбиения.

В табл. 1 приведены наиболее специфичные ДП $\Sigma^*_{k_1 k_2}$ для каждой пары классов (k_1, k_2) с точки зрения показателя (10), рассчитанные по данным обучающей и тестовой выборок для выбранной оптимальной хромосомы, и указаны значения показателя специфичности для них. При расчете показателя (10) использован линейный классификатор (11).

Из табл. 1 видно, что наиболее хорошо разделимы классы '*L*', '*F*' и '*R*', '*F*', что согласуется с нейрофизиологическими представлениями — при воображении движения руками и ногой активизируются принципиально различные участки коры головного мозга.

3.4. Формирование вектора характерных признаков и классификация. Вектор характерных признаков z в соответствии с формулой (13) формируется из значений индексов активности наиболее специфичных ДП, приведенных в табл. 1. Размерность вектора z равна 3. По построению каждый элемент вектора z является оптимальным с точки зрения попарного линейного разделения сигналов на соответствующую пару классов. На рис. 3 (см. третью сторону обложки) приведена диаграмма рассеяния элементов вектора z на плоскости индексов ДП (6,6) и (4,3).

Из рис. 3 видно, что в указанной плоскости линейно разделимы с высокой точностью все пары классов, кроме пары '*F*', '*T*' — эти классы разделимы по индексам ДП (10,7), не представленным в приведенной проекции.

Для классификации векторов характерных признаков, рассчитанных для всех сигналов обучающей выборки, был использован линейный дискриминантный анализ [12]. Получены значения показателя точности (15) на обучающей и тестовой выборках соответственно $J_{\text{обуч}} = 0{,}79$, $J_{\text{тест}} = 0{,}58$. Доля числа правильно распознанных примеров каждого класса (значения n_k/N_k в формуле (15)) линейным классификатором на обучающей и тестовой выборках приведена в табл. 2.

На рис. 4 (см. третью сторону обложки) показано распределение значений выходов построенного линейного классификатора при подаче на вход сигналов обучающей и тестовой выборок различных классов.

Заключение

Результаты применения предложенного алгоритма для выделения характерных признаков многомерных нестационарных сигналов ЭЭГ и их классификации показали его работоспособность на имеющихся экспериментальных данных. Точность линейного классификатора в построенном пространстве признаков составила 79 % на обучающей выборке и 58 % на тестовой.

Несмотря на достигнутый результат, предложенный алгоритм требует проведения ряда исследований, которые можно объединить в следующие группы:

- исследование различных пространств моментных характерных признаков;
- исследование процедуры кластеризации векторов МХП и формирования множества микросостояний;
- исследование процедуры формирования статических паттернов из микросостояний;
- исследование возможности и результатов применения динамических паттернов высоких порядков;
- исследование различных вариантов формирования вектора характерных признаков на основе индексов активности динамических паттернов;
- исследование результатов классификации векторов характерных признаков с помощью различных классификаторов.

Результатом этих исследований должны стать рекомендации по выбору числа микросостояний и числа статических паттернов, определение порядка вводимых в рассмотрение динамических паттернов, достаточного для обеспечения требуемой точности классификации, а также ряд других выводов по применимости и эффективности предложенного алгоритма формирования характерных признаков для классификации многомерных временных рядов.

Работа выполнена при финансовой поддержке Совета по грантам Президента РФ для поддержки молодых российских ученых (грант МК-4245.2009.8), а также в рамках гранта РФФИ 10-08-00816 и ФЦП "Научные и научно-педагогические кадры инновационной России" на 2009—2013 годы.

Список литературы

1. **Priestley M. B.** Nonlinear and Nonstationary Time Series Analysis. Academic Press, 1988.
2. **Nonlinear** and Nonstationary Signal Processing / Edited by W. J. Fitzgerald. University of Cambridge. 2001.
3. **Dahlhaus R.** Fitting time series models to nonstationary processes // The Annals of Statistics. 1997. Vol. 25. N 1. P. 1—37.
4. **Wolpaw J. R., Birbaumer N., McFarland D. J., Pfurtscheller G., Vaughan T. M.** Brain-computer interfaces for communication and control // Clinical Neurophysiology. 2002. Vol. 113. P. 767—791.
5. **Скругин В. И., Роик А. О., Наумов Р. А., Трофимов А. Г.** Алгоритм классификации сигналов ЭЭГ на основе анализа в частотно-временной области // Сб. науч. трудов XII Всероссийской научно-техн. конф. "Нейроинформатика-2010". М.: НИЯУ МИФИ. 2010. Т. 1. С. 266—276.
6. **Скругин В. И., Роик А. О., Наумов Р. А., Трофимов А. Г.** Алгоритм выделения пространственно-частотных паттернов в структуре сигнала ЭЭГ // Матер. избранных науч. трудов по теме: "Актуальные вопросы нейробиологии, нейроинформатики и когнитивных исследований". М.: НИЯУ МИФИ. 2010. С. 213—223.
7. **Айвазян С. А., Мхитарян В. С.** Прикладная статистика и основы эконометрики: учебник для вузов. М.: ЮНИТИ, 1998.
8. **Осовский С.** Нейронные сети для обработки информации. М.: Финансы и статистика, 2004.
9. **Емельянов В. В., Курейчик В. В., Курейчик В. М.** Теория и практика эволюционного моделирования. М: Физматлит, 2003.
10. **Зенков Л. Р.** Клиническая электроэнцефалография (с элементами эпилептологии). Руководство для врачей. М.: МЕДпресс-информ, 2004.
11. **Витязев В. В.** Вейвлет-анализ временных рядов: учеб. пособие. СПб.: Изд-во СПб ун-та, 2001.
12. **McLachlan G.** Discriminant Analysis and Statistical Pattern Recognition. New York: John Wiley & Sons, 1992.

