RU EN ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ ISSN 2079-3316
научная статья ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ, НЕЙРОННЫЕ СЕТИ

УДК 004.021+004.272
10.25209/2079-3316-2025-16-4-51-79

Алгоритм прямого распространения в парадигме потока данных

Дмитрий Николаевич **Змеев**[1✉], Николай Николаевич **Левченко**[2✉], Аркадий Валентинович **Климов**[3✉]
Национальный исследовательский центр «Курчатовский институт», Москва, Россия
[1✉]*zmejevdn@list.ru*, [2✉]*nick@burcom.ru*, [3✉]*arkady.klimov@gmail.com*

Аннотация. В статье рассматриваются вопросы разработки и реализации алгоритма прямого распространения в парадигме управления вычислениями потоком данных. Принципы управления вычислениями потоком данных (dataflow) существенно отличаются от традиционного управления потоком команд (control-flow), как и сама парадигма программирования потоков данных отличается от императивной. Программы, созданные в потоковой парадигме, являются изначально параллельными благодаря тому, что параллелизм задачи извлекается из потока данных автоматически на аппаратном уровне. В статье представлено подробное описание потокового алгоритма прямого распространения, который реализован в потоковой программе, предназначенной для исполнения на параллельной потоковой вычислительной системе «Буран».

Потоковая программа отличается компактностью и универсальностью кода. Она автоматически масштабируется на всю систему, а ее программный код не содержит ссылок на библиотечные функции и базируется исключительно на базовых арифметических операциях, таких как сложение, умножение и сравнение. Программа способна обрабатывать персептроны любой размерности без модификации и перекомпиляции программного кода. Размер и структура обрабатываемого персептрона определяется начальными данными. В экспериментальной части статьи продемонстрирована высокая адаптивность потоковой программы к обработке больших объемов непрерывно поступающих данных. Дана оценка применимости потоковой модели вычислений для решения нейросетевых задач.

Ключевые слова и фразы: алгоритм прямого распространения, параллельные вычисления, управление вычислениями на основе потока данных, потоковая парадигма программирования, параллельная потоковая вычислительная система

Благодарности: Работа проведена в рамках выполнения государственного задания НИЦ «Курчатовский институт»

Для цитирования: Змеев Д.Н., Левченко Н.Н., Климов А.В. *Алгоритм прямого распространения в парадигме потока данных* // Программные системы: теория и приложения. 2025. **Т. 16**. № 4(67). С. 51–79. https://psta.psiras.ru/read/psta2025_4_51-79.pdf

Введение

Значимость нейросетевых технологий подтверждена их практическим применением и объемом ресурсов, направляемых на их развитие. Нейросетевые технологии, которые принято называть искусственным интеллектом, уже активно применяются в транспорте, медицине и для создания мультимедийного контента, ими даже начинают замещать целые пласты профессий (журналистика, кассиры, административные ассистенты, техническая поддержка и многие другие).

В настоящее время ведется активная разработка новых алгоритмов и аппаратных решений. Большинство подобных разработок базируется на использовании графических процессоров и центральных процессоров, использующих традиционные архитектурные подходы. В статье представлен альтернативный подход к программированию алгоритмов обработки нейронных сетей, основанный на принципе управления вычислениями потоком данных.

Базовым представителем нейросетей является персептрон. Изучение алгоритмов прямого прохода и обучения многослойного персептрона – это то, с чего начинает любой, кто пытается разобраться в принципах работы нейросетей. Если углубиться в программный код этих алгоритмов, то обращает на себя внимание следующее.

С одной стороны, алгоритм прямого прохода представляется в виде движения по однонаправленному графу, а с другой стороны – реализация этого алгоритма выполняется с помощью перемножения векторов и матриц. Алгоритм обратного распространения, в свою очередь, это движение по тому же графу (обозначающему персептрон), но в противоположном (прямому проходу) направлении.

Логично предположить, что программирование этих алгоритмов в концепции движения данных по графу (персептрону), а не математическое моделирование этого движения, может дать преимущество. Подобная концепция реализована в модели с управлением вычислениями потоком данных, более известной как dataflow.

В настоящее время в мировой научной литературе представлено ограниченное количество исследований, посвященных применению аппаратных dataflow-систем для реализации нейросетевых алгоритмов. Среди наиболее значимых публикаций можно выделить работы [1, 2], в которых для выполнения алгоритма прямого распространения многослойного персептрона используется dataflow-ускоритель, разработанный компанией Maxeler [3].

