Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 11 № 2

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2021

Тематика номера:
Математическое моделирование физических процессов, оптимизация высокопроизводительных вычислений, оптимизация и обучение нейронных сетей, вопросы программирования

Журнал публикует оригинальные статьи по следующим областям исследований: математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:
Mathematical modeling of physical processes, optimization of high performance computing, optimization and learning process of neural networks, programming issues

The Journal publishes novel articles on the following research arias: mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies,
micro- and nanoelectronics, mathematical researches and problems of numerical analysis,
history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENT

Новый эффективный метод расчёта радиального электрического поля в тороидальной плазме

Ф.А. Аникеев[1], Ф.С. Зайцев[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, snowfed@gmail.com;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zaitsev@cs.msu.su

Аннотация. Обеспечение успешного функционирования термоядерного реактора требует детального моделирования радиального электрического поля E_r и изучения возможности управления его профилем и величиной. В работе предложена новая постановка задачи о вычислении поля E_r с использованием полулагранжева подхода. Поле определяется исходя из значений пяти- и шестимерных (5D и 6D) функций распределения, которые вычисляются как решение системы нелинейных кинетических уравнений. Разработаны новые численные алгоритмы для вычисления E_r. Алгоритмы реализованы в комплексе DiFF-SLPK. Показано хорошее количественное соответствие рассчитанного E_r известным ранее предельным случаям. Продемонстрировано качественное соответствие измерениям на установке AUG. С помощью мини-супер-ЭВМ НИИСИ РАН сделан прогноз для термоядерного реактора ITER. Расчёты показали заметное увеличение $|E_r|$ около границы плазмы в ITER по сравнению с известными ранее теоретическими оценками.

Ключевые слова: нелинейные кинетические уравнения, численный алгоритм, вычислительный эксперимент, термоядерный синтез, плазма тороидальной геометрии, радиальное электрическое поле.

1. Введение

Одна из основных задач научно-технологического развития РФ – переход к экологически чистой и ресурсосберегающей энергетике. В рамках решения данной задачи разрабатываются технологии управляемого термоядерного синтеза (УТС). Наиболее перспективным направлением в УТС являются системы удержанием плазмы магнитным полем тороидальной геометрии – токамаки [1, 2]. В настоящее время строится международный экспериментальный термоядерный реактор ITER (International Thermonuclear Experimental Reactor). Россия вносит существенный вклад в этот проект. Реактор должен продемонстрировать целесообразность использования термоядерного синтеза для получения энергии.

Важную информацию о процессах в тороидальной плазме дают теоретические исследования, поскольку на существующих установках не удаётся достичь параметров термоядерного разряда ITER. Современные супер-ЭВМ, параллельные технологии и специализированные программные комплексы позволяют перейти к реалистическому математическому моделированию УТС, избежать потери важных эффектов за счёт применения исходных физических законов без значительных упрощений [2].

В квазинейтральной плазме заряженные частицы перемещаются под действием электромагнитного поля. Это обуславливает возникновение сильного радиального электрического поля E_r, достигающего десятков кВ/м и существенно влияющего на многие процессы, например, радиальный перенос энергии [3, 2], уменьшение которого – одна из принципиальных проблем УТС, и на так называемый бутстреп-ток, дающий вклад более 30% в полный ток плазмы ITER и играющий значимую роль в реализации квазистационарных режимов. Кроме того, поле E_r играет важную роль в формировании периферийного транспортного барьера при переходе в режим улучшенного удержания плазмы (H-моду).

На данный момент для E_r чаще всего используются приблизительные оценки и полуаналитические формулы для предельных случаев, предложенные, например, в неоклассической теории [3]. В ряде работ результаты получены с помощью кодов, основанных на предположениях неоклассической теории [4, 5]. Также применяется подход, где для описания части физических эффектов реализуются полуаналитические аппроксимации [6].

Однако в общем виде задача определения радиального поля E_r не была решена из-за сложностей, связанных с необходимостью самосогласованного учёта движения ионов и электронов и высокоточного, до пяти значащих цифр, расчёта многомерных функций распределения частиц.

В данной статье на основе разработанного авторами ранее полулагранжевого подхода для моделирования кинетики заряженных частиц в тороидальной плазме [7] предложен новый метод расчёта радиального электрического поля, в котором применяется теория автоматического управления (АУ). Разработаны три подхода к определению E_r. Подходы реализованы программно в коде DiFF-SLPK (Distribution Function Finder – Semi-Lagrangian Particle Kinetics) с применением технологий распараллеливания MPI, OpenMP, OpenCL. Результаты расчётов сопоставлены с данными упрощённых моделей и измерениями в экспериментах. Существенно уточнено значение радиального электрического поля в термоядерном реакторе ITER.

2. Постановка математической задачи

Рассмотрим плазму, состоящую из двух сортов заряженных частиц: ионов i и электронов e. Задача для отыскания их шестимерных функций распределения f_α, $\alpha = i, e$ в полулагранжевом подходе может быть сформулирована в виде системы нелинейных кинетических уравнений [7]. Каждому сорту частиц α соответствует одно уравнение, записанное на траектории этой частицы. Его правая часть включает оператор кулоновских столкновений частицы сорта α со всеми сортами частиц плазмы β, включая $\beta = \alpha$, а также потери и источник частиц α. Задача нелинейная, т.к. коэффициенты кулоновского оператора зависят от искомой функции распределения. Формулы для коэффициентов приведены, например, в [2].

Траектория частицы определяется действием силы Лоренца. На временах, меньших характерного времени кулоновских столкновений, учитывать столкновения не требуется [7]:

$$\frac{d\vec{v}_\alpha}{dt} = \frac{e_\alpha}{m_\alpha}\left(\vec{E} + \vec{v}_\alpha \times \vec{B}\right), \qquad \frac{d\vec{r}_\alpha}{dt} = \vec{v}_\alpha. \tag{1}$$

Здесь e_α – заряд, m_α – масса частицы, $\vec{r}_\alpha$ и $\vec{v}_\alpha$ – её координаты в трёхмерных геометрическом и скоростном пространствах.

Расчёт функций распределения сводится к совместному решению системы двух нелинейных интегро-дифференциальных кинетических уравнений $\alpha = e, i$ с соответствующими граничными и начальными условиями [7] и системы 2-х ОДУ (1).

Для выделения эффекта возникновения радиального электрического поля E_r в чистом виде и сравнения результатов вычислений с данными обобщённой неоклассической теории влияние радиального электрического поля E_r на равновесие плазмы не учитывается. Тем не менее задача остаётся нелинейной относительно E_r, т.к. искомое поле E_r, входящее в систему уравнений через траектории (1), зависит от функций распределения, рассчитываемых по этой системе.

Рассмотрим три подхода к определению E_r. Сформулируем их в виде задач о нахождении минимума модуля функционала.

2.1. Первый подход

Первый подход основан на использовании теоремы Гаусса

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\varepsilon_0},$$

где ρ – плотность заряда, ε_0 – диэлектрическая проницаемость вакуума.

Запишем это уравнение в криволинейной системе координат (γ, ξ, η), где γ – переменная, характеризующая радиальное положение магнитной поверхности (метка магнитной поверхности), ξ и η – обобщённые полоидальный и тороидальный углы [2]. Якобиан преобразования декартовых геометрических координат к криволинейным (γ, ξ, η) обозначим $\sqrt{|g^{real}|}$.

Предполагая, что вектор $\vec{E}$ имеет только радиальную компоненту, получим

$$\frac{1}{\sqrt{|g^{real}|}}\frac{\partial}{\partial\gamma}\left(\sqrt{|g^{real}|}E_r(\gamma,\xi,\eta)\right) = \frac{1}{\varepsilon_0}\rho(\gamma,\xi,\eta)$$

или для усреднённого по углам (ξ, η) поля $E_r \approx E_r(\gamma)$ формулу (2).

Подчеркнём, что плотность заряда ρ зависит от E_r, поскольку равна сумме интегралов по пространству скоростей от функций распределения [2]

$$E_r(\gamma) - E_r(0)\int_{-\pi}^{\pi} d\xi \int_0^{2\pi} \frac{\left(\sqrt{|g^{real}|}\right)_{\gamma=0}}{\sqrt{|g^{real}|}} d\eta - \frac{1}{\varepsilon_0}\int_{-\pi}^{\pi} d\xi \int_0^{2\pi} \frac{\int_0^\gamma \rho(\gamma,\xi,\eta)\sqrt{|g^{real}|}d\gamma}{\sqrt{|g^{real}|}} d\eta = 0. \tag{2}$$

$$\rho = \sum\nolimits_{\alpha} e_\alpha \int f_\alpha \, d^3v \,,$$

а функции распределения зависят от E_r через траектории частиц (1).

Таким образом, в первом подходе задача сводится к поиску функции $E_r(\gamma)$, минимизирующей модуль невязки уравнения (2). С точки зрения физики процесса радиальное электрическое поле E_r обеспечивает близкие к нулю значения ρ [8].

2.2. Второй подход

Второй подход использует принцип квазинейтральности плазмы, см., например: [1, 8],

$$\rho(\gamma, \xi, \eta) \approx 0.$$

Для усреднённой по углам (ξ, η) плотности имеем формулу (3).

$$\int_{-\pi}^{\pi} d\xi \int_{0}^{2\pi} \rho(\gamma, \xi, \eta)\sqrt{|g^{real}|}d\eta \approx 0. \quad (3)$$

Во втором подходе искомая функция $E_r(\gamma)$ должна минимизировать модуль функционала в левой части (3).

2.3. Третий подход

$$j_i^{\gamma_0} = \frac{v_c^3}{\tau_c}\frac{\partial \gamma_0}{\partial v_j}\left(\frac{\partial f_i}{\partial v_k}\sum_{\beta} a_\beta^{jk}(f_\beta) + f_i \sum_{\beta} b_\beta^{j}(f_\beta)\right), \quad (4)$$

где a_β^{jk} и b_β^{j} – коэффициенты кулоновского оператора, по повторяющимся индексам j и k предполагается суммирование.

Поток дрейфовых объектов через магнитную поверхность γ определяется формулой (5), где $\sqrt{|g|}$ – якобиан преобразования к криволинейным шестимерным координатам, v – величина скорости, θ и φ – питч угол и гироугол в пространстве скоростей.

Из формул (4) и (5) видно, что поток $\Gamma_{i,0}(\gamma)$ зависит от функций распределения частиц и соответственно, через уравнение (1), от электрического поля E_r.

В результате неоклассическое условие на E_r формулируется в виде равенства нулю функционала.

$$\Gamma_{\alpha,0}(\gamma) = -\int_{-\pi}^{\pi} d\xi \int_{0}^{2\pi} d\eta \int_{0}^{\infty} dv \int_{0}^{\pi} d\theta \int_{0}^{2\pi} d\varphi \frac{\partial \gamma}{\partial \gamma_0} j_\alpha^{\gamma_0}\sqrt{|g|}\,. \quad (5)$$

В третьем подходе применяется неоклассическая теория, согласно которой поле E_r перестраивается так, чтобы обеспечить нулевой радиальный поток ионов [9, 10]. Величина потоков ионов и электронов через поверхность $\gamma = \text{const}$ определяется кулоновским взаимодействием заряженных частиц и их движением под действием силы Лоренца . В [2, п. 1.2.15] приведены формулы, описывающие потоки как в фиксированный момент времени, так и на относительно больших промежутках времени.

Формулы для фиксированного момента времени содержат быстро осциллирующий множитель $d\gamma\ /\ dt$, что сильно затрудняет их использование при численном решении. Однако наибольший практический интерес представляет расчёт E_r на временах, значительно превосходящих характерное время движения частицы по ларморовской окружности. Поэтому здесь воспользуемся формулами для радиального потока дрейфовых объектов.

Поток по переменной γ_0 («медленная» переменная, характеризующая радиальное положение дрейфовой траектории частицы как единого объекта) определяется в заданной точке пространства выражением

$$\Gamma_{i,0}(\gamma) = 0, \quad (6)$$

в котором γ – параметр.

Как отмечено выше, плотность заряда ρ обычно очень мала из-за квазинейтральности плазмы. Это значительно усложняет численное решение задачи в первом и втором подходах, где для определения ρ требуется вычислять функции распределения электронов и ионов с высокой точностью. Наиболее эффективным и целесообразным для практического применения оказался третий подход. В нём не требуется расчёт ρ, он намного менее чувствителен к изменению числа узлов сетки и шага по времени и в то же время даёт результаты близкие к получаемым в первых двух подходах.

3. Метод расчёта E_r

Рассмотренные в п. 2. подходы к определению радиального электрического поля сводятся к нахождению такого E_r, при котором некоторая величина h минимальна:

$$E_r = argmin|h|.$$

В качестве h выбирается либо невязка уравнения (2), либо ρ, либо $\Gamma_{i,0}(\gamma)$.

В такой постановке задача может быть решена методами АУ, если h рассматривать в каче-

стве вектора состояний (отклонений), а приращение E_r – как управление.

Методы АУ имеют существенное преимущество перед другими методами минимизации. В частности, они позволяют избежать появления искусственных осцилляций поля E_r и плазмы по времени, характерных для методов с поиском точного минимума. Кроме того, алгоритмы АУ достаточно быстро работают, надёжны и универсальны.

Представим объект управления в форме, принятой в теории управления. Пусть $\vec{h} = (h_1, \dots, h_N)^T$ – разница между желаемыми значениями параметров объекта и реальными значениями. В нашем случае вектор $\vec{h}$ определён на сетке по переменной γ: $\vec{\gamma} = (\gamma_1, \dots, \gamma_N)$. Задача управления состоит в минимизации нормы $\vec{h} = (h_1, \dots, h_N)$.

Характерное время установления E_r много меньше характерного времени кулоновских столкновений. Поэтому в численном алгоритме в рамках одного столкновительного шага можно решать отдельную задачу управления.

В соответствии с физическими свойствами объекта управления имеем функциональную зависимость $\vec{h} = \vec{h}(\vec{E})$, $\vec{E} = (E_{r,1}, \dots, E_{r,N})$. Раскладывая $\vec{h}$ по формуле Тейлора и отбрасывая остаточный член, приходим к дискретному уравнению состояния объекта управления:

$$\vec{x}[n+1] = A\vec{x}[n] + B\vec{u}[n], \qquad (7)$$

где $\vec{x} = \vec{h}$ – вектор состояний, $\vec{u} = \delta\vec{E}$ – вектор управлений, $A = I$ – единичная матрица, $B = \partial\vec{h}/\partial\vec{E}$ – матрица Якоби.

В общем случае матрица управления B зависит от величины электрического поля, но в данной задаче эта зависимость слабая, и в рамках одного столкновительного шага по времени B можно считать неизменной.

В предположении о влиянии электрического поля $E_{r,i}$ в основном на h_i, матрица B упрощается

$$B = \mathrm{diag}\left\{\frac{\partial h_1}{\partial E_{r,1}}, \dots, \frac{\partial h_N}{\partial E_{r,N}}\right\}.$$

Получаем уравнение (7), в котором значение h_i определяется данными только в одном узле γ_i.

Для решения задачи управления разработано множество методов. Воспользуемся линейно-квадратичным регулятором (LQR), см., например, [11]. В LQR минимизируется функционал

$$J = \sum_{n=0}^{\infty}\left((\vec{x}[n])^T Q(\vec{x}[n]) + (\vec{u}[n])^T R(\vec{u}[n])\right)$$

и управление осуществляется по обратной связи

$$\vec{u}[n] = -K\vec{x}[n],$$

где K – контроллер, вычисляемый в LQR-алгоритме.

Для рассматриваемой задачи можно положить $Q = I$, а R выбрать из соображений близости масштабов слагаемых под знаком суммы: $R = cI$, c – константа.

В случае диагональных матриц можно доказать утверждение, позволяющее свести поиск LQR-контроллера к решению N независимых квадратных уравнений.

Утверждение. Если A, B, Q и R – диагональные матрицы размера $N \times N$, $Q > 0$ и $R > 0$, то матрица $P \equiv diag\{p_{11}, p_{22}, \dots, p_{NN}\}$, где $p_{ii} = r_{ii}\beta_{ii} + \mathrm{sgn}(a_{ii})\sqrt{(a_{ii,-}^2 + \beta_{ii})(a_{ii,+}^2 + \beta_{ii})}/(2\mathrm{b}_{\mathrm{ii}}^2)$, если $a_{ii} \neq 0$; $p_{ii} = q_{ii}$, если $a_{ii} = 0$; $a_{ii,-} \equiv a_{ii} - 1$, $a_{ii,+} \equiv a_{ii} + 1$, $\beta_{ii} \equiv q_{ii}\, b_{ii}^2/r_{ii}$; является решением дискретного уравнения Риккати $P = A^T PA - A^T PB(R + B^T PB)^{-1}B^T PA + Q$, причём контроллер $K = (R + B^T PB)^{-1}B^T PA$ – стабилизирующий.

4. Параллельный алгоритм

В работе [7] предложен эффективный параллельный метод решения системы кинетических уравнений для функций распределения частиц. Представленный в п. 3 алгоритм расчёта E_r методом АУ не влияет на характеристики метода [7]. Однако теперь в рамках одного столкновительного шага требуется сделать N_E шагов управления. На каждом из них для вычисления вектора состояний $\vec{h} = \vec{h}(\vec{E})$ необходимо решить систему кинетических уравнений. В результате количество операций при расчёте E_r возрастает в N_E раз по сравнению с [7].

Расчёт E_r с применением третьего подхода позволяет уменьшить в несколько раз число арифметических операций за счёт использования в алгоритме АУ при решении системы кинетический уравнений более грубой сетки по геометрическому пространству. Погрешность практически не накапливается, если на каждом столкновительном шаге после определения E_r провести решение системы на сетке с исходным числом узлов.

В этом заключается принципиальное преимущество третьего подхода по сравнению с первыми двумя, которые, опираясь на расчёт ρ, наоборот, налагают повышенные требования на число узлов сетки в алгоритме АУ.

Все три похода дают схожие результаты при решении модельных задач. Поэтому на практике третий подход имеет значительное преимущество по быстродействию.

Наиболее времяёмкой частью общего алгоритма является расчёт траекторий частиц. В работе [7] предложен эффективный метод расчёта траекторий частиц для произвольного числа одинаковых видеокарт. Однако на практике приходиться проводить расчёты на вычислительных системах, имеющих видеокарты разной производительности. Например, авторы использовали систему с тремя видеокартами: две NVIDIA GTX 1080 и одна RTX 2070. При расчёте траекторий RTX 2070 уступает по скорости NVIDIA GTX 1080 на ~10%. В результате равномерная загрузка видеокарт не является оптимальной.

Для повышения производительности вычислительной системы реализован подход, учитывающий реальную скорость вычислений различных видеокарт. На первом шаге по времени вычислительная нагрузка равномерно распределяется между видеокартами. По его завершении определяется скорость работы (число операций в секунду) каждой из карт. На последующих шагах по времени полученная разница в скорости работы учитывается при нагрузке видеокарт. Для указанных видеокарт данный алгоритм ускорил работу всего кода на ~6%. Соответствующее уменьшение времени расчётов становится заметным для вычислений, занимающих многие часы.

Предложенная оптимизация загрузки видеокарт различного типа позволяет полноценно использовать и эффективно, в смысле отношения производительности к цене, развивать аппаратную часть мини-супер-ЭВМ НИИСИ РАН для решения задач УТС.