УДК 621.391

А. К. Цыцулин,
д-р техн. наук, проф., зам. директора,
ФГУП "НИИТ", г. Санкт-Петербург,
e-mail:atsytsulin@mail.ru,
Ш. С. Фахми, канд. техн. наук, доц.,
СПбГЭГУ "ЛЭТИ",
e-mail:Shakeebf@mail.ru,
Е. И. Колесников, аспирант,
С. В. Очкур, аспирант

Функционал взаимообмена сложности и точности систем кодирования непрерывного сигнала

Предложены критерий эффективности кодирования источника непрерывных сообщений с учетом сложности кодера и функционал, связывающий точность передачи непрерывного сигнала со скоростью передачи и сложностью кодера, стремящийся к их среднему гармоническому. На конкретных примерах показана корректность предложенного метода оценки эффективности приближения к эпсилон-энтропии различных методов кодирования изображений.

Ключевые слова: *скорость кода, сложность кодирования, критерий качества кодирования*

После того, как К. Шеннон [1] определил *идеальное* кодирование как обеспечивающее равенство пропускной способности C и энтропии H (эпсилон-энтропии H_ε) источника, сформировались два научно-технических направления: кодирование источника и кодирование канала [2, 3]. В силу нацеленности на достижение этого идеала кодирование источника названо "эффективным кодированием" [3, 4]. Поэтому главной характеристикой кодера источника традиционно считается его "эффективность", характеризуемая, во-первых, *коэффициентом сжатия* $K_{сж}$ скорости кода, определяемым как отношение $R_{вх}/R_{вых}$ скорости кода на входе кодера — *тривиальной* статистики — к скорости кода на выходе — *минимальной достаточной* статистики [4]. Во-вторых, "эффективность" характеризуется степенью приближения скорости кода на его выходе $R_{вых}$ к энтропии H (эпсилон-энтропии H_ε) источника [5]. Учет обоих этих требований — сжатия и приближения к эпсилон-энтропии — приводят к выражению для эффективности *кодирования*:

$$E = K_{сж}\frac{H_\varepsilon}{R_{вых}} = \frac{R_{вх}H}{R_{вых}^2}. \qquad (1)$$

Синтез кодера источника традиционно использует опорную триаду: априорную информацию, критерий качества и ограничения. На практике

синтез системы передачи информации включает различные виды ограничений, в первую очередь мощности в канале (рассмотренной К. Шенноном в качестве *примера* ограничения) задержки [6], длины блоков [2, 3], широкополосности кодера [7—10]. Вместе с тем, разработчики методов "эффективного" кодирования упоминают о возможных ограничениях при синтезе кодера, в первую очередь о сложности кодера, не включая эти ограничения в формулировку задачи синтеза системы связи [5, 11]. Характерно, что при этом задача уменьшения вычислительной сложности [11] рассматривается отдельно от задачи определения требуемой разрядности вычислений в кодере [12]. Формализованный учет сложности кодеров непрерывных источников, стимулированный созданием и развитием СБИС [13], систем на кристалле [14] и видеосистем на кристалле [9], требует пересмотра понятия "эффективное" кодирование.

Выше понятие "эффективности" кодирования ставилось в кавычки потому, что идеальная по Шеннону система, обеспечивающая предельную "эффективность" в смысле равенства пропускной способности и энтропии и в смысле соотношения (1), должна включать элементы, характеризуемые *бесконечной сложностью*. В частности, требование бесконечной сложности кодера источника непрерывного сигнала для достижения *конечной скорости* передачи информации через канал очевидно, хотя бы потому, что коэффициенты базисных функций и коэффициенты передачи *оптимального* конечномерного кодирующего фильтра имеют континуальные значения, требующие бесконечной емкости памяти для их хранения. Бесконечные затраты информационного ресурса (сложности) при конечном полезном эффекте (скорости передачи) означают нулевую эффективность использования этого ресурса. Отсюда возникает противоречие, которое может быть устранено лишь дополнением понятия "эффективность кодирования" другим понятием — *эффективности кодера* (кодека), учитывающим сложность достижения "эффективности кодирования".

В круг учитываемых величин, связанных с кодированием источника (показателей качества системы), должны войти не только точность передачи (ошибка) и скорость передачи, но и сложность. Известно, что эти величины взаимосвязаны: *"Оптимизацию видеокодека надо делать (минимум) по трем параметрам: по битовой скорости, по искажению и по вычислительной сложности. Все они влияют друг на друга. Например, оптимизация соотношения скорость/искажение достигается за счет повышения сложности кодирования, "быстрые" алгоритмы оценки движения часто имеют низкую вычислительную сложность за счет снижения эффективности кодирования и т. д. Эффективность кодирования и сложность кодирования являются настоящими антиподами"* [15].

Обобщенный показатель эффективности кодирования должен учитывать не только меру приближения к эпсилон-энтропии источника с помощью показателя (1), но и то, какими информационными средствами достигнуто данное приближение, в первую очередь — какой сложностью кодера.

В соответствии с методологией оптимизации радиоэлектронных устройств по совокупности показателей качества [16, 17] целесообразно сформировать функционал, включающий взвешенную сумму частных показателей качества. Вектор весовых коэффициентов при показателях качества в системном анализе принято называть "вектором концепции системы" [18]. Очевидно, что в различных системах эти векторы могут существенно различаться, что и приводит к существованию множества различных оптимальных систем связи [16]. Например, даже при равенстве весовых коэффициентов при скорости передачи и ошибке передачи системы кодирования непрерывных источников в телевизионном вещании и в аэрокосмических системах наблюдения существенно различаются в соотношении весовых коэффициентов при сложности кодера и декодера. В телевизионном вещании в силу огромных тиражей приемников существенный вес имеет сложность декодера, в аэрокосмических системах связи в силу жестких массогабаритных ограничений передатчиков — сложность бортового кодера.