В этой и многих подобных работах авторы отображают полный потоковый вычислительный граф, в котором каждый нейрон представлен

своим подграфом, непосредственно на вентили, сумматоры, умножители и связи между ними, используя FPGA в качестве целевой платформы. В результате все нейроны работают параллельно и передают данные друг другу по прямым связям вместо записи в память, чем достигается высокая производительность.

Однако, этот подход либо накладывает сильные ограничения на размер графа, либо требует нетривиальных решений по сегментации (нарезке) графа на части, которые обрабатываются последовательно. Тем не менее, результаты, полученные в рамках этих исследований, демонстрируют высокий потенциал и перспективность дальнейших разработок в этой области.

Наш подход опирается на потоковый алгоритм, описывающий один обобщенный нейрон, многие экземпляры которого возникают динамически при его выполнении в вычислительной среде особой архитектуры. Эта архитектура воплощена в параллельной потоковой вычислительной системе (ППВС) «Буран», которая разрабатывается в рамках научно-исследовательской работы Отделения проблем проектирования в микроэлектронике Центра перспективной микроэлектроники Национального исследовательского центра «Курчатовский институт» на протяжении нескольких лет.

В ППВС реализована потоковая модель вычислений с динамически формируемым контекстом. Целью статьи является демонстрация результатов разработки, реализации и исследования алгоритма прямого распространения в контексте принципов управления вычислениями потоком данных, а также оценка перспектив его применения на ППВС «Буран».

Результаты, изложенные в статье, на текущем этапе не обладают практической применимостью ввиду начального уровня их разработки. Однако они могут представлять интерес для разработчиков и исследователей, стремящихся к нестандартным подходам в области нейросетевых технологий, и в первую очередь исследователям вопросов применения тернарной ассоциативной памяти для задач искусственного интеллекта.

Статья имеет следующую структуру. В первой главе кратко описывается потоковая модель вычислений и архитектура ППВС, а также приведены их ключевые отличия от традиционной фон-неймановской модели вычислений и архитектуры. Вторая глава содержит подробный разбор потокового алгоритма прямого распространения – от разработки в виде потокового графа до реализации на языке высокого уровня и выбора эффективной функции распределения. Третья глава состоит из анализа результатов моделирования ППВС при выполнении программы прямого

распространения и оценки перспектив применения ППВС для решения нейросетевых задач.

1. Потоковая модель вычислений с динамически формируемым контекстом и реализующая ее архитектура

В основе потоковой модели вычислений лежит принцип активации вычислений по готовности данных [4–6]. Этот принцип реализуется через механизм активации вычислительных процессов, когда все требуемые данные (как минимум два значения с идентичными контекстами) поступают в одно устройство, предназначенное для их хранения и сопоставления, то есть, активация производится по приходу последнего из требуемых данных. Для сравнения, в традиционной модели вычислений обработка данных управляется потоком команд, которые содержат (в общем случае) код операции, адреса хранения данных, подлежащих обработке, и адрес записи результата обработки.

Ключевым элементом потоковой модели вычислений является контекст данных, который представляет собой совокупность параметров, идентифицирующих конкретное данное в виртуальном адресном пространстве задачи. Структура, объединяющая данное (операнд), контекст и набор признаков, называется токеном. Контекст выполняет две ключевые функции:

(*1*) На основе сравнения контекстов производится объединение двух или более токенов в один пакет данных, который передается на вычислительное устройство, где выполняются соответствующие операции и формируются новые токены.

(*2*) На основе контекста выполняется распределение вычислений по конкретным вычислительным ресурсам системы, обеспечивая тем самым взаимодействие токенов с идентичными контекстами.

Подробное описание потоковой модели вычислений представлено в [7]. Концептуальные отличия этой модели от традиционной оказывают влияние как на архитектуру вычислительной системы, реализующей эту модель, так и на парадигму программирования, используемую для разработки программ, предназначенных для выполнения на такой системе.

Архитектура параллельной потоковой вычислительной системы «Буран» (рисунок 1), реализующая потоковую модель вычислений, представляет собой масштабируемую структуру, включающую набор вычислительных модулей, объединенных коммуникационной сетью, по которой передаются сообщения (токены). Подробное описание ППВС представлено в [8], ниже приведены ключевые особенности системы.