5. Программное обеспечение и результаты расчётов

Предложенный алгоритм АУ для всех трёх подходов к определению E_r реализован программно в коде DiFF-SLPK, являющимся частью специализированного программного комплекса DiFF решения кинетических уравнений. Использованы языки Fortran 2008 и C++11, объектно-ориентированная парадигма и параллельные технологии MPI, OpenMP и OpenCL. Проведение и анализ расчётов по коду DiFF-SLPK поддерживаются с помощью удобного пользовательского интерфейса, реализованного на языке Python 3. Общий объём DiFF-SLPK составляет более 26 тысяч строк.

С учётом опыта математического моделирования в области УТС, при разработке программного обеспечения (ПО) сделан акцент на кроссплатформенность. Fortran позволяет использовать код на разных вычислительных установках без какой-либо корректировки. OpenCL, в отличие от CUDA, не привязывает ПО к графическому ускорителю конкретного производителя. C++-код в DiFF-SLPK применяется только для работы с видеокартами на стороне центральных процессоров и также является кроссплатформенным.

Корректность предложенной формулировки задачи и разработанного программного обеспечения проверена на решении модельных задач. Показано соответствие расчётов известным аналитическим формулам, полученным в предельных случаях. Все три подхода из п. 2 на модельных задачах дают близкие результаты, однако, как уже отмечалось, третий подход имеет значительные преимущества по скорости и численной точности нахождения E_r, поэтому именно он применялся для получения физических результатов.

Проведены расчёты E_r для характерных параметров установок AUG, JET, MAST и сделан прогноз для ITER. Установлено, что в общем случае аккуратный учёт E_r может приводить к принципиальному изменению свойств плазмы, в том числе величины и распределения бутстреп-тока по радиусу.

В данной работе остановимся более подробно на исследованиях, проведённых для AUG и ITER.

Результаты расчётов E_r по коду DiFF-SLPK сопоставлены с экспериментальными данными установки AUG. Использовались параметры разряда #26598 AUG с режимом улучшенного удержания [12]. Равновесие плазмы не приведено в [12] и рассчитывалось с помощью комплекса имитационного моделирования HASP CS [13, 14], разрабатываемого в НИИСИ РАН. В HASP CS реализована и верифицирована развитая самосогласованная модель тороидальной плазмы, в которой учитывается множество физических эффектов и характерных закономерностей в токамаках. Время расчёта E_r на мини-супер ЭВМ НИИСИ РАН с двумя процессорами Intel Xeon E5-2680V4, двумя видеокартами GPU NVIDIA GTX 1080 и одной картой NVIDIA RTX 2070 составляет около двух с половиной часов.

На Рис. 1 показано поле E_r, рассчитанное по коду DiFF-SLPK и найденное экспериментально [12]. По оси абсцисс отложен нормированный полоидальный поток ρ_{pol} (как в [12]), характеризующий расстояние от цента плазмы. На качественном уровне наблюдается схожее поведение значений поля. Резкое убывание E_r в приграничной области соответствует граничному транспортному барьеру (ETB), присутствующему в рассматриваемом разряде. Данный результат показывает возможность использования предложенного подхода для моделирования перехода плазмы в режим с транспортным барьером.

На графике наблюдается некоторое количественное различие в величине E_r. Это различие обусловлено физическими эффектами, не учитываемыми в данной работе: в эксперименте присутствует нагрев инжекцией нейтральных атомов (NBI) и примеси. Тем не менее, математическая модель, использованная в DiFF-SLPK, допускает учёт таких эффектов. Программная реализация соответствующих опций планируется в перспективе.

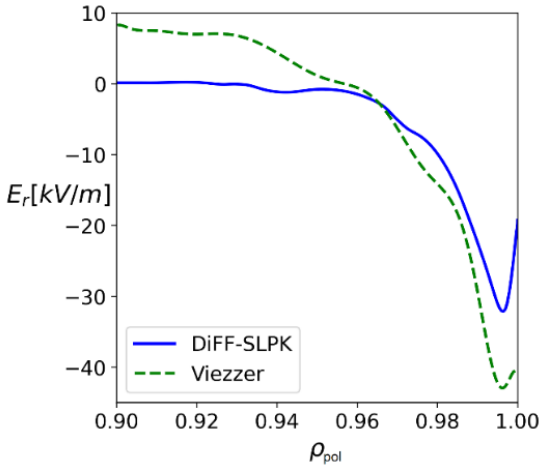


Рис. 1. Электрическое поле E_r в AUG около границы плазмы как функция нормированного полоидального потока ρ_{pol}.

Получение точного прогноза E_r для ITER, особенно в приграничной области, осложняется необходимостью учёта множества эффектов. В частности, требуется учёт МГД-активности плазмы и распределений альфа-частиц и примесей во всём фазовом пространстве (геометрическом и скоростном). Кроме того, около дивертора и в приграничной области необходимо точно описывать специфические физические процессы.

В рамках данной работы получена оценка E_r в ITER по порядку величины для омического сценария [www.iter.org]. Длительность одного расчёта на мини-супер-ЭВМ НИИСИ составляет более 3-х часов. Найденное E_r (Рис. 2) больше (по абсолютной величине) известных упрощённых неоклассических результатов в среднем на ~15%. Около границы плазмы различие в абсолютных величинах выше среднего. Отличие объясняется аккуратным учётом в предложенном методе отклонений траекторий ионов от магнитных поверхностей. Увеличение E_r необходимо принимать во внимание при обеспечении квазистационарного режима работы реактора.

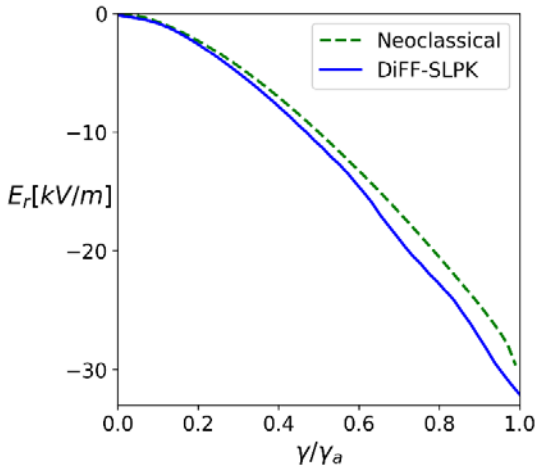


Рис. 2. Электрическое поле E_r в ITER. Отношение $\gamma\ /\ \gamma_a$ характеризует расстояние от центра до границы плазмы.

6. Заключение

Разработан новый эффективный метод определения радиального электрического поля в тороидальной плазме на основе решения системы нелинейных 5D и 6D кинетических уравнений. Показано хорошее качественное соответствие расчётов измерениям на установке AUG.

В условиях ITER установлено заметное увеличение радиального электрического поля $|E_r|$ около границы плазмы по сравнению с известными ранее теоретическими оценками. Возмущения плазмы при наличии такого поля могут привести к значительному нежелательному радиальному перемещению заряженных частиц. Поэтому для обеспечения успешной работы ITER необходимо аккуратное предсказательное моделирование E_r на основе точных моделей плазмы, позволяющих выработать адекватные рекомендации по управлению величиной и распределением E_r по радиусу.

Код DiFF-SLPK включён в программный комплекс НИИСИ РАН HASP CS [13, 14], где используется для вычисления бутстреп-тока в плазме с аккуратным расчётом радиального электрического поля по методике, представленной в данной работе.

Авторы признательны академику РАН В.Б. Бетелину и зав. отделом А.Г. Кушниренко за внимание к исследованиям и обсуждение полученных результатов.

A New Efficient Method for Calculating Radial Electric Field in Toroidal Plasma

Fedor Anikeev, Fedor Zaitsev

Abstract. A successful operation of a thermonuclear power plant is a particularly complex process. Ensuring such an operation requires detailed modelling of the radial electric field E_r as well as studying the possibility of its automated control, both shape and magnitude. Here a new problem formulation is presented for computing the field using semi-Lagrangian approach. The field is calculated based on the values of five- or six-dimensional (5D and 6D) distribution functions, which are obtained as the solution of a system of nonlinear kinetic equations. New numerical algorithms for computing E_r have been developed. The algorithms are implemented in DiFF-SLPK code. The results for E_r match quantitatively the previously known limit cases. A qualitative correspondence to the measurements at AUG plant is shown. A forecast is presented for thermonuclear reactor ITER, which has been computed on a mini-supercomputer of NIISI RAS. A significant increase in $|E_r|$ near the plasma boundary is shown in comparison with the previously known theoretical estimates.

Keywords: nonlinear kinetic equations, numerical algorithm, computer experiment, thermonuclear fusion, plasma of toroidal geometry, radial electric field.

Литература

1. Ю.Н. Днестровский, Д.П. Костомаров. Математическое моделирование плазмы. 2-е изд. М.: Наука, 1993. 336 с.

2. F.S. Zaitsev. Mathematical modeling of toroidal plasma evolution. English ed. M.: MAKS Press, 2014. 688 p.

3. C.S. Chang, F.L. Hinton. Effect of finite aspect ratio on the neoclassical ion thermal conductivity in the banana regime. Phys. Fluids, Vol. 25, No. 9, 1982. P. 1493- 1494.

4. T.P. Kiviniemi, J.A. Heikkinen, A.G. Peeters. Neoclassical Radial Electric field and Ion Heat Flux in the Presence of the Transport Barrier. Contrib. Plasma Phys. Vol 42, No. 2-4. 2002. P. 236-240.

5. M. Honda et. al. Integrated modelling of toroidal rotation with the 3D non-local drift-kinetic code and boundary models for JT-60U analyses and predictive simulations. Nuclear Fusion. Vol 55, No. 7, 2015. 073033 (11 P).

6. Y. Pianroj, S. Jumrat. Full Radial Electric Field Calculation for Predicting Pedestal Formation in H-mode Tokamak Plasma by using BALDUR code. Thammasat International Journal, Vol. 19, No. 2, 2014, P. 75-81.

7. Ф.С. Зайцев, Ф.А. Аникеев. Эффективный параллельный алгоритм расчета электрических токов в тороидальной плазме. Доклады Академии наук. — 2018. — Т. 482, No 1. — С. 19–22.

8. J. Wesson. Tokamaks. 3 ed, Clarendon Press-Oxford, 2004, 762 p.

9. F.L. Hinton, R.D. Hazeltine. Theory of plasma transport in toroidal confinement systems. Reviews of Modern Physics, Vol. 48, No. 2, 1976. P. 239-308.

10. J.W. Connor. The neo-classical transport theory of a plasma with multiple ion species. Plasma Phys., Vol. 15, 1973. P. 765-782.

11. K. Ogata. Discrete-Time Control Systems. 2nd ed. Englewood Cliffs: Prentice Hall Internat. Inc., 1995. 744 p.

12. E. Viezzer. Radial electric field studies in the plasma edge of ASDEX Upgrade. München: Universität München, 2012. 113 p.

13. F.S. Zaitsev, A.G. Shishkin, A.A. Lukianitsa, E.P. Suchkov, S.V. Stepanov and F.A. Anikeev. The Basic Components of Software-Hardware System for Modeling and Control of the Toroidal Plasma by Epsilon-Nets on Heterogeneous Mini-Supercomputers. // Commun. Comput. Phys. 2018, v. 24, N 1, p. 1–26.

14. A.G. Shishkin, S.V. Stepanov, E.P. Suchkov, F.A. Anikeev, F.S. Zaitsev. Distributed Access to the Resources of Plasma Modelling and Control Complex HASP CS. // Proc. 2018 IEEE International Conference on Computer and Communication Engineering Technology. — Beijing, 2018 — P. 193-197.

Оптимизации, применяемые к графу потока управления программы для повышения эффективности векторизации плоских циклов

Б.М. Шабанов[1], А.А. Рыбаков[2], А.Д. Чопорняк[3]

[1]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, shabanov@jscc.ru;

[2]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, rybakov@jscc.ru, +7 903 138-88-77;

[3]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, adc@jscc.ru

Аннотация. Повышение эффективности суперкомпьютерных расчетов является актуальной задачей ввиду постоянного усложнения моделей, увеличения размеров расчетных сеток и количества обрабатываемых данных. Для удовлетворения актуальных потребностей в вычислительных ресурсах для решения практических и научных задач требуется проведение оптимизации высокопроизводительных вычислений на всех уровнях: распределение вычислений между узлами суперкомпьютерного кластера, оптимизация работы многопоточных алгоритмов вычислений для систем с общей памятью, оптимизация расчетов внутри одного потока вычислений. Векторизация программного кода является низкоуровневой оптимизацией, позволяющей в несколько раз ускорить исполнение горячих участков путем объединения нескольких экземпляров фрагмента программы для одновременного выполнения на одном процессорном ядре. В данной статье описан подход к векторизации программного контекста специального вида, называемого «плоский цикл». При этом особое внимание уделяется оптимизациям графа потока управления с известным профилем исполнения для уменьшения издержек, связанных с наличием обильного управления внутри таких циклов.

Ключевые слова: векторизация, AVX-512, плоский цикл, граф потока управления программы, профиль исполнения программы, слияние линейных участков, локализация маловероятных ветвей исполнения

1. Введение

Векторизация вычислений является одной из наиболее важных низкоуровневых оптимизаций программного кода, позволяющей кратно повысить производительность суперкомпьютерных приложений. С помощью векторизации выполняется объединение однотипных операций со скалярными операндами в векторные инструкции, позволяющие одновременно выполнять данные операции над всеми элементами операндов-векторов. В настоящее время векторные инструкции присутствуют практически во всех современных архитектурах микропроцессоров: SVE (Scalable Vector Extension) в архитектуре ARM [1], инструкции AltiVec и VMX (Vector & Media Extension) в архитектуре POWER [2], расширения MMX (Multimedia Extension), SSE (Streaming SIMD Extension) и AVX (Advanced Vector Extension) в микропроцессорах Intel [3], упакованные инструкции в архитектуре «Эльбрус» [4]. Однако наиболее продвинутым набором инструкций для реализации векторных вычислений является расширение AVX-512 в микропроцессорах Intel, состоящее из векторных операций, которые работают с 512-битными регистрами zmm (расширения 256-битных ymm регистров из набора инструкций AVX). Данный набор инструкций имеет ряд уникальных особенностей, позволяющих создавать высокоэффективный параллельный код [5]. Наиболее важной особенностью данных инструкций, отличающей их от векторных инструкций в других архитектурах, является возможность выборочного применения операции к элементам векторов. Такое выборочное применение реализуется с помощью использования специальных масочных регистров, которые могут выступать в качестве операндов у большинства инструкций из набора AVX-512. Всего существует 8 масочных регистров (k0-k7). При осуществлении поэлементной векторной операции значение бита в используемом масочном регистре определяет, требуется ли применить операцию к соответствующим элементам векторных операндов (если значение бита маски равно 1, то операция выполняется).

Набор инструкций AVX-512 впервые появился в системе команд микропроцессоров Intel Xeon Phi KNL (Knights Landing) [6]. До этого векторные инструкции присутствовали также в

сопроцессорах Intel Xeon Phi KNC (Knights Corner), однако они не были x86-совместимыми в отличие от KNL. После этого расширение AVX-512 твердо заняло место в наборах инструкций линейки микропроцессоров Intel Xeon (Skylake, Cascade Lake и далее), по мере развития наполняясь новыми инструкциями. В настоящее время набор инструкций состоит из нескольких наборов, реализованных в разных семействах микропроцессоров. Вот только некоторые из них: AVX-512 Foundation – базовые операции для работы с 32-битными и 64-битными данными, включая поэлементные операции над векторами, операции с масками, слияние векторов по маске, сравнение векторов, операции перестановок, конвертации и другие; AVX-512 Conflict Detection – набор операций, предназначенный для разрешения конфликтов при векторизации циклов с помощью динамической проверки диапазонов адресов на пересечение; AVX-512 Exponential and Reciprocal – набор инструкций для поэлементного вычисления с повышенной точностью функций 2^x, x^{-1}, $x^{-0.5}$; AVX-512 Prefetch – инструкции предварительной подкачки данных для операций gather/scatter в микропроцессоре Intel Xeon Phi KNL; AVX-512 Vector Length – расширение многих инструкций на элементы данных размера 128 и 256 бит; AVX-512 Doubleword and Quadword – дополнительные инструкции для работы с элементами данных размера 32 и 64 бита; AVX-512 Byte and Word – расширение набора инструкций для работы с данными размера 8 и 16 бит; AVX-512 Vector Neural Network Instructions – дополнительные инструкции для оптимизации алгоритмов машинного обучения.

Условно по схеме работы множество инструкций AVX-512 можно разделить на несколько групп. В качестве первой группы рассмотрим операции, получающие на вход один операнд – zmm регистр и вырабатывающие один результат – также zmm регистр. При выполнении таких векторных операций к каждому элементу входного вектора применятся определенная функция, а после ее вычисления результат записывается в выходной регистр (если соответствующий бит маски выставлен в единицу). Примерами таких операций являются получение абсолютного значения, извлечение корня, операции сдвигов на константу и другие. Другой группой операций являются операции, получающие на вход два векторных операнда. Отличием от первой группы является только то, что операции, применяемые к элементам векторов, принимают два аргумента вместо одного: операции сложения, вычитания, умножения, деления и другие. В качестве следующей группы можно выделить операции поэлементной конвертации векторов из одного типа данных в другой (множества инструкций cvt и pack). Важной группой операций являются векторные операции поэлементного вычисления значений вида $\pm a \cdot b \pm c$. Также отдельной группой операций являются операции перестановки элементов векторов; они не выполняют никаких арифметических действий, а меняют порядок расположения элементов внутри векторов (к таким операциям относятся разнообразные инструкции unpck, shuf, align, blend, perm). Среди других специальных операций можно отметить операции по множественному доступу в память одновременно к элементам данным, расположенным не последовательно, а по произвольным адресам, операции пересылки данных с дублированием, операции предварительной подкачки данных в кэш и многие другие.

Использованию специфических свойств векторных операций в мире уделяется достаточно много внимания. Можно отметить работы, направленные на повышение эффективности векторизации различных высокопроизводительных приложений: ускорение программного кода LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [7], использование масочных операций для векторизации циклов с нерегулярным числом итераций на примере построения множества Мандельброта [8], векторизация упрощенного римановского решателя для численного решения уравнений мелкой воды [9,10], создание векторизованных версий сортировки массивов чисел [11], векторизация операций над разреженными матрицами [12,13], оптимизация других прикладных пакетов высокопроизводительных вычислений [14,15].

2. Плоский цикл как удобный для векторизации контекст

Векторизация программного контекста не может быть применена автоматически к коду произвольного вида. В данном разделе определим подходящих для векторизации программный контекст специального вида – плоский цикл – и опишем его свойства.

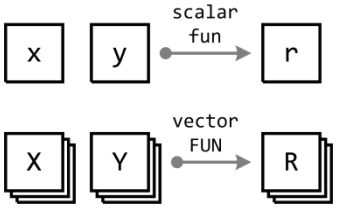


Рис. 1. Объединение нескольких экземпляров вызова функции fun для формирования векторной версии функции FUN.