Для обеспечения общего информационного подхода к решению задачи оптимизации системы связи необходимо все частные показатели качества привести к единому виду, имеющему размерность информации (бит и т. п.). В частности, учет деградации отношения сигнал/помеха следует характеризовать потерей полезной информации ΔI, вычисляемой как логарифм отношения фактической ошибки к минимально возможной (в частности, имеющейся во входном сигнале непрерывного источника, подлежащем кодированию). Следуя методике векторного синтеза системы связи [16], целесообразно сформировать обобщенный показатель эффективности P, включающий "вектор концепции системы" $\{c_i\}$ и совокупность $\{P_i\}$ частных информационных показателей качества системы связи — потери полезной информации ΔI, скорости передачи R, сложности $W_к$ кодера и сложности $W_д$ декодера. Так как все частные информационные показатели качества системы связи связаны с ошибкой передачи ε, то ниже они будут обозначаться как $P_i(\varepsilon)$:

$$P = c_0\Delta I(\varepsilon) + c_1 R(\varepsilon) + c_2 W_к(\varepsilon) + c_3 W_д(\varepsilon) \to \min. \quad (2)$$

Для выявления основных свойств обобщенного показателя качества рассмотрим его частный случай при $c_3 = 0$, характерный для бортовых систем прикладного телевидения, акцентирующих внимание на сложности кодера, опуская индекс при обозначении сложности кодера:

$$P = c_0 \Delta I(\varepsilon) + c_1 R(\varepsilon) + c_2 W_к(\varepsilon) \to \min. \qquad (3)$$

Обобщенный показатель эффективности кодирования (2) и (3) позволяет объективно сравнивать произвольные кодеры, различающиеся по всем параметрам — ошибке передачи, скорости передачи и сложности, если зафиксировать выборку испытательных сигналов.

Искомый минимум P при вариации назначаемой ошибки передачи ε может быть найден аналитически, если известны статистические свойства источника и зависимости сложности и скорости передачи от назначаемой ошибки. Известно [16], что обобщенный показатель качества допускает множество решений. Это разнообразие решений в первую очередь связано с отмеченным взаимообменом скорости передачи и сложности кодера [15].

Взаимосвязь скорости передачи и сложности кодирования при идеальной передаче сигналов дискретного источника по дискретному каналу (вероятность ошибки $p_\varepsilon \to 0$ при скорости передачи R, меньшей пропускной способности C) известна по работам К. Шеннона [1] и Р. Фано [2]. Эта взаимосвязь относится к фундаментальным положениям теории кодирования [3]: для канала с аддитивным белым гауссовским шумом и с неограниченной полосой частот вероятность ошибки пропорциональна экспоненте от произведения скорости передачи R на длину блока T:

$$P_\varepsilon \approx 2^{-RT}. \qquad (4)$$

Это означает, что "при фиксированной скорости передачи за уменьшение вероятности ошибки можно расплачиваться либо увеличением пропускной способности канала, либо увеличением сложности устройств и задержки декодирования" [3]. Можно отождествить длину блока со сложностью кодера [2] и читать этот известный принцип теории связи не только как возможность достижения нулевой вероятности ошибки передачи дискретного сигнала через дискретный канал связи при неограниченной длине блоков, но и как *взаимообмен скорости передачи и сложности кодера* при достижении заданного уровня ошибки. Этот принцип теории связи распространяется и на смешанную систему связи, включающую непрерывный источник и дискретный канал. Практические системы кодирования непрерывных сигналов (к которым относятся изображения), такие как выполняемые по стандартам MPEG, включают в себя и кодер источника, и кодер канала. Здесь будем акцентировать внимание лишь на кодере источника, полагая сложность кодера канала фиксированной и выступающей при оценке сложности кодера в целом в роли аддитивной константы. Для получения оценок влияния сложности кодера на точность передачи в такой модели кодирования непрерывных сигналов целесообразно рассмотреть частный случай кодирования сигнала с финитной функцией плотности вероятностей, для которых при критерии максимума ошибки [19] существует идеальное кодирование (равенство скорости создания информации источником и скорости передачи в канале). Для них справедлива формула эпсилон-энтропии [20], связывающая минимальное количество информации с суммой логарифмов спектрального отношения сигнал/ошибка. При этом ошибка в оптимальной системе равномерно распределена по передаваемым спектральным компонентам и является энергетическим порогом θ, по которому усекается спектр сигнала [1, 19]:

$$H_\varepsilon = \sum_{k=1}^{n} \log \frac{|A_k|}{\theta} \text{ при } \varepsilon = \theta n \sum_{k=1}^{n} |A_k|, \qquad (5)$$

где θ — шум сигнала.

Так же как и при кодировании дискретного источника, эпсилон-энтропия (5) является нижней гранью скорости передачи R для достижения заданной ошибки ε:

$$H_\varepsilon \leqslant R \to C. \qquad (6)$$

Вместе с тем, так же как и при кодировании дискретного источника, на точность передачи влияет и сложность W кодера. Конечно, изучение сложности системы имеет целью ее уменьшение при выполнении условия заданного качества системы [21, 22]. Ниже рассматривается методика проектирования кодеров источника непрерывного сигнала, в которой сложность кодера является не столько ограничением при минимизации ошибки передачи, сколько частным информационным показателем качества системы, входящим в обобщенный показатель (2).

Первое приближение к оценке сложности кодера источника может быть сделано на основе вычисления энтропии источника через логарифм энтропийной мощности [3] и, следовательно, достижения нижней границы скорости передачи дискретного по времени стационарного процесса с помощью взятия первой конечной разности. Эта операция при кодировании реальных изображений, т. е. нестационарных случайных полей, используется в различных вариантах дифференциальной импульсно-кодовой модуляции [1—4], в частности, при реализации пирамидально-рекурсивных методов кодирования [9], но уже не исчерпывает требуемой сложности кодера источника.