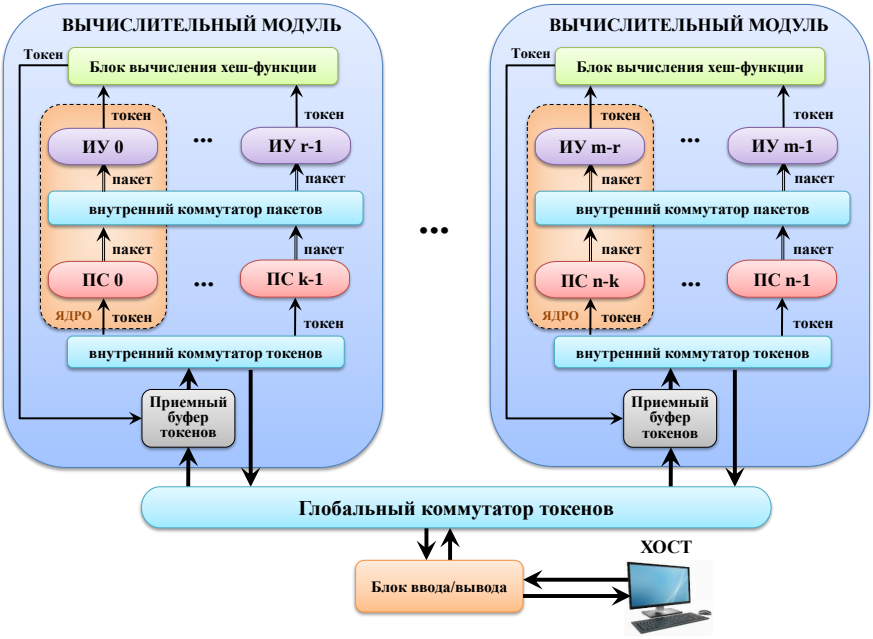


Рисунок 1. Архитектура параллельной потоковой вычислительной системы «Буран»

Каждый вычислительный модуль состоит из одного или нескольких процессоров сопоставления, коммутатора пакетов, одного или нескольких исполнительных устройств и блоков формирования хэш-функций. Модуль имеет единый приемный буфер токенов, из которого они распределяются далее по процессорам сопоставлений.

Процессор сопоставления (ПС) предназначен для сравнения токенов, поступающих на его вход, с токенами, хранящимися в его памяти. Сопоставление и хранение токенов наиболее полно реализуется через аппаратную ассоциативную память [9–11]. Исполнительное устройство (ИУ) представляет собой простой вычислитель, задачей которого является выполнение программного кода, указатель на который содержится в пакете, сформированном в результате сопоставления токенов в ПС.

Указатель на программный код (номер узла) является частью ключа сопоставления наряду с контекстом, который присутствует в каждом токене. Программный код представляет собой последовательность команд для обработки входных данных пакета и формирования новых токенов. Блок формирования хэш-функций (БХФ) представляет собой

специализированное устройство, предназначенное для расчета номера ПС, в который должен поступить каждый токен, сформированный в ИУ. Данный расчет осуществляется на основе ключа токена, состоящего из контекста и номера целевого узла. Подробное описание назначения и логики работы БХФ представлено в [**12**].

В целом, упрощенная схема работы ППВС выглядит следующим образом. Токен, поступающий на вход системы через коммуникационную сеть, направляется в определенный вычислительный модуль. В модуле он передается на вход конкретного процессора сопоставления, где осуществляется его сравнение со всеми токенами, хранящимися в памяти. Целевой вычислительный модуль и ПС в нем определяются из значения хеш-функции, сформированной в БХФ.

В случае положительного результата сравнения, формируется пакет данных, который передается на любое свободное исполнительное устройство для выполнения кода программного узла, номер которого указан в пакете. В противном случае токен сохраняется в памяти ПС.

Токены, сформированные в процессе выполнения программного узла, направляются на БХФ, где на основе их ключа определяется номер целевого процессора сопоставлений, в котором должен быть обработан этот токен. После этого токены передаются обратно в коммуникационную сеть для продолжения цикла обработки. Данный процесс повторяется до полного исчерпания всех активных (движущихся по коммуникационной сети или ожидающих обработки на входах вычислительных модулей) токенов в системе.

Поскольку каждый ПС располагает своей отдельной памятью токенов, то токены, подлежащие сопоставлению друг с другом, обязаны попасть в один и тот же ПС. Концептуально, ядром системы называется минимальная функционально-полная подструктура, состоящая из одного ПС и одного ИУ (на рисунке 1 обведена пунктиром).

Для корректной работы многоядерной системы имеет значение привязка ядра только к ПС, а какое именно ИУ будет дальше обрабатывать сформированный в ПС пакет – безразлично. Поскольку каждый ПС обрабатывает входящие в него токены последовательно, то параллелизм системы определяется именно числом ПС (их k в каждом модуле). Количество экземпляров ИУ в каждом модуле (оно обозначено как r) просто считается достаточным, чтобы не было дополнительных заторов. Поэтому ниже, говоря о ядрах и их количестве, имеется в виду общее количество ПС в системе (обозначенное на рисунке 1 как n).