На рис. 1 сверху представлена схема функции fun, которая получает на вход два аргумента x и y и на основании них формирует результат r. Будем считать, что данная функция является чистой, то есть результат ее выполнения зависит только от значений x и y (например, отсутствуют побочные эффекты через глобальную память или операции ввода-вывода). Идеология чистых вычислений берет свое начало из парадигмы функционального программирования [16], использование такого подхода открывает возможности для оптимизации программного кода, в частности для компиляторов. Если рассмотреть вместо одного вызова функции fun несколько вызовов (w штук) с разными наборами входных параметров, то их можно трактовать как вызов некоторой функции FUN, входными параметрами которой являются векторы X и Y длины w, а выходным значением является вектор R также длины w. В данном случае можно говорить, что функция FUN представляет собой реализацию векторизованной функции fun при ширине векторизации w (рис. 1 снизу). В этом случае семантику функции FUN можно записать в виде

```
for (int i = 0; i < w; i++)
{
    r[i] = fun(x[i], y[i])
}
```

Цикл такого вида будем называть плоским циклом. Определим свойства, присущие плоскому циклу. Прежде всего условимся считать, что плоский цикл это цикл for, индуктивная переменная которого меняется от 0 до w - 1, где w – ширина векторизации (например, при работе с набором инструкций AVX-512 и с вещественными значениями формата single precision ширина векторизации равна 16, то есть один zmm регистр вмещает 16 значений). Во-вторых, будем считать, что внутри плоского цикла на i-ой итерации все обращения в память (и вообще все обращения к глобальным данным) имеют вид a[i]. Такое ограничение гарантирует отсутствие конфликтов между итерациями по обращениям в память. Таким образом, итерации плоского цикла становятся независимыми между собой, а значит могут выполняться в произвольном порядке, в том числе и одновременно.

Плоские циклы, обладающие описанными свойствами, представляют собой удобный контекст для векторизации, и в большинстве случаев они могут быть векторизованы с помощью векторных инструкций AVX-512 с помощью перевода тела цикла в предикатное представление и замены скалярных инструкций векторными аналогами, реализованными с помощью функций-интринсиков [17,18].

Многие практические вычислительные задачи состоят из выполнения однотипных вычислений, применяемых к разным наборам данных, которые можно сгруппировать, трансформировав в плоский цикл, как это продемонстрировано на рис. 1 на примере функции fun.

Сложность векторизации тела полученного плоского цикла зависит от особенностей исходной функции fun. Чем сложнее управление внутри тела векторизуемого плоского цикла, тем больше трудностей может возникнуть в процессе выполнения векторизации. Для оценки сложности структуры векторизуемого тела цикла требуется построить граф потока управления данного цикла.

3. Представление структуры тела плоского цикла в виде графа потока управления

Граф потока управления (control flow graph, CFG) является одним из видов промежуточного представления программы, которые используются в частности для выполнения оптимизаций исполняемого кода [19]. Узлами данного графа являются линейные участки, состоящие из последовательностей инструкций, а ребрами – передача управления между этими линейными участками [20]. Граф потока управления является логической структурой, отражающей параллелизм программы на уровне линейных участков. Он активно используется компилятором для применения различных глобальных оптимизаций [21].

Кроме самой структуры графа потока управления для оптимизации программного кода важна статистическая информация об исполнении программы: количество выполнений различных линейных участков и данные о частоте переходов между ними. Такая статистическая информация называется профилем исполнения, и для корректного проведения оптимизаций требуется правильным образом собирать и корректировать данный профиль [22]. Для векторизации программного кода профиль исполнения программы приобретает особенную важность, так как на эффективность векторизации сильно влияет даже структура векторных масок, которая сильно разнится при сравнении результатов, полученных при генерации случайных входных данных, и при использовании реальных данных расчетов [23].

Мы в качестве профиля исполнения приложения будем использовать данные только счетчики узлов и ребер, а также вероятности ребер. Обозначим некоторый узел CFG через v. Пусть в

него входят несколько ребер $E_i(v)$, а также выходят несколько ребер $E_o(v)$ (рис. 2). Счетчиком произвольного ребра e будем называть количество переходов по этому ребру в процессе выполнения программы (значение счетчика ребра будем обозначать n(e)). Счетчиком узла будем называть суммарное количество счетчиков всех входных ребер либо всех выходных ребер (для внутренних узлов CFG эти значения совпадают).

$$n(v) = \sum_{e \in E_i(v)} n(e) = \sum_{e \in E_o(v)} n(e)$$

Вероятностью ребра будем называть отношение счетчика данного ребра к счетчику узла, из которого это ребро выходит.

$$p(e \in E_o(v)) = n(e)/n(v)$$

Построенный по телу плоского цикла граф потока управления с собранным профилем исполнения будем использовать для принятия решения о выборе методов векторизации программного контекста.

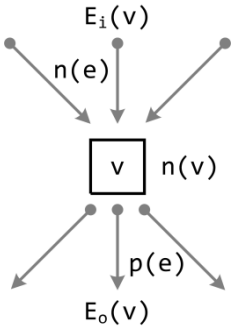


Рис. 2. Определение счетчиков узла и ребра, а также вероятности ребра.

Тривиальной разновидностью CFG является граф, состоящий из единственного линейного участка. Это означает, что в теле рассматриваемого плоского цикла отсутствуют операции передачи управления. Векторизация для таких видов цикла не представляет никаких проблем, она выполняется любым оптимизирующим компилятором (в частности gcc [24] или icc [25]) автоматически.

Если же CFG является сложным графом, то принятие решения о векторизации не всегда может быть принято компилятором. В некоторых случаях приходится прибегать к дополнительным глобальным оптимизациям, связанным с модификацией CFG. Проблема заключается в том, что переходы между линейными участками выполняются по предикатам, которые строятся исходя из условий между элементами векторов, то есть данные условия принципиально являются скалярными. При векторизации же вычислений мы сталкиваемся с ситуациями, что при выполнении векторных операций для одних элементов вектора условия переход должен быть выполнен, а для других не должен, что одновременно невозможно. На помощь в данном случае и приходят оптимизации графа потока управления.

4. Оптимизации графа потока управления для обеспечения векторизации плоского цикла

Наиболее простой и прямолинейной оптимизацией CFG для обеспечения векторизации является слияние параллельных ветвей исполнения. Рассмотрим слияние двух альтернатив одного условия, записанного в виде

```
if (cnd)
{
    block1
}
else
{
    block2
}
```

В данном случае при выполнении условия cnd будет исполнен участок кода block1, в противном случае будет выполнен участок кода block2. Пусть у нас собран профиль исполнения программы, и известны вероятности передачи управления на участки block1 и block2 (они равны p_1 и $p_2 = 1 - p_1$ соответственно). Также пусть известны времена выполнения t_1 и t_2 данных линейных участков в некоторых единицах измерения (например, в количестве инструкций). В этом случае общее время работы плоского цикла с приведенным телом равно $T_1 = (p_1 t_1 + p_2 t_2)w$. Для возможности проведения векторизации тело цикла необходимо переписать в предикатной форме [26], в которой операции перехода заменены на операции под предикатами, имеющие векторные аналоги в наборе инструкций AVX-512. Условно тело цикла может быть переписано в следующем виде:

```
block1 ? CND
block2 ? ~CND
```

При успешной векторизации данного фрагмента время работы векторизованного плоского цикла составит $T_w = t_t + t_2$. При этом скалярные операции заменены на векторные аналоги, однако оба линейных участка теперь должны выполняться под предикатами, а значит они будут занимать расчетное время. Ожидаемое ускорение от применения векторизации в данном случае равно

$$S = \frac{T_1}{T_w} = \frac{(p_1 t_1 + p_2 t_2) w}{t_1 + t_2}$$

Заметим, что данная оценка верна в предположении, что сами линейные участки block1 и block2 не содержат операций передачи управления и векторизуются.

В общем случае, можно привести оценку эффективности векторизации методом слияния всех ветвей исполнения. Пусть тело плоского цикла состоит из n линейных участков одной длины, и все они равновероятны, то есть выполняются с вероятностью $p = n^{-1}$. Тогда теоретическое ускорение от применения векторизации составит

$$S = \frac{\left(\sum_{i=1}^{n} p_i t_i \right) w}{\sum_{i=1}^{n} t_i} = \frac{w}{n}$$

Видно, что с ростом количества линейных участков даже теоретическое ускорение от векторизации падает, на практике эффективность падает гораздо сильнее, и уже при наличии 3-4 параллельных ветвей исполнения применение векторизации оказывается невыгодным.

Немного сгладить негативный эффект от безусловного слияния всех ветвей исполнения помогает проверка маски перед выполнением векторизованных линейных участков с достаточно низкой вероятностью.

Представим в графическом виде наше рассматриваемое простое условие (рис. 3).

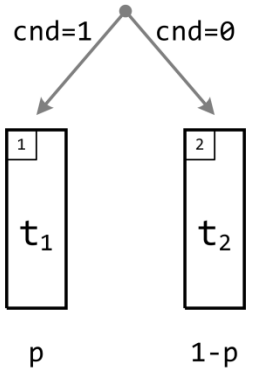


Рис. 3. Графическое представление невекторизованного условия.

В данном графе потока управления у приведенного условия есть всего две альтернативы (в зависимости от значения условия cnd). Однако при выполнении векторизация маска предикатов CND состоит из w элементов, каждый из которых может принимать значения 1 или 0. Причем если все элементы вектора CND равны 1, то выполняться должны только инструкции из линейного участка block1 (рис. 4 слева); если все элементы вектора CND равны 0, то выполняться должны только инструкции из линейного участка block2 (рис. 4 справа); если же вектор CND содержит различные значения (и 1, и 0), то необходимо выполнять инструкции из обоих линейных участков под нужными предикатами (рис. 4 по центру).

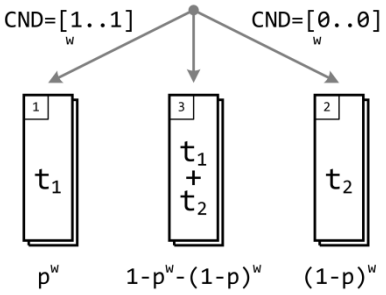


Рис. 4. Графическое представление альтернатив простого условия при выполнении векторизации.

Поэтому при выполнении слияния линейных участков векторизованные операции из соответствующего линейного участка можно выполнять при том условии, что маска предикатов этого линейного участка ненулевая.

```
if(CND != 0x0)
{
    block1 ? CND
}
if ((~CND) != 0x0)
{
    block2 ? ~CND
}
```

На первый взгляд может показаться, что такие проверки масок на пустоту не способны принести значительную пользу при векторизации, так как теоретические вероятности данных событий (p^w, $(1-p)^w$) – величины маленькие по сравнению со значениями p, $1-p$. Однако, как показывает практика, в реальных расчетных приложениях, относящихся к задачам компьютерного моделирования физических процессов, при объединении нескольких экземпляров функций в плоские циклы значения a[i] всех элементов некоторого вектора оказываются крайне близкими между собой. Поэтому часто оказывается, что и все значения предикатных векторов также близки между собой. А так как элементами предикатных векторов являются только два значения (1 или 0), то часто оказывается, что предикатный вектор это либо пустая, либо полная маска, поэтому простые проверки маски на пустоту способны существенно повысить производительность векторизуемого кода. Данный эффект подробно описан в [23]. Стоит правда заме-

тить, что данный прием работает только с расчетными кодами, касающимися компьютерного моделирования физических процессов. Для программ с дискретным контекстом (таких, как сортировка массивов чисел или задачи обработки дискретных структур данных) этот метод не приносит ничего кроме накладных расходов на дополнительные операции сравнения [27].

В рассмотренных выше случаях слияния ветвей исполнения под разными предикатами всегда предполагалось, что программный код всех линейных участков векторизуется. Однако, возможны ситуации, когда в теле плоского цикла существуют пути исполнения с экстремально низкой вероятностью, векторизация которых либо невозможна, либо не представляет никакого практического смысла (требует слишком много затрат). Однако плоские циклы с такими редкими путями исполнения также могут быть векторизованы. Одним из способов векторизации может выступать локализация ветви с низкой вероятностью. Данный метод описан в [28] на примере практической задачи. Мы же опишем этот метод в общем случае с помощью подхода, который в некоторых источниках применительно к своим предметным областям встречается под названием blackhole (черная дыра) [29].

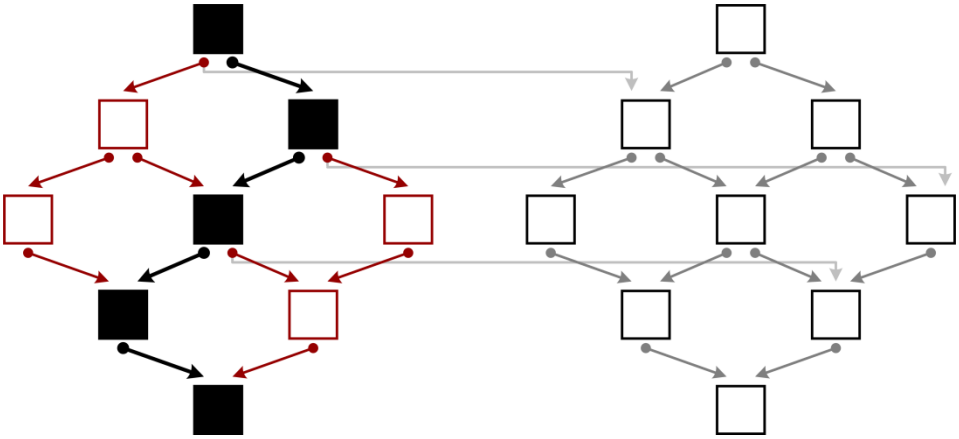


Рис. 5. Иллюстрация схемы работы оптимизации blackhole.

Пусть дам некоторый CFG тела плоского цикла (рис. 5 слева), в котором присутствует явно выделенный пусть исполнения (на рисунке выделен черным цветом), вероятность прохождения которого близка к единице. Другие крайне маловероятные линейные участки (выделенные на рисунке красным цветом) представляют собой программный контекст, который практически никогда не исполняется, и векторизацию которого проводить нецелесообразно, либо невозможно. Оптимизация blackhole заключается в создании точной копии CFG, на которую перенаправляются все маловероятные переходы из основного тела. После выполнения перенаправления маловероятных переходов все ребра и линейные участки, отмеченные на рис. 5 красным цветом, могут быть удалены. Таким образом, после выполнения оптимизации в качестве объекта векторизации остается ограниченный и пригодный к векторизации программный контекст. В случае же если один из маловероятных переходов все же осуществится, то выполнится переход на точную копию изначального CFG, который хоть и не оптимально, но корректно отработает данную редкую ситуацию. Данная оптимизация получила свое название blackhole потому что после выполнения редкого перехода на копию CFG программа не может вернуться обратно в векторизованную версию, в этом случае данный экземпляр плоского цикла закончит свое исполнение в черной дыре. Данная оптимизация может применяться в различных вариациях. Например, можно создавать копию не целого CFG, а только конкретных маловероятных регионов; в этом случае мы будем иметь оптимизацию локализации редких путей исполнения. Наоборот, более консервативным вариантом оптимизации является перенаправление маловероятных переходов сразу на голову невекторизованной копии тела цикла, что соответствует просто проведению повторного расчета при возникновении исключительной ситуации.

В данном разделе был рассмотрен набор оптимизаций, которые могут быть применены к CFG тела плоского цикла для обеспечения возможности применения к нему векторизации. Они различаются по условиям применения и характеристикам профиля исполнения.

При отсутствии профиля исполнения программы единственным вариантом применения векторизации остается безусловное слияние ветвей исполнения. Если профиль исполнения доступен, то выбор того или иного метода вектори-

зации зависит от значений вероятностей линейных участков рассматриваемого CFG. При умеренно низких вероятностях исполнения линейных участков и относительной доступности векторизации их кода целесообразно ограничиваться добавлением проверок на пустоту масок этих линейных участков. Если же в коде присутствуют линейные участки с крайне низкой вероятностью исполнения, которые к тому же трудно поддаются векторизации, то такие участки стоит подвергать локализации, выносить из циклов. В случае преобладания в цикле доминантного хорошо векторизуемого пути исполнения, вероятность которого близка к единице (в условиях наличия в теле цикла обильных участков кода с экстремально низкой вероятностью), применение оптимизации blackhole представляется наилучшим выбором.

5. Заключение

Проведение исследований в области разработки новых методов векторизации программного кода имеет важное значение для повышения эффективности выполнения суперкомпьютерных расчетов. В современных суперкомпьютерных приложениях часто встречаются фрагменты кода с сильно разветвленным графом потока исполнения, что препятствует векторизации с помощью автоматических средств. Особенности набора векторных инструкций AVX-512 позволяют выполнять векторизацию практически произвольного программного контекста. Использование подхода по описанию вычислений в виде плоских циклов позволяет еще больше повысить шансы на успешную векторизацию. Существует набор методов по векторизации программного кода с разветвленным управлением, основанных на использовании статистики исполнения отдельных линейных участков программы. В зависимости от статистики для эффективной векторизации кода могут быть выбраны различные методы с разной степенью агрессивности выполнения оптимизации. Сбор и анализ профиля исполнения программы крайне важен для принятия решения о выборе стратегии векторизации. Разработка и анализ различных подходов и методов векторизации сложного программного контекста были произведены коллективом авторов данной статьи на основе опыта проведения векторизации программного кода реальных высокопроизводительных приложений: была выполнена полная векторизация кода точного римановского решателя, что позволило ускорить его в 7 раз по сравнению со скалярной версией [30]; были разработаны векторизованные версии функций для работы с геометрическими примитивами [31] в рамках реализации метода погруженных границ численного решения задач газовой динамики [32]; были разработаны векторизованные версии функций для работы с матрицами специального вида [33] при реализации метода RANS/ILES расчета турбулентных течений [34]; были опробованы методы векторизации гнезд циклов на примере сортировки массивов чисел [35], была векторизована реализация определения конфликтов со спутными следами летательных аппаратов [28].

Работа выполнена при поддержке гранта РФФИ № 20-07-00594. При проведении исследований были использованы суперкомпьютеры МСЦ РАН: МВС-10П, МВС-10П ОП [36].

Optimizations Applied to the Control Flow Graph of a Program to Improve the Efficiency of Flat Loops Vectorization

B.M. Shabanov, A.A. Rybakov, A.D. Chopornyak

Abstract. Improving the efficiency of supercomputer calculations is an urgent problem due to the permanent complication of models, an increase in the size of computational meshes and the amount of data being processed. To meet the current needs for computing resources for solving practical and scientific problems, it is necessary to optimize high-performance computing at all levels: the distribution of computations between the nodes of the supercomputer cluster, optimization of the operation of multithreaded computational algorithms for systems with shared memory, optimization of computations within one computation thread. Vectorization of program code is a low-level optimization that allows several times to speed up the execution of hot spots by combining several instances of a program fragment for simultaneous execution on one processor core. This article describes an approach to vectorizing a program context of a special kind called a flat loop. At the same time, special attention is paid to optimizing the control flow graph with a known execution profile to reduce the costs associated with the presence of abundant control within such loops.