Известно [21, 22], что требуемая сложность W вычислений логарифмически связана с задаваемой ошибкой:

$$W \approx \log \frac{1}{\varepsilon}. \qquad (7)$$

При использовании двоичных логарифмов сложность, так же как и скорость передачи, измеряется в битах и имеет смысл взвешенной суммы числа вентилей и числа операции (шагов алгоритма) кодирования источника. При приложении теории сложности вычислений к кодированию источников непрерывных сигналов обычно стремятся к снижению степени зависимости числа операций кодера от числа отсчетов сигнала. Известно, что сложность спектрального преобразования, часто составляющего ядро кодера источника, может иметь оценку $O(n \log n)$ и за счет распараллеливания вычислений может быть уменьшена до $O(n)$ [11, 13]. Эту зависимость сложности от числа отсчетов $O(n)$ можно обобщить с учетом точности представления операндов (входных сигналов и значений базисных функций). С точностью до аддитивной константы, учитывающей операции синхронизации, эпсилон-сложность (т. е. минимальная сложность [22]) кодера источника, так же как и эпсилон-энтропия, может быть представлена суммой логарифмов спектрального отношения сигнал/ошибка:

$$W_\varepsilon = \sum_{k=1}^{n} \log \frac{|A_k|}{\theta}. \qquad (8)$$

На практике сложность кодера, так же как и скорость передачи, рассматривают как нормированную величину, относя их к информационным затратам на один пиксел. Ввести учет сложности в проектирование системы связи можно несколькими способами, в частности, сложность можно ввести как ограничение при пропускной способности в уравнении связи [8]. Ниже, аналогично введению понятия эпсилон-энтропии с задержкой [6], введем понятие *эпсилон-энтропии с ограничением сложности* $H_{\varepsilon w}$, определяемой как минимальное число операций, необходимое для кодирования изображений со скоростью R при заданной точности. Так как дополнительное ограничение неизбежно ведет к увеличению скорости передачи, то общие условия передачи (6) трансформируются к виду

$$H_\varepsilon \leqslant H_{\varepsilon w} \leqslant R \to C. \qquad (9)$$

Опираясь на экспериментальные исследования по взаимообмену скорости передачи и сложности кодирования [15, 23], можно считать, что, так же как и в формуле (4) для дискретного канала, требуемая предельная скорость передачи информации R и сложность W (т. е. длина кода T) оказывают одинаковое влияние на ошибку передачи ε. Вместе с тем, в отличие от формулы (4) для дискретного источника ни скорость передачи, ни сложность кодера ($T = W$) не могут быть сделаны сколь угодно малыми за счет другой из этих величин, а имеют нижние границы (5) и (8). Такое положение при любом способе вычисления эпсилон-энтропии — и при среднеквадратической мере ошибки, и при ограничении максимума ошибки — может быть записано аналитически в виде, объединяющем эпсилон-энтропию с ограничением сложности $H_{\varepsilon w}$ и эпсилон-сложности W_ε кодирования в виде функционала, восходящего к колмогоровским средним, а именно гармоническому среднему, отличающегося от гармонического среднего отсутствием множителя числа слагаемых:

$$H_\varepsilon = \frac{H_{\varepsilon w} W}{H_{\varepsilon w} + W}. \qquad (10)$$

Эпсилон-энтропия H_ε (5) является пределами эпсилон-энтропии с ограничением сложности $H_{\varepsilon w}$ и сложности W. Этот предел достигается лишь при условии использования разложения Карунена—Лоэва и стремлении к бесконечности доступной сложности $W_к$ описания базисных функций и операций вычисления соответствующих скалярных произведений. При этом формула (10) переходит в формулы (5) и (8):

$$\lim H_{\varepsilon w} \to \lim W = H_\varepsilon,$$
$$W \to \infty,\ H_{\varepsilon w} \to \infty.$$

Так как эпсилон-энтропия по определению является нижней гранью количества информации, необходимого для достижения заданной ошибки, то ошибка, вычисляемая из формулы для эпсилон-энтропии, будет минимальной ошибкой, выступающей нижней гранью значения ошибки при соответствующей скорости кода:

$$\varepsilon \geqslant \varepsilon_{\min} = 2^{H\varepsilon}. \qquad (11)$$

С учетом формулы взаимообмена скорости передачи и сложности кодера нижняя грань достижимой ошибки составит (рис. 1):

$$\varepsilon \geqslant \varepsilon_{\min} = 2^{-\frac{H_{\varepsilon w} W}{H_{\varepsilon w} + W}}. \qquad (12)$$

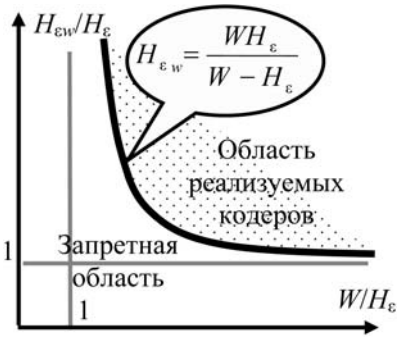


Рис. 1

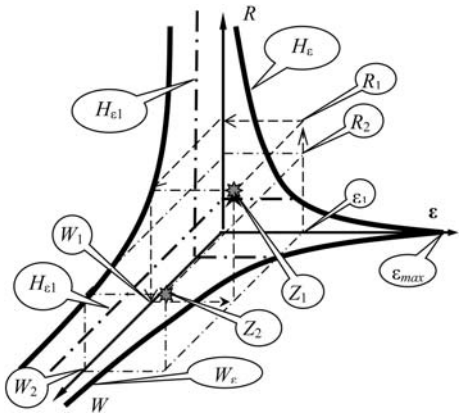


Рис. 2

Начальный этап проектирования кодера источника на основе выражений (9) и (10) иллюстрируется рис. 2, показывающим взаимосвязь трех важнейших величин: ошибки передачи, скорости передачи и сложности (ε—R—W). Из него видно, что при одной и той же ошибке передачи ε_1 переход от способа кодирования Z_1 к способу кодирования Z_2 ценой перехода от сложности W_1 к большей сложности W_2 обеспечивает лучшее приближение к эпсилон-энтропии: $R_2 < R_1$. Уменьшение назначаемой ошибки в соответствии с формулами (5), (11), (12) ведет к увеличению эпсилон-энтропии и сдвигу границы реализуемых кодов на рис. 1 вправо и вверх, на рис. 2 влево и вверх.