Описанная организация вычислений ППВС существенно отличается от традиционной, что предполагает иной подход к разработке программ для

этой системы относительно традиционной императивной парадигмы. Особенности создания параллельных программ в рамках потоковой парадигмы программирования подробно рассмотрены в [13, 14].

2. Потоковый алгоритм прямого распространения

Нейронная сеть прямого распространения, или многослойный персептрон, состоит из одного входного слоя, одного выходного слоя и одного или нескольких скрытых слоев. Алгоритм прямого распространения сводится к вычислению (для каждого нейрона в каждом слое сети) значения функции активации от взвешенной суммы входов нейрона по формуле:

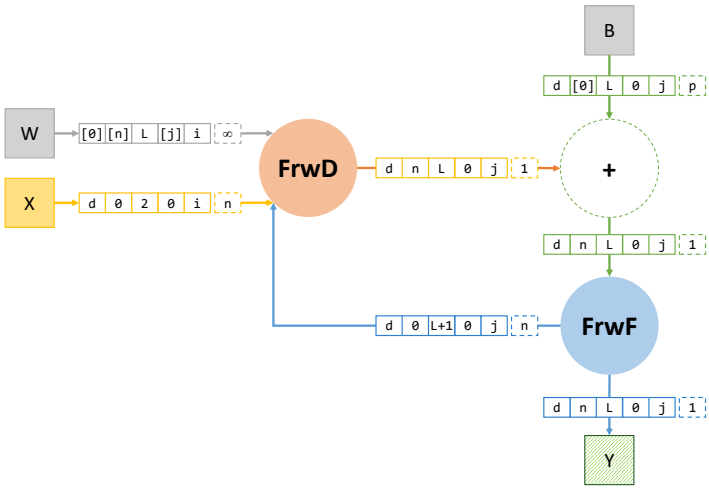
$$H_j^L = f\left(\sum_i w_{ji}^L \cdot H_i^{L-1} + b_j^L\right), \quad \text{где} \tag{1}$$

H – значение на выходе нейрона (при этом H_i^1 – значения нейронов входного слоя),
f – функция активации,
L – индекс слоя,
j – индекс нейрона в слое L,
i – индекс нейрона в слое $(L-1)$,
w – весовой коэффициент,
b – коэффициент смещения.

В формуле (1) можно выделить три основных компонента: произведение, сложение и вычисление функции. В рамках традиционной парадигмы программирования рассматриваемая задача может быть решена с использованием трех вложенных циклов, которые последовательно обходят все слои (цикл по L), все нейроны каждого слоя (цикл по j) и выполняют суммирование всех произведений на входах каждого нейрона (цикл по i) с последующим вычислением функции активации. При этом производятся обращения к значениям выходов предыдущего $(L-1)$ слоя, вычисленным на предыдущей итерации внешнего цикла.

Параллелизм задачи ограничен наличием зависимости между внешними итерациями. Однако, использование библиотечных функций для работы с матрицами и векторами, или технологий программирования графических ускорителей (GPGPU), позволяет достичь хорошей эффективности за счет распараллеливания двух внутренних циклов.

В отличие от традиционных последовательных и параллельных программ, потоковая программа распределена на три в равной степени важные части: программный код задачи (представлен на рисунке 2 графом 2*а* и листингом на языке DFL 2*б*), алгоритм формирования начальных данных (с листингом 2*в*) и функция распределения вычислений.



(*а*) потоковый граф

```
module ForwardPropagation;

type F1 = {d: byte, n: byte, L: byte, j: byte, i: byte};

node FrwD(a, b : double) : F1; dgrouped;
begin
    send <fsum> a*b to FrwF.a {d,n,L,0,j};
end;

node FrwF (a : double) : F1;
var ReLU : double;
begin
    if a<0 then ReLU:=0;
           else ReLU:=a;
    end;

    if n=0 then send ReLU to host;
           else send ReLU to FrwD.b {d,0,L+1,0,i} <<n>>;
    end;
end;

endmod.
```