Keywords: vectorization, AVX-512, flat loop, program control flow graph, program execution profile, linear sections merging, unlikely execution branches localization

Литература

1. N. Stephens, S. Biles, M. Boettcher et al. The ARM scalable vector extension. IEEE Micro, 2017, 37 (2), p. 26-39.
2. M. Gschwind. Workload acceleration with the IBM POWER vector-scalar architecture. IBM Journal of Research and Development, vol. 60 (2016), No. 2/3, paper 14, p. 1-18.
3. Intel 64 and IA-32 architectures software developer's manual. Combine volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4. Intel Corporation, April 2021.
4. В.Ю. Волконский, А.В. Брегер, А.В. Грабежной и др. Методы распараллеливания программ в оптимизирующем компиляторе для ВК семейства Эльбрус. Современные информационные технологии и ИТ-образование, 2011, № 7, с. 46-59.
5. B.M. Shabanov, A.A. Rybakov, S.S. Shumilin. Vectorization of high-performance scientific calculations using AVX-512 instruction set. Lobachevskii Journal of Mathematics, vol. 40 (2019), № 5, p. 580-598.
6. J. Jeffers, J. Reinders, A. Sodani. Intel Xeon Phi processor high performance programming, Knights Landing edition. Morgan Kaufmann Publishers, 2016.
7. W. McDoniel, M. Höhnerbach, R. Canales et al. LAMMPS' PPPM long-range solver for the second generation Xeon Phi. J.M. Kunkel et al. (Eds.): ISC High Performance 2017, LNCS, vol. 10266 (2017), p. 61-78.
8. O. Krzikalla, F. Wende, M. Höhnerbach. Dynamic SIMD vector lane scheduling. M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS, vol. 9945 (2016), p. 354-365.
9. M. Bader, A. Breuer, W. Holtz, S. Rettenberger. Vectotization of an augmented Riemann solver for the shallow water equations. Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014, p. 193-201.
10. C.R. Ferreira, K.T. Mandli, M. Bader. Vectorization of Riemann solvers for the single- and multilayer shallow water equations. Proceedings of the 2018 International Conference on High Performance Computing and Simulation, HPCS 2018, p. 415-422.
11. B. Bramas. A novel hybrid quicksort algorithm vectorized using AVX-512 on Intel Skylake. International Journal of Advanced Computer Science and Applications, vol. 8 (2017), № 10, p. 337-344.
12. E. Coronado-Barrientos, M. Antonioletti, A.G. Loureiro. A new AXT format for an efficient SpMV product using AVX-512 instructions and CUDA. Advances in Engineering Software, vol. 156 (2021).
13. B. Bramas, P. Kus. Computing the sparse matrix vector product using block-based kernels without zero padding on processors with AVX-512 instructions. arXiv:1801.01134v2, 2018.
14. I. Kulikov, I. Chernykh, A. Tutukov. Hydrogen-helium chemical and nuclear galaxy collision: Hydrodynamic simulations on AVX-512 supercomputers. Journal of Computational and Applied Mathematics, 2021, 391 (1).
15. I.M. Kulikov, I.G. Chernykh, A.V. Tutukov. A new parallel Intel Xeon Phi hydrodynamics code for massively parallel supercomputers. Lobachevskii Journal of Mathematics, vol. 39 (2018), No. 9, p. 1207-1216.
16. J. Armstrong. Programming Erlang. Software for a concurrent world. The Pragmatic Programmers, 2013, 520 p.
17. G.I. Savin, B.M. Shabanov, A.A. Rybakov, S.S. Shumilin. Vectorization of flat loops of arbitrary structure using instructions AVX-512. Lobachevskii Journal of Mathematics, vol. 41 (2020), No. 12, p. 2566-2574.
18. Intel Intrincs Guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide (дата обращения 01.06.2021)
19. S. Muchnick. Advanced compiler design and implementation. Morgan Kaufmann Publishers, 1997.
20. А.А. Рыбаков. Алгоритм создания случайных графов потока управления для анализа глобальных оптимизаций в компиляторе. Parallel and Distributed Computing Systems PDSC 2013, Collection of scientific papers, p. 269-275.
21. A.V. Aho, M.S. Lam, R. Sethi, J.D. Ulman. Compilers: principles, techniques, and tools. Prentice Hall, 2nd ed, 2006.
22. О.А. Четверина. Методы коррекции профильной информации в процессе компиляции. Труды ИСП РАН, том 27 (2015), вып. 6, с. 49-63.
23. А.А. Рыбаков, А.Д. Чопорняк. Повышение производительности векторного кода с помощью

мониторинга плотности масок в векторных инструкциях. Труды НИИСИ РАН, т. 10 (2020), № 4, с. 40-47.

24. Using the GNU Compiler Collection. For GCC version 7.4.0. Richard M. Stallman and the GCC Developer Community.

25. Intel C++ Compiler Classic Developer Guide and Reference. https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top.html (дата обращения 01.06.2021)

26. В.Ю. Волконский, С.К. Окунев. Предикатное представление как основа оптимизации программы для архитектур с явно выраженной параллельностью. Информационные технологии, 2003, № 4, с. 36-45.

27. А.А. Рыбаков, П.Н. Телегин, Б.М. Шабанов. Проблемы векторизации гнезд циклов с использованием инструкций AVX-512. Электронный научный журнал: Программные продукты, системы и алгоритмы, 2018, № 3, с. 1-11.

28. А.А. Рыбаков. Оптимизация задачи об определении конфликтов с опасными зонами движения летательных аппаратов для выполнения на Intel Xeon Phi. Программные продукты и системы, т. 30 (2017), № 3, с. 524-528.

29. B. Ilbeyi. Co-optimizing hardware design and meta-tracing just-in-time compilation. A dissertation for the degree of Doctor of Philosophy, Cornell University, 2019.

30. A.A. Rybakov, S.S. Shumilin. Vectorization of the Riemann solver using the AVX-512 instruction set. Program Systems: Theory and Applications, vol. 10 (2019), № 3 (42), p. 41-58.

31. А.А. Рыбаков. Векторизация нахождения пересечения объемной и поверхностной сеток для микропроцессоров с поддержкой AVX-512. Труды НИИСИ РАН, т. 9 (2019), № 5, с. 5-14.

32. Y.-H. Tseng, J.H. Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. Journal of Computational Physics, v. 192 (2003), p. 593-623.

33. Л.А. Бендерский, А.А. Рыбаков, С.С. Шумилин. Векторизация перемножения матриц специального вида с использованием инструкций AVX-512. Современные информационные технологии и ИТ-образование, т. 14 (2018), № 3, с. 594-602.

34. L.A. Benderskii, D.A. Lyubimov, A.O. Chestnykh, B.M. Shabanov. The use of the RANS/ILES method to study the influence of coflow wind on the flow in a hot, nonisobaric, supersonic airdrome jet during its interaction with the jet blast deflector. High temperature, vol. 56 (2018), No. 2, p. 247-254.

35. А.А. Рыбаков, С.С. Шумилин. Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций. Программные системы: Теория и алгоритмы, т. 10 (2019), № 4 (43), с. 77-96.

36. JSCC RAS Supercomputing resources. http://www.jscc.ru/supercomputing-resources/ (дата обращения 01.06.2021)

Сокращение нейронной сети ResNet-32 с применением структурного прунинга и направленного дропаута

М.М. Пушкарева[1], Я.М. Карандашев[2]

Научно-исследовательский институт системных исследований РАН, Москва, Россия

[1]pushkarevamariia@yandex.ru [2]karandashev@niisi.ras.ru

Аннотация. В данной статье изучается сокращение сверточных нейронных сетей со слоями батч-нормализации и остаточными связями. Рассматривается такой подход, как структурный прунинг без дальнейшего дообучения. При этом сравниваются различные виды дропаута при обучении исходной сети для повышения устойчивости к прунингу. Также для сокращения требуемой памяти предлагается алгоритм перестроения сети, который позволяет уменьшить нейронную сеть в несколько раз. Сеть ResNet-32 для задачи классификации CIFAR-10 удалось сократить в 4.5 раза при падении точности ~10%.

Ключевые слова: нейронные сети, структурный прунинг, направленный дропаут, сокращение нейронных сетей

1. Введение

Нейронные сети решают широкий класс задач, но сейчас наиболее актуально стоит вопрос их применимости на мобильных устройствах и различных чипах, что накладывает ограничения на ресурсы, предоставляемые сети. При этом есть потребность не только хранить и использовать сеть, но также обучать локально на устройстве.

Одним из подходов для сокращения и упрощения сетей является прунинг. Прунинг – это метод, который зануляет веса сети и при обучении занулённые веса не изменяются. Неструктурный прунинг допускает, что нули тензора весов могут появляться в произвольных местах при выполнении условий отбора весов. Структурный прунинг подразумевает зануление p% по выбранной размерности тензора, например, при обрезке по выходным каналам будет полностью зануляться p% карт признаков.

В работах зачастую используют неструктурированный прунинг и в дальнейшем дообучают сеть [1, 2]. Но данный подход кажется не совсем подходящим для сокращения требуемой памяти из-за появления нерегулярной разреженности. Более выигрышным с точки зрения уменьшения нейронной сети мы считаем структурный прунинг. В работе [3] после применения структурного 70% прунинга для сети ResNet-32 [4] для задачи классификации CIFAR-10 [5] падение точности составляет 3%. В некоторых работах предлагают применять прунинг на недообученной модели [6]. В некотором смысле такой метод является техникой поиска оптимальной архитектуры сети. Однако в таком случае требуются существенные затраты на дообучение сети, поэтому мы также будем рассматривать экстремальный случай применения прунинга при отсутствии ресурсов на дообучение.

1.1. Структурный прунинг

Существует множество стратегий, с помощью которых происходит обрезка сети. В случае неструктурированного прунинга стандартным подходом является удаление весов минимальных по абсолютному значению. В работе [7] для ранжирования карт признаков предлагается энтропийный подход и метод на основе косинусного коэффициента. В дальнейшем на ранжированных картах признаков применяется жадный алгоритм, уменьшающий KL-дивергенцию между отобранными картами признаками и полным набором карт признаков. В нашей работе мы будем удалять веса (каналы/нейроны) наименьшие в смысле некоторой нормы. Такой прунинг реализован во фреймворке PyTorch [8]. При уровне прунига p% выбираются (100 – p)% максимальных по модулю значений, остальные зануляются.

Для *структурного* прунинга вычисляют нормы не отдельных весов нейронной сети, а целых подтензоров. Например, в случае обычных полносвязных слоёв, когда веса представлены в виде двумерных матриц w_{ij}, описывающих связь между i-м нейроном следующего слоя и j-м нейроном предыдущего, при структурном прунинге удаляют сразу все связи ведущие к одному нейрону, т.е. p% векторов-строк матрицы с наименьшими нормами.

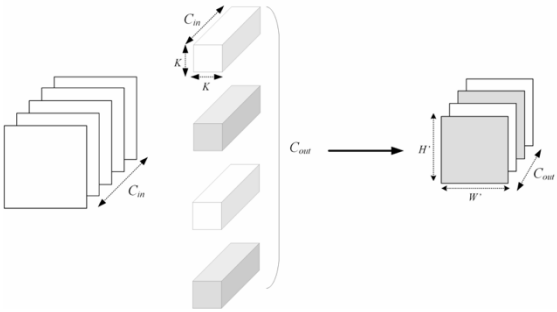


Рис. 1. Структурный прунинг выходных каналов в конволюционном слое. Серые каналы зануляются. Слева – входная картинка из C_{in} каналов; тензоры посередине означают набор из C_{out} фильтров свёртки размера $C_{in} \times K \times K$; справа выходной тензор.

В случае конволюционных нейронных сетей веса представляют собой набор трёхмерных фильтров свёртки, объединённые в один 4-мерный тензор w_{ijmk}, где первый индекс $0 \le i \le C_{out}$ означает номер фильтра, соответствующий номеру выходного слоя получаемого изображения (см. рис.1). Второй индекс $0 \le j \le C_{in}$ означает номер входного канала изображения, а индексы m и k означают позицию веса в ядре свёртки $0 \le m, k \le K$, где K – размер ядра свёртки (обычно рассматривается свёртка размера 3 x 3, т.е. K=3).

В случае таких четырёхмерных тензоров настраиваемых параметров весов, структурированный прунинг может быть построен по-разному. Из тензора w_{ijmk} можно удалять целиком отдельные одиночные вектора, двумерные матрицы или трёхмерные подтензоры. Мы рассматриваем именно последний вариант. В данной работе будем рассматривать структурный прунинг на выходных каналах (рис. 1), то есть среди C_{out} фильтров будут зануляться p% фильтров с наименьшей нормой, уменьшая тем самым число выходных каналов:

$$\text{argmin}_p \sum_{j,m,k} w_{ijmk}^2 \qquad (1)$$

1.2. Двумерный дропаут

Для устойчивости сети к прунингу мы будем рассматривать прунинг на сетях, обученных с использованием дропаута. Так как мы изучаем структурный прунинг, сети будут обучаться с двумерным дропаутом на выходах (рис. 2). Структура двумерного дропаута аналогична структурному прунингу, т.е. часть выходных каналов просто зануляются. Отличие от прунинга в том, что дропаут включается во время обучения: выходной тензор с предыдущего слоя поэлементно умножается на маску, в которой случайно занулены α% подтензоров. Таким образом, после применения двумерного дропаута будет занулено α% каналов. На следующем батче маска дропаута задается заново.

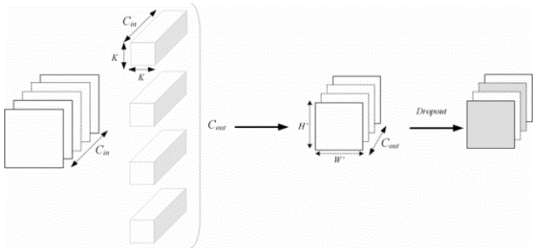


Рис. 2. Двумерный дропаут на выходных каналах (серые каналы занулены).

1.3. Направленный дропаут

Также будем использовать *направленный (или таргетированный) дропаут* [3]. Особенность данного вида дропаута в том, что он ещё ближе к прунингу, поскольку в процессе обучения зануляет блоки, выбирая их не полностью случайно, а ориентируясь на их норму. А именно выбирается γ каналов, которые соответствуют наименьшим в смысле некоторой нормы весов (кандидаты на отсев выделены серым на рис. 3), и с вероятностью α эти каналы зануляются.

На каждом проходе выборка для дропаута выбирается заново. Таким случайным выбором из наименьших по норме имитируется работа прунинга, но при этом сохраняется случайность.

В зависимость от величин параметров γ и α, как частные случаи мы можем получить модель без дропаута (при α=0) и модель с обычным дропаутом (при γ=1, α≠0). Направленность дропаута возникает, когда 0<γ<1 и α≠0.

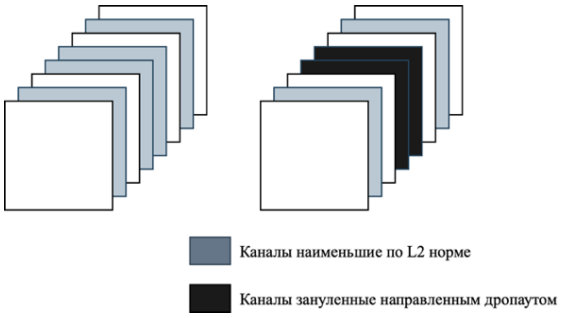


Рис. 3. Маска для направленного дропаута на выходных каналах тензора весов. Серым выделены γ наименьших по норме весов, эти каналы с вероятностью α зануляются (выделены черным).

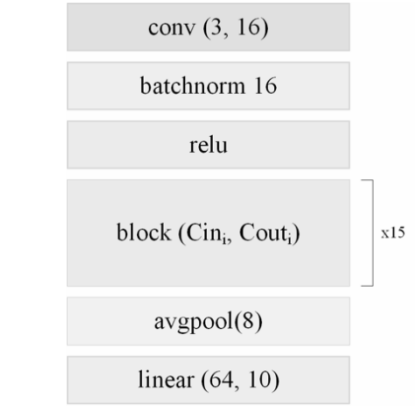


Рис. 4. Схема ResNet-32

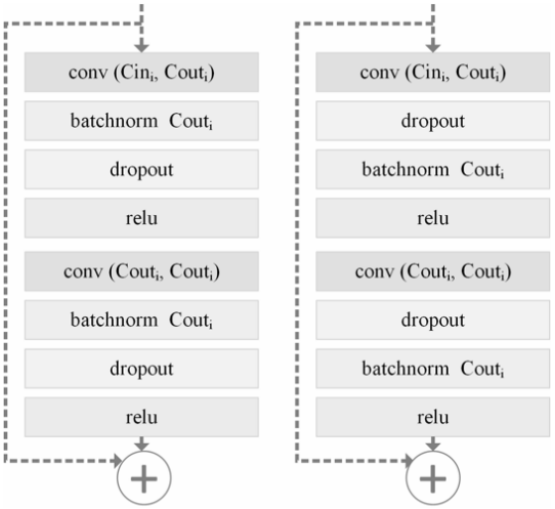


Рис. 5. ResNet-блоки, с различным расположением дропаут слоев.

2. Алгоритм

В нашей работе мы рассматриваем задачу классификации CIFAR-10 нейронной сетью с архитектурой ResNet-32. Для экспериментов используется фреймворк PyTorch [9].

Так как главной задачей стоит сокращение требуемой памяти и ускорение инференса модели, мы предлагаем переформатировать сеть с учетом удаленных каналов, при этом стоит рассмотреть отдельные случаи с учетом особенностей архитектуры сети ResNet-32, которые мы опишем ниже.

2.1. Описание ResNet-32 сети

На рисунке 4 изображена структура сети ResNet-32, состоящая из следующих элементов:

conv(a, b) - конволюционный слой с ядром 3 x 3, нулевым отступом, единичным шагом, количество входных каналов – a , количество выходных каналов – b;

batchnorm – слой батч-нормализации с указанным количеством каналов, особенность данного модуля мы подробнее рассмотрим в следующем пункте;

relu – функция ReLU;

avgpool – усредняющий пулинг;

linear(a, b) – полносвязный слой, где a – количество входных признаков, b – количество выходных признаков;

block - ResNet-блок, две вариации которого подробнее изображены на рис. 5. Два варианта различаются расположением дропаут слоев (dropout). У конволюционных слоев указано количество входных каналов и выходных каналов, количество входных каналов во второй конволюции блока равно количеству выходных каналов с предыдущего слоя. Пунктиром показан неизменный сетью сигнал с предыдущего блока.

2.2. Прунинг при батч-нормализации

В каждом ResNet-блоке после каждого конволюционного слоя присутствует слой батч-нормализации. Данный слой работает в двух режимах. В режиме обучения используются статистики по батчу (матожидание и дисперсия) и накапливаются статистики по пропущенным через сеть батчам, также обучаются параметры аффинного преобразования θ и β. В режиме eval параметры θ и β не обучаются и используются накопленные статистики по всей выборке.

На рисунке 6 показана работа данного слоя. Если в слой батч-нормализации входит тензор с зануленными каналами, то после преобразования тензора данным слоем на выходе получаем ненулевой тензор. Если мы удаляем из тензора, выходящего из слоя батч-нормализации, каналы, соответствующие нулевым каналам входного тензора, то мы теряем некоторую информацию и из-за таких слоев сеть с применением структурного прунинга не будет эквивалентна перестроенной сети. Мы предполагаем, что расположение дропаута до слоя батч-нормализации должно благоприятно влиять на применение дальнейшего прунинга (рис. 5), так как батч-норм слой настраивается на то, что каналы, соответствующие наименьшим по норме весам, будут занулены. При этом расположение дропаута после батч-норма должно сделать сеть более устойчивой к удалению каналов, так как последующая конволюция будет настраиваться на то, что определенные каналы будут занулены.