Важной иллюстрацией взаимообмена сложности кодера источника и приближения к эпсилон-энтропии является метод создания библиотек базисов для кодирования сигналов с большой априорной неопределенностью. Для достижения максимального сжатия кодер существенно усложняется введением множества устройств хранения информации о различных совокупностях базисных функций и устройства выбора наиболее эффективного из них [25]. Эта же тенденция существенного увеличения сложности для относительно небольшого увеличения сжатия видеоинформации наблюдается и при переходе от стандарта MPEG-2 к стандарту MPEG-4. Вместе с тем, сложность различных кодеров при сопоставимых коэффициентах сжатия может существенно различаться. Например, при кодировании динамических сюжетов использование трехмерного дискретного косинусного преобразования (3D ДКП) позволяет на несколько порядков сократить сложность кодера источника по сравнению с кодером MPEG-4 [26]. Это обусловлено, во-первых, тем, что ДКП является асимптотически оптимальным (при больших размерах фрагментов) разложением стационарного сигнала с экспоненциальной автокорреляционной функцией, хорошо аппроксимирующей реальные изображения. Во-вторых, эти два вида кодеров оптимальны для различных характеристик изменения сюжета во времени: MPEG-4 ориентирован на слежение за объектами, а 3D ДКП выделяет любые изменения в сюжете.

Корректность предложенного функционала иллюстрируется сравнительной оценкой зависимости скорости кодирования R при заданной средней квадратической ошибке ε и использовании различных методов кодирования источника (рис. 3, см. третью сторону обложки). Эти эксперименты на представительной выборке сюжетов показали [27], что дискретное пространственное преобразование (ДПП, без перехода в спектральную область) позволяет снизить скорость передачи по сравнению с вейвлет-преобразованием (ДВП, с переходом в спектральную область и использованием локальных интервалов задания базисных функций) в 1,2 раза и по сравнению с дискретным косинусным преобразованием (ДКП, с переходом в спектральную область) в 1,3 раза [9, 29], а уменьшение назначаемой ошибки при любом методе ведет и к увеличению скорости передачи, и к увеличению сложности кодера (рис. 4, см. третью сторону обложки).

Переход от одного способа кодирования к другому соответствует взаимообмену скорости передачи и сложности кодера, предельные возможности которого описываются формулами (9) и (10). Преимущество ДПП обусловлено нестационарностью сигналов реальных изображений, так как кодирование с преобразованием в спектральную область предполагает стационарность сигнала в пределах фрагмента. Этот тезис полностью относится к ДКП и, в меньшей мере, к вейвлет-преобразованию. Поэтому можно утверждать, что размер фрагмента при кодировании изображений выбирается не только исходя из ограничения сложности, но и с учетом нестационарности реальных сигналов.

Наглядное представление взаимообмена скорости и сложности, соответствующего формуле (10), может быть дано номограммой, аналогичной известной в оптике (рис. 5).

Ее анализ показывает, что числовые значения скорости передачи и сложности кодирования в

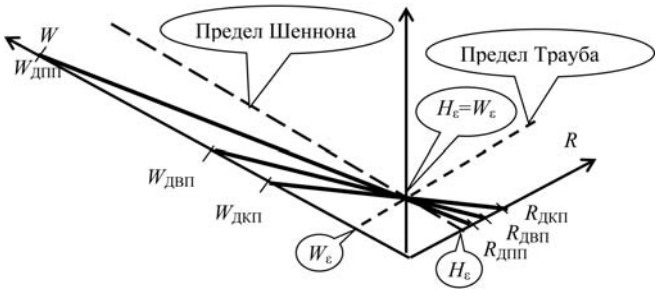


Рис. 5

эксперименте хорошо совпадают с предложенным функционалом (10) [29]: линии, соединяющие сложность и скорость передачи для каждого из методов кодирования, пересеклись в одной точке на вертикальной оси номограммы, которая соответствует эпсилон-энтропии кодера источника. Полезным свойством предложенного метода оценки эффективности кодера является возможность оценки эпсилон-энтропии кодируемого сигнала по номограмме рис. 5. Например, удалось оценить, что дискретно-пространственное преобразование не только ценой почти трехкратного усложнения кодера позволяет снизить скорость передачи на 30 % по сравнению с вейвлет-кодированием, но и обеспечивает в среднем скорость передачи, лишь на 25 % большую, чем эпсилон-энтропия кодируемого изображения.

Выбор того или иного способа кодирования непрерывного источника означает выбор конкретного соотношения коэффициентов c_1 и c_2 в векторе концепций в критерии (3), а выбор назначаемой ошибки означает выбор отношения коэффициента c_0 при потере полезной информации к коэффициентам c_1 и c_2. Взаимообмен скорости передачи и сложности при фиксированной ошибке передачи представлен в таблице, в которой каждому из рассмотренных способов кодирования непрерывного источника сопоставлено соответствующее значение соотношения коэффициентов c_1 и c_2 в векторе концепций в критерии (3).

Учет взаимообмена скорости передачи информации и сложности кодера, формализуемый формулой (10), позволяет найти минимум в критерии (3), который достигается при оптимальных значениях скорости $H_{\varepsilon w\text{o}}$ и сложности $W_{\varepsilon\text{o}}$. Подставляя формулу (10) в формулу (3), получаем:

$$P = c_0\Delta I(\varepsilon) + c_1 H_{\varepsilon w} + c_2 W_\varepsilon = \\ = c_0\Delta I + c_1 H_{\varepsilon w} + c_2 \frac{H_\varepsilon H_{\varepsilon w}}{H_{\varepsilon w} - H_\varepsilon}. \qquad (13)$$

Дифференцируя полученное выражение по эпсилон-энтропии с ограничением скорости $H_{\varepsilon w}$ и приравнивая производную нулю, получаем:

$$H_{\varepsilon w\text{o}} = H_\varepsilon\left(1 + \sqrt{\frac{c_2}{c_1}}\right);\ W_\text{o} = H_\varepsilon\left(1 + \sqrt{\frac{c_1}{c_2}}\right). \qquad (14)$$

В результате учета взаимообмена скорости передачи и сложности при реализации оптимального кодирования, ведущего к соотношениям (14) и обеспечивающего равенство в формуле (12), критерий оптимальности кодера (3) приводится к виду

$$P = c_0\Delta I + (\sqrt{c_1} + \sqrt{c_2})^2 H_\varepsilon. \qquad (15)$$

При известных зависимостях потери информации и эпсилон-энтропии от назначаемых ошибок можно найти оптимальную ошибку передачи, т. е. оптимальную деградацию отношения сигнал/шум в ходе кодирования и передачи. В частности, для одномерных сигналов с амплитудой A с учетом зависимости потери информации и кодовой эпсилон-энтропии от минимальной ошибки [20]

$$\Delta I = \frac{1}{2}\sum_{k=1}^{n}\log\frac{\varepsilon_k}{\varepsilon_{\min k}};\ H_\varepsilon = \frac{1}{2}\sum_{k=1}^{n}\log\frac{A_k}{\varepsilon_k - \varepsilon_{\min k}}$$

получаем $$\varepsilon_0 = \varepsilon_{\min}\frac{c_0}{c_0 - (\sqrt{c_1} + \sqrt{c_2})^2}. \qquad (16)$$

Формула для оптимального значения ошибки передачи показывает, в частности, что искомый экстремум качества кодера достигается не при любых векторах концепции системы, а только для таких, в которых весовой коэффициент потери полезной информации превышает среднее геометрическое весовых коэффициентов скорости передачи и сложности кодера.

При оптимизации кодирования многомерных сигналов должны также решаться задачи достижения максимума качества информации [10] и распределения сложности между различными узлами (функциями) кодера, например, распределение памяти между числом базисных функций и их разрядностью [24]. Учет этих дополнительных требований при кодировании многомерных сигналов ведет к тому, что вектор концепции в критерии (3) и его следствиях (14)—(16) не может назначаться произвольно, а должен учитывать потенциальное количество информации в передаваемом непрерывном зашумленном сигнале.

Изложенная методика концептуального проектирования кодеров непрерывных источников на основе введенного обобщенного показателя качества (3) и взаимообмена скорости передачи и сложности кодера (10) позволяет оптимизировать проектирование, особенно актуальное при реализации кодеров в виде СБИС и СФ-блоков [14, 29].

Метод	Относительная скорость передачи	Относительная сложность	c_1/c_2
ДКП	1,62	1	2,6
ДВП	1,5	1,5	4
ДПП	1,25	3,5	16
Триангуляционное преобразование	1,1	10	100
Преобразование Карунена—Лоэва	1	∞	∞

Выводы

- Предложен критерий (3) эффективности кодера непрерывного источника, включающий взвешенную (на основе вектора концепции системы) сумму потери полезной информации, скорости передачи и сложности кодера.
- Предложен функционал (10), связывающий точность передачи непрерывного сигнала со скоростью передачи и сложностью кодера, восходящий к их среднему гармоническому, и на конкретных примерах различных методов кодирования изображений показана корректность предложенной аппроксимации взаимообмена этих величин.
- Показано, что в системе, оптимизированной по критерию (3) с учетом взаимообмена скорости передачи и сложности (10), ошибка передачи зашумленных непрерывных сигналов определяется минимальной ошибкой (лимитируемой входным шумом) и соотношением (16) компонентов вектора концепции системы.