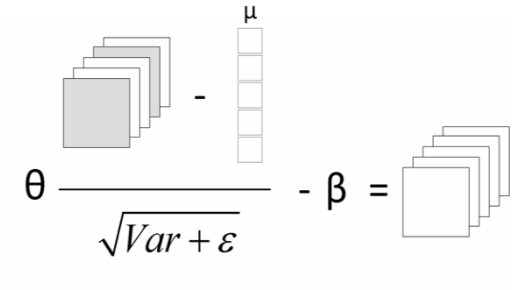


Рис. 6. Работа слоя батч-нормализации в режиме тестирования. μ - накопленное мат. ожидание, Var - накопленная дисперсия, ε - некоторое фиксированное число (по умолчанию 10e-5), θ и β - настраиваемые параметры аффинного преобразования. Серым выделены зануленные каналы входного тензора.

2.3. Две конволюции в одном блоке

Как видно из структуры ResNet-блока (рис. 5), в каждом блоке присутствует по две идущих подряд конволюции. Это значит, что при зануленении выходных каналов первой конволюции, на вход следующей конволюции подается тензор с частью уже зануленных каналов, таким образом сокращение во втором конволюционном слое составляет $p(1+p)$ часть весов (рис. 7), где p - уровень прунинга или уровень дропаута. Т.е. второй слой конволюции в блоке сокращается сильнее первого.

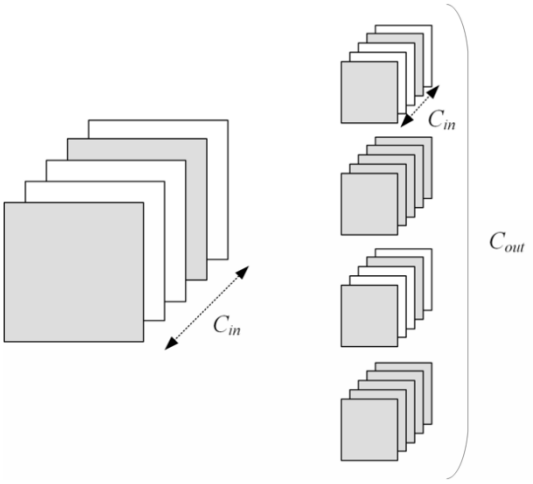


Рис. 7. Тензор, который приходит во вторую конволюцию внутри ResNet-блока, имеет зануленные входные каналы, прунинг во второй конволюции дополнительно зануляет выходные каналы второго конволюционного слоя.

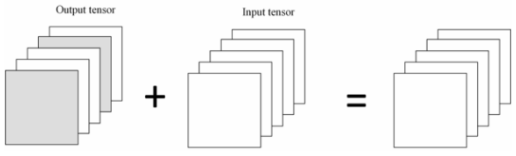


Рис. 8. Тензор, выходящий из последнего слоя ResNet-блока с зануленными каналами (Output tensor) после применения структурного прунинга, складывается с тензором, входящим в текущий блок (Input tensor). Тензор, выходящий из ResNet-блока, ненулевой.

2.4. Прунинг с остаточными связями

При удалении каналов также возникает проблема из-за остаточных связей (skip-connections): сигнал, пришедший в блок, имеет определенное количество каналов и их удаление приведет к существенной потере информации по сравнению с сетью после прунинга без перестроения (рис. 8). Поэтому мы запоминаем индексы неудаленных каналов и выполняем сложение с исходным тензором по данным индексам.

Наличие остаточных связей не позволяет сократить количество каналов в тензоре, входящих в следующий ResNet-блок.

2.5. Алгоритм сокращения сети

Для исследования сокращения требуемой памяти и для ускорения инференса модели мы предлагаем следующий порядок действий (рис. 9). Исходную модель обучаем либо без применения дропаута, либо с применением двумерного дропаута, либо с применением двумерного таргетированного дропаута. Далее применяем структурный прунинг и перестраиваем модель, удаляя зануленные каналы из сети с учетом особенностей, описанных в предыдущих пунктах. Сеть после прунинга и перестроения не дообучаем.

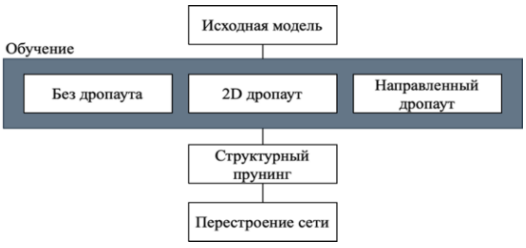


Рис. 9. Схема, описывающая три варианта с обучением без дропаута, с двумерным и таргетированным дропаутом.

3. Результаты

Мы сравнили модели, обученные с различными параметрами дропаута. γ - доля кандидатов на отсев, α - вероятность отсева среди кандидатов. При этом, если α=0, то данная модель была обучена без применения дропаута, если γ=1, α≠0, то это соответствует обычному двумерному дропауту. Если же γ<1 и α≠0, то данная модель обучена с таргетированным дропаутом.

В таблице 1 представлены точность (accuracy) моделей с уровнем прунинга 25% с различным расположением дропаута, при этом количество параметров в сети после перестроения составит 303050 из 465290 исходных, при этом без перестроения количество параметров равно количеству параметров исходной модели, то есть прунинг не меняет количество параметров сети.

При таком небольшом уровне прунинга падение точности составляет меньше 7% для всех случаев и более важную роль играет то, что модели с высоким уровнем дропаута, где отсев составляет около 67% (α=0.75, γ=0.9 и α=0.9, γ=0.75), имеют точность на 6-15% меньше, чем у других моделей, поэтому они проигрывают в точности после перестроения. При таком уровне прунинга нет существенных различий между различным расположением дропаута относительно перестроения, но модель с дропаутом перед слоем батч-нормализации обучилась лучше, чем модель с дропаутом после батч-нормализации. Это можно объяснить тем, что во втором случае мы теряем информацию, накопленную батч-норм слоем, также при зануления каналов дропаутом обучаемые параметры батч-норма обучаются реже, чем в первом случае.

Сравним модели в таблице 2, для которых применяется 50% прунинг, здесь уже заметно преимущество моделей, обученных с направленным дропаутом. Падение точности модели, обученной без дропаута, составляет 40%, с двумерным дропаутом ~35%, падение на 2-3% наблюдается у модели с таргетированным дропаутом, где уровень дропаута составляет примерно 0.25. При этом уже заметна большая устойчивость к прунингу и перестроению модели с дропаутом после слоя батч-нормализации, где уровень таргетированного дропаута около 67%.

В таблице 3 представлены точности моделей, к которым применялся прунинг на уровне 67%, что соответствует наибольшему уровню таргетированного дропаута. Модель с таким уровнем таргетированного дропаута оказалась наиболее устойчивой, точность не изменилась

Таблица 1. Точность моделей с уровнем прунинга p=0.25 для различного расположения дропаута для исходной модели, модели после применения прунинга и после перестроения сети.

p = 0.25		Accuracy BN →Drop			Accuracy Drop → BN		
α	γ	Initial	Pruned	Resized	Initial	Pruned	Resized
0.25	0.5	86.76	86.54	86.44	87.08	87.16	86.76
0.25	1	89.02	82.03	83.66	89.11	83.56	83.1
0.5	0.5	85.73	85.81	85.8	85.87	85.79	85.52
0.75	0.9	74.98	75.03	74.99	78.18	78.21	75.96
0.9	0.75	79.82	79.82	79.82	83.41	83.43	81.61
0	1	89.15	77.54	82.53	89.15	77.54	82.53

Таблица 2. Точность моделей с уровнем прунинга p=0.5 для различного расположения дропаута для исходной модели, модели после применения прунинга и после перестроения сети.

p = 0.5		Accuracy BN →Drop			Accuracy Drop → BN		
α	γ	Initial	Pruned	Resized	Initial	Pruned	Resized
0.25	0.5	86.76	84.7	84.5	87.08	87.68	86.13
0.25	1	89.02	50.15	55.94	89.11	63.56	63.51
0.5	0.5	85.73	85.67	85.47	85.87	85.7	84.25
0.75	0.9	74.98	74.98	74.98	78.18	78.27	72.83
0.9	0.75	79.82	79.82	79.82	83.41	83.39	81.11
0	1	89.15	41.73	41.19	89.15	41.73	41.19

Таблица 3. Точность моделей с уровнем прунинга p=0.67 для различного расположения дропаута для исходной модели, модели после применения прунинга и после перестроения сети.

p = 0.67		Accuracy BN →Drop			Accuracy Drop → BN		
α	γ	Initial	Pruned	Resized	Initial	Pruned	Resized
0.25	0.5	86.76	48.74	49.81	87.08	43.56	44.86
0.25	1	89.02	28.56	31.12	89.11	38.34	32.45
0.5	0.5	85.73	37.89	45.5	85.87	37.71	39.03
0.75	0.9	74.98	74.76	74.81	78.18	78.21	71.09
0.9	0.75	79.82	79.85	79.83	83.41	83.41	77.34
0	1	89.15	18.34	17.64	89.15	18.34	17.64

для модели с дропаутом после батч-норм слоя и изменилась на ~5% для модели с дропаутом перед слоем батч-нормализации. При этом в этих моделях только 100573 параметров из 465290 исходных, таким образом сеть сократилась в 4.6 раз.

Таким образом, при обучении модели с высоким уровнем таргетированного дропаута мы получаем сеть, которая более устойчива к структурному прунингу с высоким уровнем отсева, а для устойчивости к дальнейшему перестроению слои дропаута в исходной модели должны располагаться после слоев батч-нормализации. Алгоритм перестроения сети, описанный в данной работе, можно перенести на сети с конволюционными и полносвязными слоями, со слоями батч-нормализации и резидуал-связями, однако,

для других сетей требуется дальнейшее изучение и более подробное рассмотрение особенностей архитектур.

Работа выполнена при поддержке гранта РФФИ No 19-29-03030 «Поиск физических, технологических и схемотехнических нейросетевых решений на основе мемристорных кроссбаров».

Reduction of Neural Network ResNet-32 by Structural Pruning and Targeted Dropout

M.M. Pushkareva, I.M. Karandashev

Abstract. This article examines the reduction of convolutional neural networks with batch normalization layers and residual connections. Structural pruning without further training is considered. We compared different types of dropouts while training the original network to increase resistance to pruning. Also, it is proposed a network rebuilding algorithm, which reduces the required memory and compresses neural networks several times. The ResNet-32 network for the CIFAR-10 classification problem was reduced 4.5 times with ~ 10% drop of accuracy.

Keywords: neural networks, structural pruning, targeted dropout, neural networks compression

Литература

1. Wang E., Davis J., Zhao R., Ng H., Niu X., Luk W., Cheung P., Constantinides G., Deep neural network approximation for custom hardware: Where We've Been, Where We're going, ACM Computing Surveys, 2019, 52(2).
2. Han S., Mao H., Dally W., Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, 2015, http://arxiv.org/abs/1510.00149
3. Gomez A., Zhang I., Kamalakara S., MadaanD., Swerky K., Gal Y., Hinton G., Learning Sparse Networks Using Targeted Dropout, 2019, http://arxiv.org/abs/1905.13678
4. He K., Zhang X., Ren S., Sun J., Deep Residual Learning for Image Recognition, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, DOI: 10.1109/CVPR.2016.90.
5. Dataset CIFAR-10 and CIFAR-100 https://www.cs.toronto.edu/~kriz/cifar.html
6. Van Amersfoort J., Alizadeh M., FarquharS., LaneN., Gal Y., Single Shot Structured Pruning Before Training, 2020, http://arxiv.org/abs/2007.00389
7. Singh A., Rajan P., Bhavsar A., Deep Hidden Analisys: a statistical Framework to Prune Feature Maps, ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing, 2019, DOI: 10.1109/ICASSP.2019.8682796.
8. Pytorch pruning: https://arxiv.org/pdf/2004.13770.pdf
9. PyTorch Framework https://pytorch.org

Распознавание текста на зашумленных изображениях

А.А. Рыбаков[1], С.А. Фрейлехман[2]

[1]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, rybakov@jscc.ru;

[2] МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, freysa@jscc.ru, +7 926 125-66-20

Аннотация. Автоматическая сегментация изображений является востребованной задачей. В современном мире эта задача решается с использованием обученных искусственных нейронных сетей. Для обучения нейросети необходимо подготовить обучающий набор данных. Подобный набор данных должен состоять из изображений для обучения и «правильных ответов» к ним. Формирование набора данных является трудоемкой и дорогостоящей задачей. Если учитывать необходимое количество этих пар (сотни или тысячи штук), то задача может стать трудно выполнимой за разумное время. Для решения этой задачи было предложено разработать генератор обучающей выборки.

Ключевые слова: генератор текста, искусственная нейронная сеть, машинный текст, машинное обучение

1. Введение

Искусственные нейронные сети (ИНС) структурируют и анализируют большие объемы данных. Все чаще применяется машинное обучение для предсказания событий и обнаружения неочевидных закономерностей. Все это помогает создавать и улучшать продукты в науке, промышленности и бизнесе [1], [2].

Одной из популярных задач в области классификации объектов с использованием ИНС является задача распознавания текста.

Сложность обучения ИНС для решения этой задачи, как и во многих других задачах, сводится к подготовке качественного набора обучающих данных. Данное суждение основано на том, что насколько бы ни была сложной или наоборот, простой архитектура модели ИНС, она не сможет решить поставленную перед ней задачу, если она не обладает качественной тренировочной выборкой, на которой она сможет учиться [3].

В данной статье приводится анализ классических моделей ИНС для решения задачи распознавания текста с использованием генератора обучающей выборки.

2. Постановка задачи исследования

Автоматическая сегментация изображений (*image segmentation*) – востребованное направление использования нейронных сетей. Например, чтобы вручную разметить спутниковые снимки, обрисовав контуры домов, дорог, парковых зон или выделить текст на фоне некоего изображения, нужно потратить колоссальное количество времени.

Подобные задачи ныне решаются с использованием ИНС. Для этого ИНС на вход подается обучающая выборка, состоящая из исходного исследуемого изображения и сегментированного варианта исходного изображения, преобразованного в маску. Результатом обучения станет модель ИНС с подобранными весами, способная решать поставленную перед ней задачу с определенной долей погрешности, от которой собственно и зависит качество самой модели [4].

В данной статье исследуется задача бинарной сегментации изображения с черным машинным текстом на случайном фоне. Размер текста – крупный, для того, чтобы при сжатии изображения до входного тензора ИНС сохранились основные признаки символов текста. Результатом работы модели ИНС ожидается классификация пикселей исходного изображения на два класса - текст и фон (не текст).

Метод исследования – попиксельная сегментация. В данном методе каждому пикселю исходного изображения модель ИНС будет присваивать один из двух классов. Реализация данного метода заключается в формировании обученной моделью маски изображения со значениями пикселей 1 и 0, где: 1 – текст, 0 – фон (не текст).

Цель исследования: анализ методов получения эффективного набора данных для обучения модели ИНС для задачи бинарной сегментации изображения с текстом.

Задачи исследования:

- получить некий первичный набор данных для исследований с использованием генератора данных;
- оценить характерные свойства полученной

выборки;

- задать параметры обучения модели;

- экспериментальным путем отобрать подходящую модель ИНС для решения данной задачи;

- применительно к выбранной модели провести анализ влияния параметров обучающей выборки на качество предсказаний модели.

2.1. Основные параметры обучающего набора данных

В рамках данного исследования было принято решение исследовать изображения, содержащие *машинный текст на русском языке* (кириллица). Также модель необходимо научить распознавать символы и знаки, часто встречающиеся в русских текстах, для полноты передачи исходного текста после обработки изображения моделью ИНС.

Полностью русскоязычный алфавит с дополнительными символами для генератора обучающих данных представлен ниже:

```
alphabet_RU = 'абвгдеёжзийклмнопр-
стуфхцчшщъыьэюяАБВГДЕЁЖЗИКЛМНОПРСТУФ-
ХЦЧШЩЪЫЬЭЮЯ'
symbol_RU = '!@#$%*+/=()[]{}~‘’”№<>,
.;:?'
```

Для генерируемой обучающей выборки было принято решение в качестве *цвета текста* использовать случайные *оттенки черного*.

Для выборки *фон* генерируется из *случайных фигур случайного цвета*.

В рамках данной задачи подразумевается, что изображение, содержащее текст, не обязательно должно иметь идеально горизонтальное расположение строк. Потому введена возможность *случайного поворота изображения с текстом* в пределах заданных углов (± 5º).

В данной задаче хоть и исследуется только *машинный текст*, но и он бывает представлен в виде *различных стилей и размеров шрифтов*. Для определения этого параметра исходных обучающих данных для модели ИНС, были отобраны наиболее часто встречающиеся шрифты [5], [6].

Ниже представлен список шрифтов, которые применялись для генерации текста на обучающей выборке:

```
often_fonts = ['arial.ttf',
'ARIALN.TTF', 'BOOKOS.TTF',
'cour.ttf', 'GOTHIC.TTF',
'georgia.ttf', 'lucon.ttf',
'micross.ttf', 'verdana.ttf',
'ariblk.ttf', 'BKANT.TTF',
'comic.ttf', 'GARA.TTF', 'impact.ttf',
'l_10646.ttf', 'times.ttf',
'lucon.ttf']
```

Пример сгенерированных дынных, с учетом выше предложенных переменных, представлен на рисунке 1.

Рис. 1. Пример работы генератора

2.2. Свойства генератора обучающей выборки

Чтобы оценить качество обучающей выборки, состоящей из набора параметров, определенных случайным образом, введем понятие совокупности общих свойств, которые будут оказывать прямое влияние на формирование весов модели – формирование признаков, по которым модель будет распознавать текст.

Представленные выше условия формирования генератора и сам генератор, формируют набор данных со следующими свойствами:

- случайный фон из всей палитры цветов (имеет однозначные границы перехода цветов – нет плавного перехода цветов);

- цвет текста – оттенки черного в диапазоне от 0 до 30 по каждому каналу (формат цвета RGB);

- текст разного размера;

- текст имеет разные шрифты;

- тест имеет случайный угол наклона;

- из-за того, что картинки генерируются разных размеров, а на вход в модель подаются меньшего размера, то появляются шумы, которые также становятся качествами – свойствами данной выборки.

Отсюда можно предположить какую совокупность признаков сможет сгенерировать модель для распознания текста на изображении:

- геометрические очертания букв и символов теста, отличные от фона;

- резкий перепад градиент цвета пикселей на границе текста и фона;

- цвет фона и текста - различаются.

3. Параметры модели сегментации изображения

3.1. Гиперпараметры для обучения модели ИНС

Перед обучением ИНС необходимо настроить гиперпараметры – параметры, которые зада-

ются перед обучением и не являются результатом вычислений ИНС. В данной статье к таким параметрам отнесены: размер сгенерированного датафрейма, его тренировочная и обучающая части, размер пакета данных, подаваемого за раз на обучение, количество шагов в одной эпохе, количество эпох обучения, количество шагов валидации и значение шагов через которое необходимо проводить валидацию результатов обучения. Значения вышеописанных параметров:

```
dataframe = 1000
train_dataframe = 700
val_dataframe = 300
batch_size = 20
steps_per_epoch = 100
epochs = 25
validation_steps = 50
validation_freq = 1
```

Прочие не указанные гиперпараметры, например, такие как скорость обучения и параметры ее изменения во время обучения, взяты по умолчанию из библиотеки Keras [7].

3.2. Размерность входного и выходного тензоров модели

Тензор входного слоя для всех моделей – (batch_size, 256, 256, 3) или (*None, 256, 256, 3*) на момент компиляции модели;

Размер тензора выходного слоя моделей при решении задачи сегментации изображения (классификации пикселя) – (*None, 256, 256, 1*) на момент компиляции модели.

3.3. Настройки модели

Функция активации нейронов – *ReLu*. ReLu (рис. 2) менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции.