Список литературы

1. **Шеннон К.** Работы по теории информации в кибернетике. М.: ИЛ, 1963. 830 с.
2. **Фано Р.** Передача информации. Статистическая теория связи. М.: Мир, 1965. 439 с.
3. **Оливер Б.** Эффективное кодирование // Теория информации и ее приложения / Под ред. А. А. Харкевича. М.: Физико-математическая литература. 1959, 328 с.
4. **Цуккерман И. И., Кац Б. М., Лебедев Д. С.** и др. Цифровое кодирование телевизионных изображений / Под ред. И. И. Цуккермана. М.: Радио и связь, 1981. 240 с.
5. **Свириденко В. А.** Анализ систем со сжатием данных. М.: Связь, 1977. 184 с.
6. **Горбунов А. К., Пинскер М. С.** Эпсилон-энтропия с задержкой при малой среднеквадратической ошибке воспроизведения // Проблемы передачи информации. 1987. Т. 23. № 2. С. 3—8.
7. **Хромов Л. И., Цыцулин А. К., Куликов А. Н.** Видеоинформатика. М.: Радио и связь, 1991. 192 с.
8. **Хромов Л. И.** Теория информации и теория познания. СПб.: РФО, 2006. 200 с.
9. **Березин В. В., Умбиталиев А. А., Фахми Ш. С., Цыцулин А. К., Шипилов Н. Н.** Твердотельная революция в телевидении. М.: Радио и связь, 2006. 312 с.
10. **Цыцулин А. К.** Теория линейного кодирования зашумленных сигналов // Вопросы радиоэлектроники. Сер. Техника телевидения, 2009. Вып. 2. С. 16—40.
11. **Нуссбаумер Г.** Быстрое преобразование Фурье и алгоритмы вычисления сверток. М.: Радио и связь, 1985. 248 с.
12. **Борисов Б. И.** Первая отечественная система на кристалле с быстродействующими ЦАП/АЦП 600 Мвыборок/с по двум квадратурным каналам // Электроника: наука, технология, бизнес. 2004. № 2. С. 36—42.
13. **Сверхбольшие** интегральные схемы и современная обработка сигналов / Под ред. С. Гуна, Х. Уайтхауса, Т. Кайлата. М.: Радио и связь, 1989. 470 с.
14. **Немудров В. Г., Мартин Г.** Системы на кристалле. Проблемы проектирования и развития. М.: Техносфера, 2004. 216 с.
15. **Ричардсон Я.** Видеокодирование. H.264 в MPEG-4 — стандарты нового поколения. М.: Техносфера, 2005. 368 с.
16. **Гуткин Л. С.** Оптимизация радиоэлектронных устройств. М.: Сов. радио, 1975. 366 с.
17. **Окунев Ю. Б., Плотников В. А.** Принципы системного подхода к проектированию в технике связи. М.: Связь, 1976. 184 с.
18. **Моисеев Н. Н.** Математические задачи системного анализа. М.: Наука, 1981. 488 с.
19. **Колмогоров А. Н.** Теория информации и теория алгоритмов. М.: Наука, 1987. 304 с.
20. **Цыцулин А. К., Фахми Ш. С., Зубакин И. А.** Решения уравнения связи // Матер. конф. "Научно-технические проблемы в промышленности" к 100-летию НИИ "ВЕКТОР" / СПб.: 12—14 ноября 2008. С. 53—54.
21. **Клир.** Системология. М.: Радио и связь, 1990. 544 с.
22. **Трауб Дж., Васильковский Г., Вожьняковский Х.** Информация, неопределенность, сложность. М.: Мир, 1988. 184 с.
23. **Гонсалес Р., Вудс Р.** Цифровая обработка изображений. М.: Техносфера, 2005.
24. **Зубакин И. А., Цыцулин А. К.** Моделирование влияния ограничения сложности кодера на качество кодирования изображений с преобразованием // Вопросы радиоэлектроники. Сер. Техника телевидения, 2006. Вып. 2. С. 32—40.
25. **Reichel J., Ziliani F.** Method of selecting among N "spatial video codes" the optimum codes for a same input signal // International Application Number: PCT/IB2003/005852. Priority Data: 17.12.2002.
26. **Умбиталиев А. А.** Перспективы развития цифрового телерадиовещания: комплексное решение внедрения цифрового телевидения в регионах // Вопросы радиоэлектроники. Сер. Техника телевидения. 2008. Вып. 2. С. 3—8.
27. **Зубакин И. А., Фахми Ш. С.** Классификация нестационарных изображений и разработка методики оценки алгоритмов кодирования источника // Науч.-техн. вестник СПбГУ ИТМО. 2010. № 2(66). С. 54—59.
28. **Зубакин И. А., Фахми Ш. С., Шагаров С. С.** Адаптивные алгоритмы кодирования видеоинформации // Приборы. 2010. № 4. С. 28—31.

ИНФОРМАЦИЯ

И. П. Норенков, д-р техн. наук, проф.

Суперкомпьютеры списка TOP500

Опубликован очередной список 500 наиболее производительных суперкомпьютеров мира Top500 [1, 2]. В 2010 г. за полгода (с июня по ноябрь) в списке Top500 добавилось 195 новых суперкомпьютеров. Лидером в мире суперкомпьютеров в ноябре 2010 г. стал китайский компьютер Tianhe-1A производительностью 2,57 petaflop/s. На втором месте — лидер предыдущего списка Top500 Cray XT5 "Jaguar" (США) производительностью 1,75 petaflop/s. Третье место занимает опять же китайский суперкомпьютер Nebulae (1,27 petaflop/s). В первой десятке (Top10) находятся пять американских, два китайских и по одному компьютеру из Японии, Франции и Германии, причем японский суперкомпьютер Tsubame 2.0 занимает четвертое место. Из 500 компьютеров 274 находятся в США, 124 — в Европе, 84 — в Азии (в Китае 41, в Японии 26), 11 — в России. В список не вошли компьютеры с производительностью менее 31,1 teraflop/s. На 17-м месте — российский суперкомпьютер Ломоносов производительностью 350 teraflop/s на процессорах Intel EM64T Xeon X55xx (Nehalem-EP) 2930 MHz (11,72 Gflops) с памятью 54 Тбайт.

Среди вендоров лидером является IBM — 200 суперкомпьютеров, далее следуют Hewlett-Packard — 159 и Cray Inc. — 29 систем. В России система Ломоносов создана в компании Т-Платформы.

Преобладают системы с кластерной архитектурой (83 %) и с архитектурой MPP (16 %). В качестве коммутирующей среды используются Gigabit Ethernet (45,6 % систем) и Infiniband (42,6 %). Операционная система — или Linux (91,8%), или Windows (5%).

На процессорах Intel выполнены 79,6 % систем, преобладают процессоры Xeon E55xx (Nehalem-EP), Xeon E54xx (Harpertown) и Xeon X56xx (Westmere-EP), 11,4% — AMD Opteron, 8% — IBM Power. Процессоры с шестью и более ядрами имеются в 95 суперкомпьютерах, большинство построено на двухядерных СБИС. В китайских и японском суперкомпьютерах для увеличения производительности использованы графические процессоры NVIDIA. Число процессоров в Tianhe-1A равно 186368, в системе Ломоносов — 35360, большинство систем имеют число процессоров в диапазоне от 4 до 8 тысяч.

Средняя потребляемая мощность по компьютерам Top500 — 447 кВт (195 Mflops/W), а по Top10 — 3,2 МВт.

Для сравнения и иллюстрации прогресса в области суперкомпьютеров приведем данные по Top500 трехлетней давности. В список Top500 (ноябрь 2007 г.) были включены суперкомпьютеры с производительностью не менее 5,9 Tflop/s. Общая производительность всех 500 компьютеров составляла 6,97 Pflop/s. 100-й номер в списке имел производительность 9,29 Tflop/s. Семь позиций в списке занимали компьютеры, установленные в России, наиболее производительный среди них имел 33 Tflop/s и находился на 33 месте.

70,8% систем были построены на процессорах Intel, 15,6% — на AMD Opteron, 12,2% — на IBM Power. Наблюдалась тенденция к увеличению доли Intel. В основном использовались двухядерные процессоры.

Кластерами являлись 406 систем из 500. Внутренняя коммутация реализована на Gigabit Ethernet в 270 системах, Infiniband в 121 системе.

В качестве примера рассмотрим конструкцию лидера списка Top500 от ноября 2007 г. суперкомпьютера Blue Gene/L [3].

Базовым компонентом суперкомпьютера Blue Gene/L является так называемая вычислительная карта (*compute card*), которая состоит из двух вычислительных узлов, где каждый узел содержит два процессора PowerPC 440. 16 вычислительных карт группируются в модульную (или узловую) карту, которая, таким образом, содержит уже 64 процессора. В свою очередь, 16 модульных карт установлены на объединительной панели (*midplane*), и две таких панели смонтированы в серверную стойку (*cabinet*), которая содержит в итоге 1024 узла с общим числом процессоров 2048.

Процессор PowerPC 440 способен выполнять за такт 4 операции с плавающей запятой, что для заданной частоты соответствует пиковой производительности в 1,4 Tflop/s на одну объединительную панель, если считать, что на каждом узле установлено по одному процессору. Второй процессор на узле, идентичный первому, призван выполнять телекоммуникационные функции, т. е. он предназначен преимущественно для осуществления связи с другими процессорами суперкомпьютера. Кроме того, конструкцией суперкомпьютера предусмотрена установка некоторого числа двухпроцессорных узлов, которые занимаются исключительно операциями ввода/вывода.

Узлы суперкомпьютера связаны между собой пятью сетями:

- трехмерной тороидальной сетью для обмена данными непосредственно между блоками-узлами;
- сетью с топологией "дерева" для выполнения общих (глобальных) для всех блоков операций;
- сетью барьеров и прерываний;
- сетью Gigabit Ethernet-JTAG (Ethernet со специальным интерфейсом JTAG);
- еще одной сетью Gigabit Ethernet для подключения к главному (хост-) компьютеру, файловым серверам и другим системам.

Энергопотребление суперкомпьютера Blue Gene/L составляет порядка 1,6 МВт.

Источники

1. **Top500** Supercomputer Sites. — http://www.top500.org/
2. **Steen A., Dongarra J.** Overview of Recent Supercomputers. URL: http://www.top500.org/resources/orsc
3. **Сердюк Ю. П.** Кластерные вычисления // Интернет-университет информационных технологий. URL: http://www.intuit.ru/department/calculate/clusterexec/1/2.html

CONTENTS

tration at top levels is conducted on the basis of inquiries of users of the bottom level. Thus the Universal Declaration of Human Rights is not broken. The offered system is very flexible as all opinions of users in formation and realization of the policy of struggle against a spam are accepted without any restrictions. In the article the step-by-step algorithm of the considered approach is offered.

Keywords: spam, e-mail filtration, human rights, anti-spam policy, multilevel architecture

A useful in the computers numerations with the fixed point: a unsigned representation, a direct code, one's complement code and two's complement code, a bias code and a bias code with the negative zero are considered. A simplest of action for numbers in these codes are discussed for granted.

Keywords: computer, numbers, codes, fixed point, operations

A definition of a new modular logarithmic apparatus is given. Actual realization problem of one of the main non-modular operations — reverse conversion from modular logarithmic LG-code into traditional number system, is studied. Efficient DSP-oriented method of pipeline conversion from LG-code into binary notation is described.

Keywords: residue logarithmic number system, LG-code, reverse converter, digital signal processing, pipeline structure

The problem of multidimensional dynamical data classification is considered. The method of feature vector construction for multidimensional signal based on dynamical patterns revealing is described. The results of experimental investigations for applied problem of electroencephalogram classification are discussed.

Keywords: multidimensional signals, classification, dynamical pattern, computational electroencephalography

There are offered an effectiveness criterion of continuous messages source encoding in view of coder complexity and a functional, knotting the accuracy of continuous signal transmission with transmission rate and coder complexity, going back to their average harmonic. On concrete examples is shown the correctness of the offered method of the estimation of efficiency of approximation to an epsilon-enthropy of various methods of image encoding.

Keywords: code rate, complexity of encoding, quality criterion of encoding

Адрес редакции:

107076, Москва, Стромынский пер., 4

Телефон редакции журнала **(499) 269-5510**
E-mail: it@novtex.ru

Дизайнер *Т.Н. Погорелова*. Технический редактор *Е. В. Конова*.
Корректор *Т. В. Пчелкина*.
Сдано в набор 07.02.2011. Подписано в печать 21.03.2011. Формат 60×88 1/8. Бумага офсетная. Печать офсетная.
Усл. печ. л. 9,8. Уч.-изд. л. 11,20. Заказ 198. Цена договорная.

Журнал зарегистрирован в Министерстве Российской Федерации по делам печати, телерадиовещания и средств массовых коммуникаций.
Свидетельство о регистрации ПИ № 77-15565 от 02 июня 2003 г.

Отпечатано в ООО "Подольская Периодика"
142110, Московская обл., г. Подольск, ул. Кирова, 15