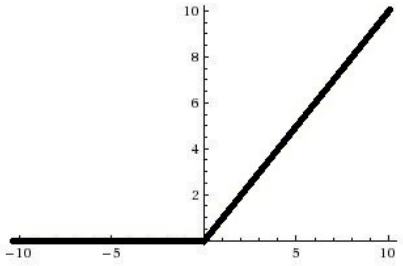


Рис. 2. Функция активации ReLu

Данная функция используется для активации скрытых слоев в каждой исследуемой модели.

В качестве выходной функции активации используется *сигмоида* (рис. 3) – логистическая функция активации, результатом работы которой является вероятность принадлежности к анализируемому классу. Ее диапазон значений от 0 до 1.

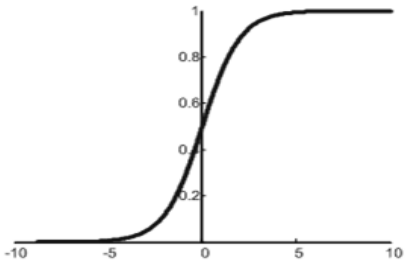


Рис. 3. Функция активации выходного слоя

Данная функция используется для активации выходных нейронов в каждой исследуемой модели.

Функция оптимизации модели для достижения глобальных минимумов во время обучения модели – *Adam*, так как он выдаёт наилучшие результаты при минимальном подборе параметров [7]:

```
Adam(lr=0.0001, beta_1=0.9,
     beta_2=0.999, epsilon=1e-08,
     decay=0.0)
```

Обучающая функция потерь (*loss*) – двоичная кросс-энтропия (binary_crossentropy), как наиболее подходящая функция потерь при тренировке двоичного классификатора [8].

3.4. Исследуемые модели ИНС

Сверточные нейронные сети (convolutional neural networks, CNN) и *глубинные сверточные нейронные сети* (deep convolutional neural networks, DCNN) сильно отличаются от других видов сетей. Обычно они используются для обработки изображений, реже для аудио. Типичным способом применения CNN является классификация изображений. Для формирования данной архитектуры за основу использовали некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определенного набора простых клеток. Таким образом, идея сверточных нейронных сетей заключается в чередовании сверточных слоев (англ. convolution layers) и субдискретизирующих слоев (англ. subsampling layers или англ. pooling layers, слоёв подвыборки). Структура сети – однонаправленная (без обратных связей), принципиально многослойная [9].

SegNet – это архитектура глубокого кодировщика-декодера для мультиклассовой пиксельной сегментации, также эффективная для понимания сцен внутри помещений, исследованная и разработанная членами группы компьютерного зрения и робототехники Кембриджского университета, Великобритания.

Архитектура состоит из последовательности уровней нелинейной обработки (кодировщиков) и соответствующего набора декодеров, за которыми следует пиксельный классификатор. Обычно каждый кодер состоит из одного или нескольких сверточных слоев с пакетной нормализацией и нелинейностью ReLU, за которыми следуют неперекрывающиеся maxpooling и субдискретизация. Разреженное кодирование из-за процесса объединения повышается в декодере с использованием индексов maxpooling в последовательности кодирования (рис. 4). Одним из ключевых компонентов SegNet является использование индексов максимального объединения в декодерах для выполнения повышающей дискретизации карт функций с низким разрешением.

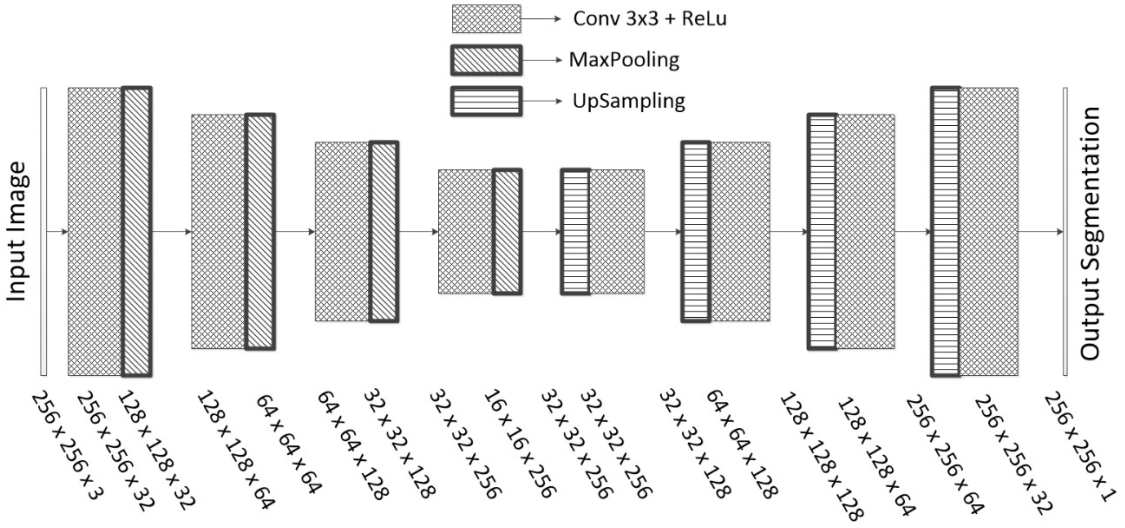


Рис. 4. Архитектура модели SegNet

Это имеет важные преимущества, заключающиеся в сохранении высокочастотных деталей в сегментированных изображениях, а также в сокращении общего количества обучаемых параметров в декодерах [10].

U-Net считается одной из стандартных архитектур CNN для задач сегментации изображений, когда нужно не только определить класс изображения целиком, но и сегментировать его области по классу, т. е. создать маску, которая будет разделять изображение на несколько классов.

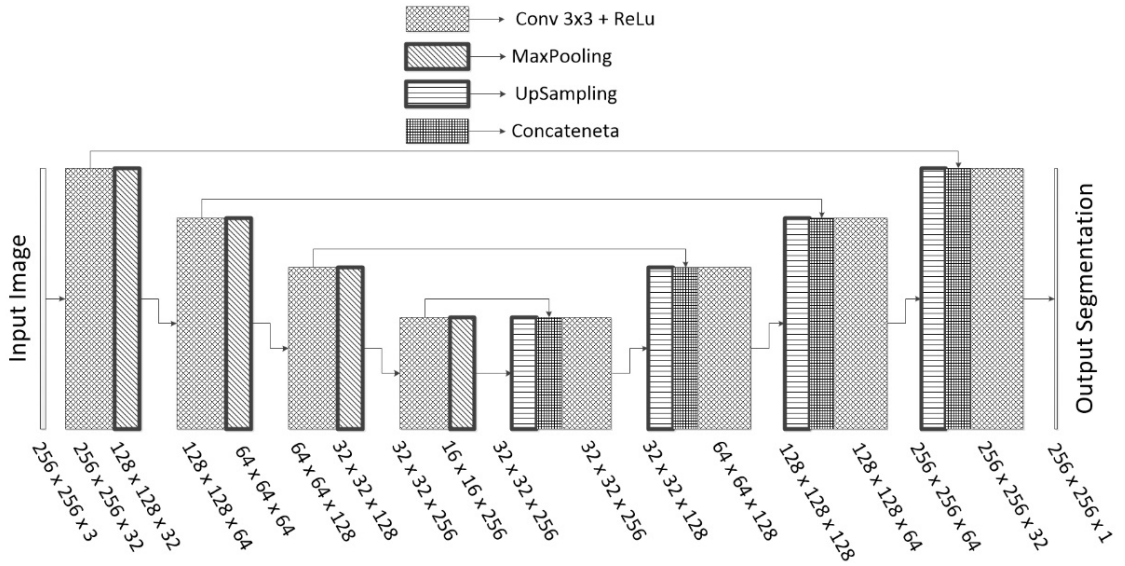


Рис. 5. Архитектура модели U-Net

Архитектура состоит из стягивающего пути

для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию.

Для U-Net характерно:

- достижение высоких результатов в различных реальных задачах, особенно для биомедицинских приложений;
- использование небольшого количества данных для достижения хороших результатов.

Архитектура ИНС (рис. 5) состоит из сужающегося пути (слева) и расширяющегося пути (справа). Сужающийся путь – типичная архитектуре сверточной нейронной сети. Он состоит из повторного применения двух сверток 3×3, за которыми следуют ReLU и операция максимального объединения (2×2 степени 2) для понижения разрешения.

На каждом этапе понижающей дискретизации каналы свойств удваиваются. Каждый шаг в расширяющемся пути состоит из операции повышающей дискретизации карты свойств, за которой следуют:

- свертка 2×2, которая уменьшает количество каналов свойств;
- объединение с соответствующим образом обрезанной картой свойств из стягивающегося пути;
- две 3×3 свертки, за которыми следует ReLU.

Обрезка необходима из-за потери граничных пикселей при каждой свертке.

На последнем слое используется свертка 1×1 для сопоставления каждого 64-компонентного вектора свойств с желаемым количеством классов. Всего сеть содержит 23 сверточных слоя [11].

4. Результат обучения ИНС

Модели SegNet и U-Net были обучены и протестированы на наборе данных, полученном с использование генератора. Все модели тренировались на одной и той же сгенерированной выборке данных.

Исходные гиперпараметры ИНС перед обучением – идентичные.

Обучение моделей производилось в облачной среде для разработки и обучения ИНС – Google Colab [12]. Обучение производилось с использованием графической карты NVIDIA Tesla P100.

Результаты обучения ИНС представлены на рисунках 6-7.

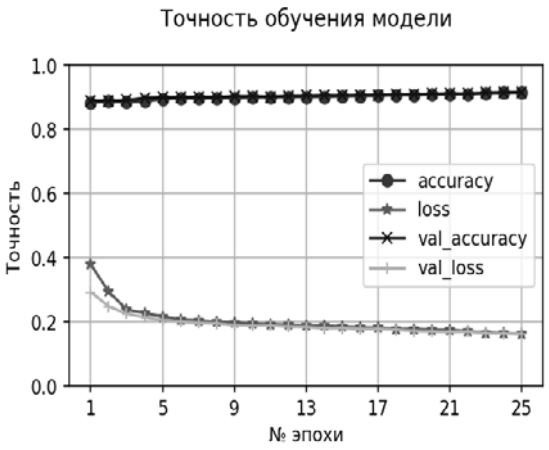


Рис. 6. Точность обучения модели SegNet

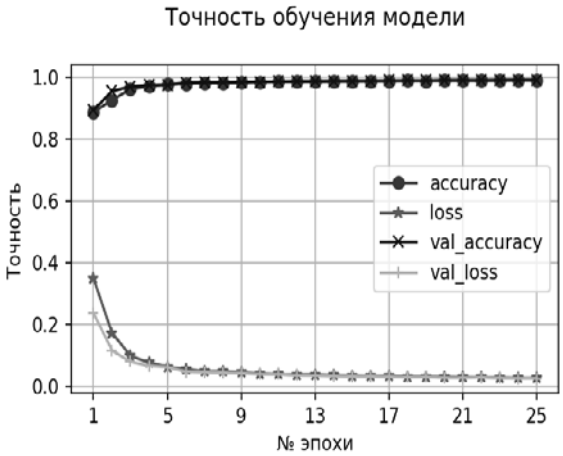


Рис. 7. Точность обучения модели U-Net

На рис. 6 и рис. 7 представлены следующие величины:

loss – ошибка обучения;

accuracy – точность обучения;

val_loss – ошибка валидации;

val_accuracy – точность валидации.

Сравнительный анализ результатов исследований приведен в таблицах 1-2.

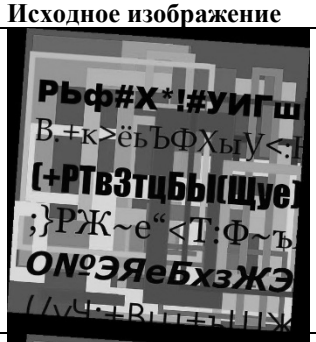
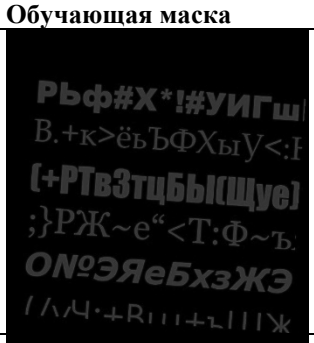
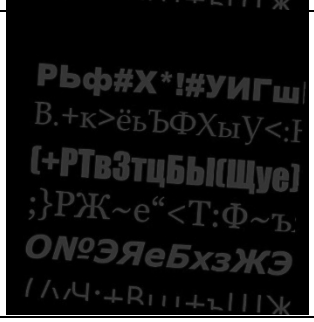
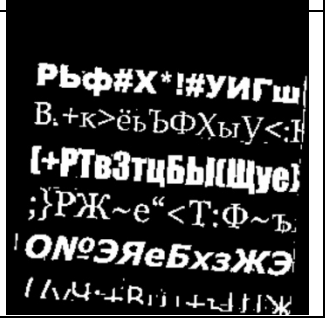
Таблица 1. Время обучения моделей ИНС

ИНС	**Время**
SegNet	1 ч. 58 мин. 46 сек.
U-Net	2 ч. 12 мин. 38 сек.

Таблица 2. Результаты обучения моделей ИНС

ИНС	**loss**	**acc.**	**val_loss**	**val_acc.**
SegNet	0,1627	0,9158	0,1628	0,9170
U-Net	0,0255	0,9899	0,02468	0,990

Таблица 3. Предсказания моделей ИНС

ИНС	Исходное изображение	Обучающая маска	Предсказание модели
SegNet			
U-Net			

5. Заключение

В таблице сравнительного анализа ИНС однозначно видно, что в ограниченных условиях обучения, архитектура модели ИНС U-Net продемонстрировала свои качества быстрого и высокоточного обучения, что было подтверждено в ходе данного эксперимента.

Созданный в рамках эксперимента генератор обучающей выборки продемонстрировал умеренные качества генерации обучающей выборки для задачи обучения ИНС сегментации изображения с черным машинным текстом на случайном фоне.

Следующими этапами разработки генератора обучающей выборки будут:

- использование произвольных цветовых схем для текста и сгенерированного фона;
- усложнение фона – добавление внутреннего генератора шумов;
- добавление возможности формирования фона с использованием естественных изображений (изображения неба, воды, ландшафтов и др.);
- добавление возможности использования различных настроек освещения изображения и размытия текста.

Также, необходимо разработать «фильтр», через который данные будут подаваться в модель ИНС. Такое решение позволит подготавливать большие изображения с малым размером шрифта текста к последующей сегментации. Это необходимо для того, чтобы не ухудшалось качество исходного изображения при его сжатии до формы входного тензора модели ИНС.

Работа выполнена в МСЦ РАН в рамках государственного задания по теме 0580-2021-0016.

Text Recognition on Noisy Images

A.A. Rybakov, S.A. Freylekhman

Abstract. Automatic image segmentation is a popular task. In the modern world, this problem is solved using trained artificial neural nets. To train a neural net, it is necessary to prepare a training data set. Such a data set should consist of images for training and "correct answers" to them. Generating a data set is a time-consuming and expensive task. If you take into account the required number of these image pairs (hundreds or thousands of them), then the task may become difficult (or not feasible). To solve this problem, it was proposed to develop a training sample generator.

Keywords: text generator, artificial neural net, machine text, machine learning

Литература

1. С.Н. Богославский. Область применения искусственных нейронных сетей и перспективы их развития. «Научный журнал КубГАУ», Т. 3 (2007), № 27, 1–11.

2. А.М. Самарин. История нейрокомпьютинга и его применение в бизнесе. «Вестник Сибирского института бизнеса и информационных технологий», Т. 6 (2013), № 2, 48-54.

3. Д.В. Васенков. Методы обучения искусственных нейронных сетей. «Компьютерные инструменты в образовании», Т. 1 (2007), № 1, 20–29.

4. С.В. Белим, С.Б. Ларионов. Алгоритм сегментации изображения с помощью искусственной нейронной сети без использования других изображений. «Радиостроение», Т. 3 (2018), № 1, 43-53.

5. Habr.com. Шрифты, общие для всех (актуальных) версий Windows, и их Mac-эквиваленты. 2009. URL: https://habr.com/ru/post/68189/ (дата обращения: 30.04.2021).

6. Redhamster. Топ-7 шрифтов, используемых профессионалами в дизайне. 2019. URL: https://redhamster.bz/top-7-shriftov-ispolzuemyh-professionalami-v-dizajne/ (дата обращения: 30.04.2021).

7. Русскоязычная документация Keras. Оптимизаторы. URL: https://ru-keras.com/optimizer/ (дата обращения: 30.04.2021).

8. Ф.М. Бетелин, А.Ф. Галимянов. Искусственные нейронные сети и их приложения. Казань, Казан. ун-та, 2018.

9. X. Zhang, J. Zhao, Y. LeCun. Character-level convolutional networks for text classification. « Proceedings of the 28th International Conference on Neural Information Processing Systems», Volume 1, 7 December, 2015, 649–657.

10. arXiv.org. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. 2017. URL: https://arxiv.org/pdf/1511.00561.pdf (дата обращения: 30.04.2021).

11. Р.А. Соловьев, Д.В. Тельпухов, А.Г. Кустов. Автоматическая сегментация спутниковых снимков на базе модифицированной свёрточной нейронной сети UNET. «Инженерный вестник дона», Т 1 (2017), № 4, 1-36.

12. Google Colaboratory. Добро пожаловать в Colaboratory!. URL: https://colab.research.google.com/notebooks/intro.ipynb (дата обращения: 30.04.2021).

Вещественная арифметика в ДССП для троичной машины

А.А.Бурцев[1]

[1] ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. Созданный в НИЛ троичной информатики на факультете ВМК МГУ программный комплекс ДССП-ТВМ можно использовать в качестве среды разработки и прогона программ для троичного компьютера. Ранее в нём были предусмотрены лишь операции целочисленной арифметики. Теперь для ДССП-ТВМ создан пакет, дополняющий словарь ДССП операциями вещественной арифметики с плавающей точкой.

В статье характеризуются основные возможности представленного пакета, а также поясняются ключевые аспекты его реализации на языке ДССП-Т в интерпретаторе ДССП/ТВМ.

Ключевые слова: троичная симметричная система счисления, троичная машина, троичная логика, троичная арифметика, ДССП, вещественная арифметика с плавающей точкой.

1. Введение

Двоичная микроэлектроника уже исчерпала весь могучий потенциал своих технологических возможностей. В качестве одного из возможных путей дальнейшего развития вычислительной техники предлагается вновь обратиться к троичной симметричной системе счисления (ТССС) с цифрами (–1,0,+1). Перспективность такой системы счисления когда-то отметил ещё Дональд Кнут [1], а её вычислительная эффективность была убедительно подтверждена практической разработкой и применением троичных цифровых машин «Сетунь» и «Сетунь-70», созданных более полувека назад [2] в НИЛ ЭВМ МГУ под руководством Брусенцова Н.П.

О преимуществах троичных машин написано немало книг и статей. Как самим Брусенцовым [3], так и его соратниками и учениками [4,5]. Проявляются они не только в упрощении аппаратной реализации компьютерных блоков (микросхем) и в построении ПО троичного компьютера. Главный выигрыш троичности состоит в том, что она открывает ряд новых возможностей [6], которые привносит в мир вычислительной техники троичная логика и троичная арифметика.

Обычно в списке основных преимуществ, обеспечиваемых троичной (симметричной) системой счисления, отмечают следующие.

1. Троичная – самая экономичная система счисления (т.к. ближе к числу e=2.71828) [7].

2. Тем же количеством разрядов можно кодировать больший диапазон чисел.

3. Можно единообразно обрабатывать как положительные, так и отрицательные числа, для представления которых уже не требуется прибегать к каким-либо «ухищрениям» (например, так называемому дополнительному коду).

4. Возможность непосредственного воплощения логики с тремя значениями: (+1, -1, 0), которые могут быть истолкованы как результат троичного варианта ответа на вопрос: “да”, “нет”, “не знаю” или “может быть”.

5. Упрощённая реализация ряда арифметических операций (сдвига, сравнения чисел, смены знака), а также трёхзначность функции “знак числа”.

6. Оптимальное округление чисел простым отсечением младших разрядов и взаимокомпенсируемость погрешностей округления в процессе вычислений [8]. Особую значимость такое преимущество приобретает при реализации операций вещественной арифметики.

Недавно в преимуществах троичной вычислительной техники убедились и разработчики квантовых компьютеров, опубликовав на просторах Интернета статью с вызывающим заголовком: “будущее квантовых компьютеров — в троичных вычислениях” [9]. Троичность может положительно сказаться и при построении биоморфных нейропроцессоров, поскольку учёные (нейробиологии) сделали важное открытие, что нейрон мозга человека тоже троичен [10].

Всё это даёт серьёзные основания утверждать, что с троичными вычислениями и троичными компьютерами будет связано будущее вычислительной техники. Поэтому проводимые в мире исследования возможных путей дальнейшего развития вычислительной техники не обходят вниманием и вопросы возможной аппаратной реализации троичных элементов на современной элементной базе [11].

В НИЛ Троичной Информатики ВМК МГУ на протяжении ряда лет (с 2010 по 2017 гг.) Масловым С.П. (соратником Брусенцова Н.П.) были проведены исследования, в результате которых были предложены [12] способы реализации на

современной элементной базе таких аппаратных троичных элементов, которые могли бы функционировать аналогично тем, что были реализованы в машинах семейства «Сетунь».

С 2011 года в качестве среды разработки и прогона троичных программ (т.е. программ для троичного компьютера) можно использовать программный комплекс ДССП-ТВМ [5], созданный в НИЛ ТИ на факультете ВМК МГУ. Он включает имитатор троичной машины ТВМ и систему разработки программ для нее (кросс-компилятор и интерпретатор) на языке ДССП-Т – троичном варианте языка ДССП. ДССП – Диалоговая Система Структурированного Программирования, созданная в НИЛ ЭВМ МГУ под руководством Брусенцова в 80-х годах XX века и затем реализованная на микрокомпьютерах самых разнообразных архитектур [13].

Ранее в ДССП-ТВМ можно было создавать троичные программы, используя лишь операции целочисленной арифметики. Теперь для ДССП-ТВМ создан пакет, дополняющий словарь ДССП операциями вещественной арифметики с плавающей точкой.

В статье характеризуются основные возможности представленного пакета, а также поясняются ключевые аспекты его реализации на языке ДССП-Т в интерпретаторе ДССП/ТВМ.

2. Общая характеристика пакета вещественной арифметики

Этот пакет загружается в интепретаторе ДССП командой **`LOAD Real`**:

```
TVM - Ternary virtual machine
tvm (PC=#00000000000000)> go; q;
****** DSSP/TVM *******  v.3d
* LOAD Real
* SAVE DSSP-3dR
```

А команда **`SAVE`** позволяет сохранить (в файле **`DSSP-3dR.nth`**) обновлённую версию интерпретатора, уже включающую этот пакет.

Подлежащие обработке вещественные значения размещаются в стеке операндов как особым образом закодированные 27-тритные величины. И все предусмотренные в пакете вещественные операции действуют над величинами, помещёнными в стек.

Двуместные вещественные операции сложения, вычитания, умножения и деления **R+ R- R* R/** принимают из стека два верхних его элемента (вершину и подвершину) в качестве аргументов, а затем результат тоже помещают в стек. А одноместные операции смены знака **RNEG** и получения абсолютной величины **RABS** действуют только над вершиной стека.

Помимо этих основных операций предусмотрены также эффективно реализуемые дополнительные операции умножения на 3, 9, 10: **R*3 R*9 R*10**, а также на их целочисленные степени: **R*3^ R*9^ R*10^**. И аналогичные операции деления на 3, 9, 10: **R/3 R/9 R/10**.

Для сравнения вещественных значений предусмотрен такой же богатый ассортимент операций, как и для сравнения целочисленных значений. Он включает не только привычные операции: **R< R= R> R<= R>=**, дающие двоичный логический ответ (0 или 1). Но и операции, выдающие троичный результат: **RSGN RCMP** :

```
{z} RSGN { = -1 (z<0), 0 (z=0), +1 (z>0) }
{x,y} RCMP { = -1 (x<y), 0 (x=y), +1 (x>y) }
```

Предусмотрены также и полезные операции для двоичной **RSEG** и троичных проверок **RTSEG RTSEG!** попадания точки в сегмент:

```
{a,b,x} RSEG { = +1 if x in[a,b] else 0 }
{a,b,x} RTSEG {=+1 (a<x<b), 0(x=a|x=b), else 0 }
{a,b,x} RTSEG! {= -1(x<a),0(x in[a,b]),+1(b<x) }
```

С помощью операции **mkReal#** вещественное значение **X** с плавающей точкой можно сформировать явно из двух составляющих его величин: мантиссы **m** и порядка, приготовленных в закодированном виде. А из сформированного вещественного значения можно получить его закодированные значения мантиссы и порядка с помощью операций **Rp** и **Rm** :

```
{p,m} mkReal# {X}
{X} Rp {p}    {X} Rm {m}
```

С помощью операции **Int->Real** вещественное значение можно получить из целочисленного. А с помощью операции **Real->Int** можно, наоборот, получить целочисленное значение из целой части вещественного:

```
{N} Int->Real {X}
{X} Real->Int {N}
```

Логическая операция **RNorm?** позволяет проверить, нормализовано ли вещественное число, а операция **RNorm** позволяет, если нужно, привести его к нормализованному виду.

Для обеспечения ввода и вывода вещественных чисел в привычном десятичном виде предусмотрены операция преобразования строки символов в вещественное значение **Str->Real** и операция получения из вещественного числа строки **Real->Str**, его представляющей. Эти операции используются интерпретатором при формировании вещественного значения в стеке при вводе из входного потока изображающей его строки, а также при выдаче на терминал вещественного значения вершины стека в обычном десятичном формате с помощью слов: **.Real .Realp .R .?R** .

В случае невозможности исполнения какой-либо операции предусмотрено возбуждение соответствующей исключительной ситуации. Например, если операции **Str->Real** не удаётся

сформировать вещественное число из заданной строки, то возникает ситуация **WrongReal!**.

Операция **Str->Real** воспринимает вещественное число в любом из следующих привычных форматов его представления (см. табл.1).

Таблица 1.Форматы вещественного числа

Форматы	**Примеры**
±dd.	2. 123. +17. –5 –1234.
±dd.ff	+3.14 0.5 –1234.567
±dd±Epp	1E+6 –1E-7 12E34
±dd.ff±Epp	5.6E+12 –0.1E-9 +12.34E56
± - возможный знак + или – **dd** – десятичные цифры целой части **ff** – десятичные цифры дробной части **pp** – десятичные цифры порядка	

Для некоторых особо употребительных вещественных констант в словарь ДССП пакетом уже занесены представляющие их слова:

```
0.0 1.0 0.1 10.0
Pi 2Pi Pi/2 EXP1 LN2 LN3 LN10
+MAXREAL –MAXREAL
+MINREAL –MINREAL
```

Перед началом использования операций пакета должно вызываться слово **InitReal**.

Вместе с пакетом **Real** библиотека ДССП оснащена дополнительным пакетом **RealF**, в котором представлены типичные часто используемые функции вещественной арифметики. Командой **LOAD RealF** (в интерпретаторе ДССП/ТВМ) можно осуществить загрузку этого дополнительного пакета, вызвав затем командой **InitRealFunc** его инициализацию.

В настоящее время в этом пакете уже определены слова для вычисления экспоненты (**EXP**), натурального логарифма (**LN**), квадратного корня (**SQRT**), синуса (**SIN**), косинуса (**COS**). Предусмотрены также функции возведения произвольного вещественного значения в натуральную степень (**R^**) и эффективного вычисления полинома (**PLNM**), т.е. многочлена с заданными коэффициентами по схеме Горнера.

Для некоторых функций определены слова-дубликаты: **EXPn LNn SQRTn SINn COSn**, в которых дополнительным параметром **n** явно задаётся, сколько итераций осуществить (при исполнении функции) для достижения нужной точности вычислений. При вызове основного слова функции (например,**EXP**) точность вычисления определяется значением, установленным в переменной EPS (по умолчанию равной константе EPS_=1E-6), а количество итераций не превосходит максимального значения, установленного константой MAXRFCnt (=99).

Вычисление функции можно осуществлять в отладочном режиме с печатью важнейших промежуточных значений на каждой итерации, если перед вызовом функции установить (в 1) переменную **RFDbg**. В другой переменной **RFCnt** фиксируется реальное количество итераций, затраченных на исполнение функции.

Если вычислить вызванную функцию не представляется возможным, возбуждаются соответствующие исключительные ситуации. В настоящее время в пакете **RealF** предусмотрены две таких ситуации: 1) ситуация **SQRT<0!** возбуждается, если вызывается функция **SQRT** с отрицательным аргументом; 2) а ситуация **LN<=0!** возбуждается, если вызывается функция **LN** с неположительным аргументом (<=0).

3. Реализация основных операций вещественной арифметики

В настоящее время все операции над троичными вещественными числами в форме с плавающей точкой реализованы в ДССП-ТВМ на основе операций целочисленной арифметики на языке ДССП-Т. Познакомимся подробнее с приёмами реализации основных арифметических операций вещественной арифметики.

3.1. Представление вещественного числа в форме с плавающей точкой

В ДССП для троичной машины вещественное число в форме с плавающей точкой представляется в памяти одним 27-тритном машинным словом, состоящем из 3-х трайтов: в старшем трайте слова (в разрядах 18-26) записывается значение порядка **p**, а в двух младших трайтах (в разрядах 0-17) располагается значение мантиссы **m** (см. рис.1). Значение представленного таким способом троичного вещественного числа **X** формируется как произведение $\mathbf{X}=\mathbf{m}\times 3^{\mathbf{p}}$.

Нормализованное троичное вещественное число – это число, мантисса которого содержит не ноль в первой старшей цифре (17-м разряде). Такая мантисса **m** по абсолютной величине является дробным числом в диапазоне от 1/6 до 1/2, т.е. $1/6 < |\mathbf{m}| < 1/2$.

Обеспечиваемая такой 18-тритной мантиссой точность представления вещественного числа равна 3^{-18} ($\approx 2.58\times 10^{-9}$); и это уже значительно лучше, чем 2^{-24} ($\approx 59.9\times 10^{-9}$) в случае 32-битного формата single float, применяемого в современных двоичных компьютерах.

Используемое 9-тритное значение порядка, варьируемое в пределах [-9841,+9841], позволяет задавать диапазон вещественных чисел (по абсолютной величине) от минимального значения 0.5×3^{-9842} ($\approx 0.22\times 10^{-4695}$) до максимального значения $0.5\times 3^{+9841}$ ($\approx 1.12\times 10^{+4695}$), что значительно перекрывает диапазон вещественных чисел формата single float двоичных машин.

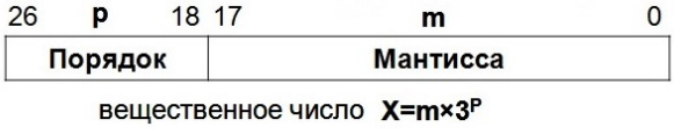


Рис. 1.Схема представления троичного вещественного числа в форме с плавающей точкой

Рис. 2. Схема выполнения операции нормализации вещественных чисел

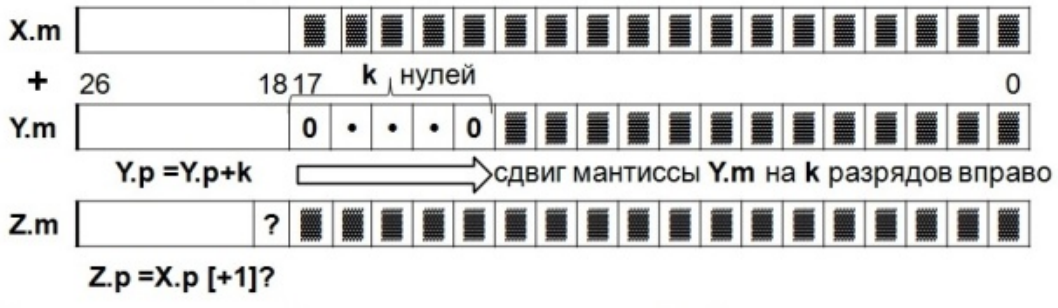


Рис. 3. Схема выполнения операции сложения вещественных чисел

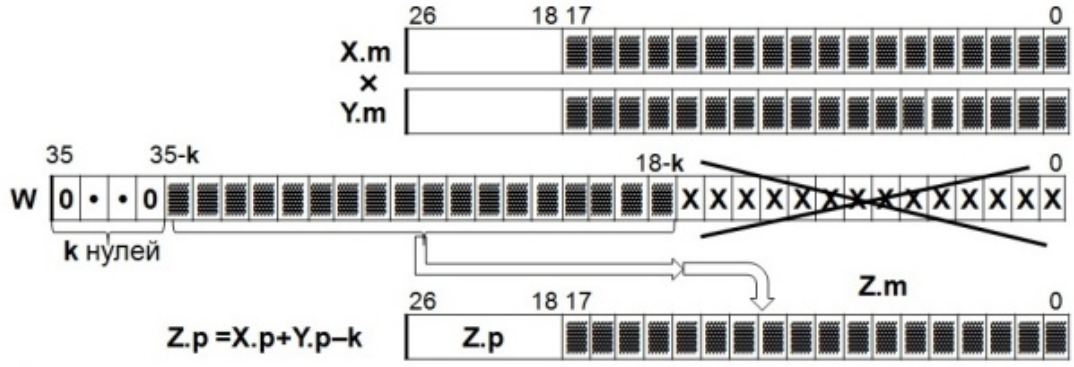


Рис. 4. Схема выполнения операции умножения вещественных чисел

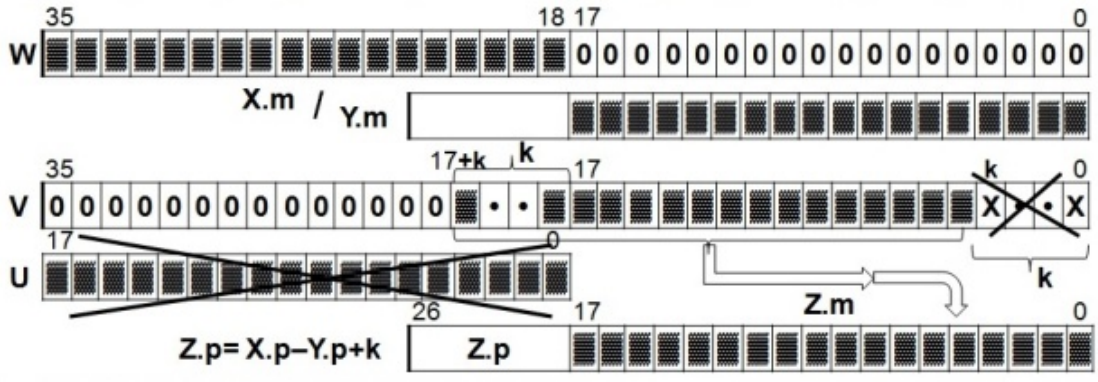


Рис. 5. Схема выполнения операции деления вещественных чисел

Для представления нулевого вещественного числа (0.0) сделано исключение: оно представляется нулевой мантиссой (m=0) и нулевым порядком (p=0), т.е. нулями во всех разрядах.

3.2. Операция нормализации

Далее предполагается, что основные арифметические операции выполняются над вещественными значениями, представленными в нормализованном виде. Для приведения ненулевого вещественного значения к нормализованному виду потребуется совершить ряд действий (см. рис.2).

1. Сдвинуть мантиссу **m** влево на несколько (**k**) разрядов, пока в её старшем (17-ом) разряде не появится ненулевое значение (-1 или +1).

2. И затем уменьшить значение порядка **p** на значение количества **k** передвинутых разрядов.

3.3. Операция сложения и вычитания

Заметим, что когда один из аргументов операции сложения нулевой, тогда другой аргумент просто будет результатом их сложения. Поэтому предполагается, что в описываемой ниже схеме операции сложения участвуют ненулевые нормализованные вещественные значения.

Для сложения двух вещественных значений **X+Y** совершим следующий ряд действий (см. рис.3).

1. Выясним, у какого вещественного значения больше порядок; предположим, что большим будет порядок числа **X: X.p>Y.p**; выясним, на какое значение: **k=X.p–Y.p**.

2. Уравниваем порядки складываемых чисел: **Y.p=Y.p+k.**

3. И сдвигаем мантиссу **Y.m** вправо на **k** разрядов.

4. Теперь складываем мантиссы как 18-тритные целые числа: **Z.m=X.m+Y.m**.

5. Если при этом возник перенос (+1 или -1) в 18-й разряде, то сдвигаем мантиссу **Z.m** вправо на 1 разряд и корректируем порядок: **Z.p=Z.p+1**.

6. Если получено ненормализованное вещественное значение Z (17-й разряд нулевой), то выполняем над ним операцию нормализации.

Для исполнения же операции вычитания двух вещественных значений **X–Y** выполним два действия: сначала изменим знак у 2-го аргумента: **–Y** (для этого инвертируем его мантиссу); а затем уже выполним операцию сложения: **X+(–Y)**.

3.4. Операция умножения

Для умножения двух вещественных значений **X×Y** выполним следующий ряд действий (см. рис.4).

1. Складываем их порядки: **Z.p= X.p + Y.p**.

2. Перемножаем их мантиссы как 18-разрядные целые числа: **W= X.m × Y.m**.

3. У полученного 36-разрядного целого числа **W**[35..0] выделяем 18-разрядный участок, который начинается со старшего (**35-k**)-го ненулевого разряда.

4. Принимаем эту выделенную старшую часть **W** в качестве мантиссы результата: **Z.m= W[35-k..18-k]**; а оставшуюся младшую часть **W** отбрасываем.

5. Проведя такую нормализацию **W**, корректируем порядок результата **Z.p= Z.p – k**.

Заметим, что простота округления вещественного числа в ТССС как раз и выражается в том, что младшую часть **W** здесь анализировать не надо! (т.к. её можно просто отбросить!)

3.5. Операция деления

Для деления вещественного значения **X** на ненулевое вещественное значение **Y** выполним следующий ряд действий (см. рис.5).

1. Вычитаем порядки: **Z.p= X.p – Y.p** .

2. Мантиссу **X.m** превращаем в 36-разрядное целое число $\mathbf{W= X.m \times 3^{18}}$.

3. Делим 36-разрядное целое **W** на мантиссу **X.m** как на 18-разрядное целое число, получая 36-разрядное целое частное **V** и 18-разрядное целое **U** в качестве остатка.

4. У величины **V** выделяем 18-разрядный участок, который начинается со старшего (**17+k**)-го ненулевого разряда, и принимаем эту часть **V** в качестве мантиссы результата: **Z.m= V[17+k.. k]**; оставшуюся часть **V** (как и величину **U**) отбрасываем

5. Проведя такую нормализацию мантиссы, корректируем порядок результата: **Z.p= Z.p + k**.

Заметим, что простота округления вещественного числа в ТССС позволяет просто отбросить величину **U** и младшую часть **V** (т.к. их анализировать не надо!) и тем самым упростить реализацию операции деления.

4. Представление вещественных чисел в виде символьных строк

Познакомившись с тем, в каком закодированном виде вещественные значения хранятся в памяти троичной машины и как над ними исполняются арифметические действия, обратимся теперь к проблемам представления таких значений в символьном виде, удобном для восприятия их человеком.

4.1. Получение значения вещественного числа из строки символов

Операция **Str->Real** позволяет получить значение вещественного числа из строки символов, соответствующей одному из принятых форматов, представленных в таблице 1. Приведём примеры применения этой операции для получения вещественных значений:

```
"123.45E+7" Str->Real .R D CR
```

```
"-12345." Str->Real .R D CR
"123.45E-5" Str->Real .R D CR
"-12345," Str->Real .R D CR
```

А также примеры выдачи их на терминал с помощью операции **.R**, которая печатает вершину стека как вещественное число:

```
0.12345E+10
-12345.
0.0012345
WrongReal!"-12345,"-12345.
```

В последней строке выдачи отображена стандартная реакция на исключительную ситуацию **WrongReal!** с печатью «ошибочной» строки, вызвавшей исключение.

Введение в язык ДССП-Т операций вещественной арифметики влечёт за собой необходимость распознавать в тексте ДССП-программы вещественные константы-литералы. Значит, интерпретатор и кросс-компилятор ДССП следует «научить» понимать вещественные числа, встречающиеся в программе, как константы-литералы и формировать для них 27-тритный код, состоящий из мантиссы и порядка.

В ходе разбора символьной строки вида «**±dd.ff±Epp**» для формирования вещественного значения приходится выполнять операции сложения, умножения на 10 (для целой части) и деления на 10 (для дробной части) над троичными вещественными значениями. Интерпретатор ДССП/ТВМ с загруженным пакетом **Real** становится способным исполнять такие операции. Поэтому в интерпретаторе можно реализовать операцию преобразования **Str->Real**.

Но ещё надо «надоумить» интерпретатор вызывать эту операцию всякий раз, когда ему приходится разбирать введённую строку, анализируя, представляет ли она собой константу-литерал. Прежний алгоритм интерпретатора при разборе введённой строки-литерала не учитывал, что могут встретиться строки, представляющие вещественные константы. Как можно теперь «научить» интерпретатор правильно понимать и обрабатывать такие строки?

Для решения такой проблемы пришлось модифицировать алгоритм разбора констант-литералов интерпретатора следующим образом. В блок распознавания литералов интерпретатора добавлена возможность вызывать свою оригинальную процедуру распознавания строки-константы. Так чтобы саму подобную процедуру стало возможным задавать позднее непосредственно в ДССП-программе, обрабатываемой интерпретатором.

Продемонстрируем, как такая процедура назначается при инициализации пакета **Real** :

```
: GetRealVal{Adr,len}
   EON WrongReal! WrongRealCode!
   Str->Real +1 {RVal,Code=+1};
: WrongRealCode! {RealVal,Code}
   T-1 {RVal, Code=-1} ;
: InitGetRealVal {вызывается в InitReal}
 '' GetRealVal !GetSpecValueProc ;
```

Теперь (после такого назначения) интерпретатор будет вызывать процедуру **GetRealVal** всякий раз, когда он не сумеет во введённой строке распознать какую-либо константу-литерал прежними способами. Если процедура **GetRealVal**, вызвав операцию **Str->Real**, распознает в заданной строке вещественную константу, она вернёт сформированное для неё 27-тритное значение (RealVal) и успешный код завершения (Code=+1). Если не удастся распознать в строке вещественное число, то возникнет ситуация **WrongReal!**, в результате обработки которой будет установлен отрицательный код завершения (Code=-1), означающий, что распознать корректное вещественное число в заданной строке, увы, не удалось.

Таким образом, интерпретатор ДССП/ТВМ удаётся «научить» понимать вещественные константы. А вот «обучить» этому свойству кросс-компилятор ДССП-ТВМ, оказывается, не так-то просто. Ведь компилятор должен не только провести синтаксический разбор заданной строки, представляющей вещественную константу, но и получить её троичное представление в форме 27-тритного значения, складывающегося из закодированных троичных значений мантиссы и порядка. А для этого компилятор должен уметь исполнять вещественные операции сложения, умножения и деления с троичными числами. Понятно, что кросс-компилятор, функционирующий на двоичной машине, этих операций, увы, сделать не сможет. Получается, чтобы решить обозначенную проблему, компилятор надо снабдить собственным имитатором троичных операций вещественной арифметики. А это задача сама по себе весьма трудоёмкая.

4.2. Получение из вещественного числа символьной строки для печати

Чтобы полученные в результате вычислений вещественные значения представить в визуальном виде, необходимо уметь выводить их на печать (например, на экран терминала). В простейшем случае вещественное значение вершины стека можно просто изобразить в виде двух троичных или 9-тичных значений мантиссы и порядка, что обеспечивается операцией **.Real#.** Например, фрагмент ДССП-программы:

```
."3.=" 3. .Real# CR
."-27.=" -27. .Real# CR
."-1/9=" 1. 9. R/ .Real# CR
."-1/81=" -1. 81. R/ .Real# CR
```

выдаст на печать:

```
3.=0.#300000000E#2
-27.=0.#600000000E#4
1/9=0.#300000000E#8
-1/81=0.#600000000E#6
```

Однако, желательно всё же изображать вещественные значения на печати в традиционной десятичной форме их записи. Поэтому в пакете **Real** предусмотрена операция преобразования **Real->Str**, которая позволяет получить строку с изображением вещественного значения в десятичном виде в формате с фиксированной или плавающей точкой (как в табл.1):

```
{X,p,Str} Real->Str {Str,Len}      TOS
```

При этом параметр p задаёт количество значащих цифр в десятичной мантиссе. Если p>0, то число (если это возможно) изображается в форме с фиксированной точкой (и без незначащих нулей в дробной части). Если p<0, то число изображается обязательно в форме с плавающей точкой. А при p=0 значение p берётся по умолчанию (=9).

Сформированную такой операцией строку {Str,Len} можно сразу выдавать на печать командой **TOS**. Именно так и исполняется основная команда печати вещественного значения вершины стека **.Realp** :

```
: .Realp {Y,p} 'RealStr Real->Str TOS { };
: .Real {Y} 0 .Realp ;
: .R C .Real ;
: .?R .? SP .R ;
```

Остальные команды печати используют её с параметром p=0. Команда **.R** оставляет печатаемое значение в вершине стека. А команда **.?R** дополняет действие команды **.?**, изображающей вершину стека в различных целочисленных форматах (троичном, 9-ном, 10-ном и 16-ном), после чего распечатывает вершину стека ещё и как 10-ное вещественное число.

Приведём примеры их использования:

```
12345.67  7 .Realp CR
12345.67 -7 .Realp CR
12345.67 .Real CR
```

вместе с результатами выдачи на печать, которые характеризуют различие этих команд:

```
12345.67
0.1234567E+5
12345.6702
```

Заметим, что не всякое вещественное число, записанное в десятичной системе счисления (например, числа 0.5 и 0.1), может быть точно представлено в троичном коде. И наоборот, некоторые числа, точно представимые в троичном коде (например, 1/3 или 1/9), невозможно точно изобразить вещественным десятичным числом. Так что в результате исполнения вычислений:

```
."1/3=" 1. 3. R/ .R CR
```

```
."1/9=" 1. 9. R/ .R CR
```

получим такую выдачу на печать:

```
1/3=0.333333333
1/9=0.111111111
```

При реализации операции преобразования **Real->Str** возникает необходимость решить следующую задачу: как из троичного вещественного числа $\mathbf{X=m\times3^p}$ получить его десятичное представление в виде: $\mathbf{X=z\times10^q}$ такое, чтобы мантисса **z** оказалась по абсолютной величине в диапазоне: $0.1\le|\mathbf{z}|<1.0$. В результате надо получить значение мантиссы $\mathbf{z=X/10^q}$ и само значение порядка **q**.

Для решения этой задачи можно предложить такой алгоритм действий. При p>0 многократно (в цикле) делить X на 10., пока не станет |z|<1.0, подсчитывая при этом, сколько раз (**q**) было исполнено такое деление. А при p<0, наоборот, умножать X на 10., пока не станет $|\mathbf{z}|\ge0.1$, и подсчитывать при этом отрицательное значение порядка **q**.

Но при больших значениях порядка **p** такой алгоритм будет работать слишком медленно. Для его ускорения можно предложить предварительно скорректировать значение X, поделив (или умножив) его сразу на величину $\mathbf{10^t}$, взяв $\mathbf{t=p}/2$ ($\mathbf{p=2t+j}$). Тогда в качестве первого приближения для **z** можно получить величину $\mathbf{z=m\times3^j\times9^t/10^t}$ и **q=t**. А далее уже продолжить действия по описанному алгоритму.

5. Примеры применения пакета вещественной арифметики

Проиллюстрируем применение операций вещественной арифметики примерами разработки некоторых программ в ДССП-ТВМ.

5.1. Пример реализации стандартной функции EXP

Приведённый ниже фрагмент программного кода (на языке ДССП-Т) определяет слово **EXPn**, которое реализует функцию вычисления экспоненты в виде ряда:

$$\sum_{k=0}^{n}\frac{x^k}{k!}=1+\frac{x}{1!}+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}$$

Вычисление ряда заканчивается при достижении заданного (n) максимального количества слагаемых или возможно раньше, когда очередной член ряда, добавляемый в накапливаемую сумму (Sk), уже оказывается меньше (по абсолютной величине) величины желаемой точности, предварительно заданной словом EPS.

```
: EXP {X} MAXRFCnt EXPn {exp(X)} ;
: EXPn {X,n} 1.0 C C E4 0 E2
   {X,Fk=1!=1.0,Xk=1.0,Sk=1.0,k=0,n}
```

```
   DO- EXPk ! RFCnt E4 DDD {Res=Sk};
: EXPk {X,Fk=k!,Xk=X^k,Sk,k,i} E2 1+ E2
   5 ET C2 Int->Real R* 5 ET E4
   6 CT R* E4 C4 6 CT R/ E4 C4 R+
   E4 C EPS? BR+ EPS! D ;
: EPS? {Z} RABS EPS R<= {|Z|<=EPS?} ;
: EPS! {i,Z} DD 0 {i=0} ;
```

А словом **EXP** функция **EXPn** вызывается с максимально предусмотренным параметром **n**, задаваемым словом-константой MAXRFCnt.

Примеры вызовов функций **EXP** и **EXPn** :

```
." EXP(1.)= " 1.0 EXP .R CR
." EXP(-1.)= " -1.0 EXP .R CR
." EXP(-2.)= " -2.0 9 EXPn .R CR
." EXP(3.)= " 3. 20 EXPn .R D .RFCnt CR
." EXP(5.0)= " 5.0 EXP .R D .RFCnt CR
```

В итоге этих вызовов на экран терминала выдаются следующие результаты:

```
EXP(1.)= 2.7182818
EXP(-1.)= 0.36787919
EXP(-2.)= 0.135097
EXP(3.0)= 20.085537 [17]
EXP(5.0)= 148.41316 [23]
```

Заметим, что при вызове слова .RFCnt здесь печатается (в квадратных скобках) количество слагаемых, которое было учтено в итоговой сумме при вычислении ряда функции.

5.2. Пример процедуры поиска корня уравнения

Продемонстрируем применение операций вещественной арифметики для реализации в ДССП-ТВМ процедуры поиска единственного корня уравнения F(X)=0 на отрезке [a,b] простым методом деления отрезка пополам. При вызове этой процедуры **EQUROOT** предусмотрена проверка, что на концах этого отрезка функция принимает значения разных знаков. Если это условие нарушено, то вызывается исключительная ситуация **NoRoot!**, означающая, что вызванная процедура не сможет найти требуемый корень на заданном отрезке.

```
SITUATION NoRoot! .NoRoot{Нет корня!}
ACT VAR F {указатель на тело функции F(X) }
: EQUROOT_{a,b} GTP {a,b,F} EQUROOT {X} ;
: EQUROOT {a,b,F} ! F C2 F C2 F
   isRoot? BR+ FindRoot NoRoot! {X};
: isRoot? {a,b,Fa,Fb} C2 RSGN C2 RSGN
    NEG = {(sign(Fa)=-sign(Fb)?} ;
: FindRoot {a,b,Fa,Fb} C2 C2 R>
     {Fa>Fb?} IF+ ba {swap [a,b]}
   {a,b,Fa,Fb} getFcRoot? DW new[a,b]
     D 5 ET DDDD {X=c} ;
: ba{a,b,Fa,Fb} E2 E3 E4 E3 {b,a,Fb,Fa};
: getFc {a,b,Fa,Fb} C4 C4 R+ 2. R/
   C F {..,c=a+b)/2,Fc=F(c)} ;
: getFcRoot? getFc {a,b,Fa,Fb,c,Fc} 6 CT
  6 CT =?EPS C2 EPS? OR NOT {..,~Lv} ;
    { Lv= (|a-b|<=EPS?) or (|Fc|<=EPS?) }
: =?EPS {a,b} R- RABS EPS R<= ;
: new[a,b]{a,b,Fa,Fb,c,Fc} C RSGN
   BR+ [a,c] [c,b] {a',b',Fa',Fb'} ;
: [a,c] E3 D E4 D {a,b=c,Fa,Fb=Fc} ;
: [c,b] E4 D 5 ET D {a=c,b,Fa=Fc,Fb} ;
```

В переменной F запоминается указатель на тело заданной функции. Предусмотрена также и другая процедура **EQUROOT_**, при вызове которой имя-указатель функции можно передавать не в стеке, а задавать тут же в теле на месте вызова этой процедуры.

Применим эти процедуры для решения двух трансцендентных уравнений:

(1) $\sin x = x^2 - 1$ на отрезке $[0,\pi]$

(2) $e^x = x + 2$ на отрезке [1,2]

Для этого определим слова F1 и F2, задающие вычисление соответствующих функций:

```
{ F1(X)= sin(X) -  X^2 + 1 }
: F1{X} C SIN E2 C R* R- 1. R+ ;
{ F2(X)= exp(X) -  X- 2 }
: F2{X} C EXP E2 R- 2. R- ;
```

И осуществим с ними вызов рассмотренных процедур поиска корня:

```
."root for sin(X)-X^2+1.=0 in[0.,Pi]=" CR
 SP 0. Pi EQUROOT_ F1 .R CR
."root for exp(X)-X-2.=0 in[1.,2.]=" CR
 SP 1. 2. '' F2 EQUROOT .R CR
```

В итоге этих вызовов получим такой вот результат выдачи на экран терминала:

```
root for sin(X)-X^2+1=0 in[0.,Pi]=
 1.409624
root for exp(X)-X-2=0 in[1.,2.]=
 1.14619325
```

Проверочные вызовы функций с этими значениями корней выдают нулевые значения.

6. Заключение

В статье представлены основные возможности пакета операций вещественной арифметики для ДССП троичной машины (ТВМ). Пояснены важнейшие аспекты его реализации на языке ДССП-Т в интерпретаторе ДССП/ТВМ. Продемонстрированы примеры его применения для построения типовых вычислительных программ в интерпретаторе ДССП/ТВМ.

Таким образом, теперь для троичной машины в ДССП/ТВМ стало возможным создавать вычислительные программы с использованием операций вещественной арифметики.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2021-0004

«Разработка архитектуры, системных решений и методов для создания микропроцессорных ядер и коммуникационных средств семейства систем на кристалле двойного назначения. 0580-2021-0004», Рег. № 121031300049-0.

Real Arithmetic in DSSP for Ternary Machine

A.A. Burtsev

Abstract. The DSSP-TVM software complex created at the Research Laboratory of Ternary Informatics at the Faculty of Computational Mathematics and Cybernetics of Moscow State University can be used as an environment for developing and running programs for a ternary computer. Previously, it only provided integer arithmetic operations. Now for DSPP-TVM a package has been created that supplements the DSSP dictionary with operations of real floating point arithmetic.

The article describes the main features of the presented package, and also explains the key aspects of its implementation in the DSPP-T language in the DSPP/TVM interpreter.

Keywords: ternary symmetric number system, ternary machine, ternary logic, ternary arithmetic, DSSP, floating point real arithmetic.

Литература

1. Кнут Д. Искусство программирования на ЭВМ. Т.2 Получисленные алгоритмы. М., Мир, 1977, п.4.1, 216-219.

2. Н.П. Брусенцов Н.П., Х. Рамиль Альварес. Троичные ЭВМ “Сетунь” и “Сетунь 70”. «Первая Международная конференция "Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы" SORUCOM-2006», Россия, Петрозаводск, 3-7 июля 2006. Петрозаводск, Изд-во ПетрГУ, 2006, ч.1, 45-51.

3. Брусенцов Н.П. Заметки о троичной цифровой технике. «Вычислительная техника и вопросы кибернетики», Т.15 (1978), 145–155.

4. Владимирова Ю.С. Введение в троичную информатику: учебное пособие. М., АРГАМАК-МЕДИА; 2015.

5. Бурцев А. А., Сидоров С. А. Троичная виртуальная машина и троичная ДССП. «Программные системы: теория и приложения» Т.6 (2015), №4, 29–97, http://psta.psiras.ru/read/psta2015_4_29-97.pdf.

6. Бурцев А. А. Особенности программирования троичной машины: новые возможности и новые задачи. «Труды НИИСИ РАН», Т. 10 (2020), № 3, 49–60.

7. А.А. Бурцев, В.А. Бурцев. О преимуществах троичных машин и эффективности троичных вычислений. «Труды НИИСИ РАН», Т. 10 (2020), № 3, 60–65.

8. Ким Г.Д., Воеводин В.В. Машинные операции с точки зрения математика. «Вычислительные методы и программирование», Т. 26 (1977), 31-35.

9. Будущее квантовых компьютеров — в троичных вычислениях, http://www.infuture.ru/news.php?news_id=475pdf.

10. Brain circuitry findings could shape computer design, http://cbcl.mit.edu/news/files/liu-tp-picower.html

11. Zarin Tasnim Sandhie, Jill Arvindbhai Patel, Farid Uddin Ahmed, Masud H. Chowdhury. Investigation of Multiple-valued Logic Technologies for Beyond-binary Era. ACM Comput. Surv. 54, 1, Article 16 (January 2021), 30 pages, https://doi.org/10.1145/3431230.

12. Маслов С.П. Об одной возможности реализации троичных цифровых устройств. «Программные системы и инструменты» Т.12 (2011), 222-227.

13. Бурцев А.А., Сидоров С.А. История создания и развития ДССП: от "Сетуни-70" до троичной виртуальной машины. «Вторая Международная конференция “Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР” SORUCOM-2011», Россия, В.Новгород, 12-16 сентября 2011. В.Новгород, Изд-во НовГУ, 2011, 83-88.

Подписано в печать 22.07.2021 г.
Формат 60x90/8
Печать цифровая. Печатных листов 5,25
Тираж 100 экз. Заказ № 544

Отпечатано в ППП «Типография «Наука»
121099, Москва, Шубинский пер., 6