(*б*) программный код на языке DFL

```
// D                          - количество образцов
// N                          - количество слоев (включая входное и выходное)
// L[1..N+1]                  - массив нейронов по слоям. L[N+1] всегда равно 0.
// X[1..D][1..L[1]]           - матрица входных данных всех образцов
// W[2..N][1..L[1]][1..L[1-1]] - матрица весовых коэффициентов  [слой][нейрон][вход]
// B[2..N][1..L[B]]           - матрица коэффициентов смещения [слой][нейрон]

//Входные данные
for d:=1 to D do
  for i:=1 to X.size do send X[i] to FrwD.b {d,0,2,0,i} <<L[2]>>;

//Весовые коэффициенты
for l:=2 to L.size-1 do
  for j:=1 to L[1] do
    for i:=1 to L[1-1] do
      send W[1][j][i] to FrwD.a {[0],[L[1+1]],1,[j],i} <<inf>>;

//Коэффициенты смещения и финализация аппаратного суммирования
for d:=1 to D do
  for l:=2 to L.size-1 do
    for j:=1 to L[1] do
      send <fsum> B[1][j] to FrwF {d,[0],1,0,j} <<L[1-1]>>;
```

(*в*) псевдокод формирования начальных данных

Рисунок 2. Алгоритм прямого распространения

Алгоритм состоит из двух программных узлов, использует аппаратные возможности ППВС «Буран» и поддерживает любую структуру многослойного персептрона, размер которого ограничивается разрядностью используемых полей контекста.

2.1. Графическое представление алгоритма

На графическом представлении алгоритма (рисунок 2*а*) стрелками показаны направленные связи, по которым происходит передача данных (в форме токенов) от начальных данных, между программными узлами и к выходным данным. Данная форма визуализации применяется как для

первоначального формирования алгоритма, так и для его наглядного представления, чему способствует небольшой набор правил и форм, представленных на графе.

Все данные, включая начальные и выходные, обозначаются в виде закрашенных квадратов с указанием названия в центральной части (квадраты `W`, `X`, `B` и `Y`). Программные узлы представляются в виде закрашенных окружностей с указанием названия узла в центральной части (`FrwD` и `FrwF`). Специальные функции, реализованные в процессоре сопоставлений, обозначаются белыми кругами с пунктирными границами («+» символизирует аппаратное суммирование). Специальные функции «прозрачны» в том смысле, что посылаемые на них токены в языке адресуются на следующий за ним узел, который активируется по завершению суммирования.

Потоки данных между объектами отображаются посредством линий со стрелками, на которых отмечены контекст и кратность токенов. Контекст токена представляет собой набор полей, значения которых указаны в клеточках со сплошной границей на линии связи. Значение поля контекста, взятое в квадратные скобки, указывает на то, что данное поле замаскировано, т.е. оно игнорируется при сопоставлении двух токенов. Значение кратности токена указывается справа от полей контекста в отдельном квадрате с пунктирной границей.

С учетом описанных правил, представленный на рисунке 2*а* алгоритм прямого распространения следует интерпретировать следующим образом:

1. Потоки данных `W` (весовые коэффициенты) и `X` (значения тестовых образцов) поступают в узел `FrwD`, где осуществляется их перемножение.
2. Результат умножения передается в узел `FrwF` посредством аппаратного механизма суммирования, представленного в виде узла «+», дополнительным аргументом которого служит поток данных `B` (число слагаемых аппаратного суммирования формально передается в позиции кратности, а значение содержит дополнительное слагаемое – здесь это bias, коэффициент смещения).
3. Итоговое значение, полученное в результате сложения указанного числа произведений и коэффициента смещения, инициирует срабатывание узла `FrwF`, в котором производится вычисление функции активации и проверка условий завершения алгоритма.
4. Результат вычисления функции активации передается в качестве второго множителя в узел `FrwD` для последующего взаимодействия с весовыми коэффициентами на входах нейронов следующего слоя.

5. Окончательным результатом работы алгоритма (поток данных `Y`) является вычисление функции активации для нейронов последнего слоя.

Ключевым элементом приведенного алгоритма является структура контекста формируемых токенов. Значения полей контекста непосредственно влияют на сопоставление токенов и формирование пакетов, активирующих программные узлы. Таким образом организация контекста токена (разбиение контекста на поля и логика заполнения этих полей значениями) представляет собой наиболее значимый и сложный аспект потокового программирования, требующий применения нелинейного мышления и глубокого понимания реализуемого алгоритма.

Одним из подходов к определению формата контекста является анализ точек пересечения потоков данных в алгоритме. Первой такой точкой пересечения данных является узел `FrwD`, для активации которого требуется, чтобы контексты весовых коэффициентов и данных указывали на один и тот же вход определенного нейрона в конкретном слое (индексация по i, j и L согласно формуле). Второй точкой пересечения потоков данных является активация узла `FrwF` посредством аппаратного суммирования произведений и коэффициента смещения, относящихся к определенному нейрону конкретного слоя (индексация по j и L согласно формуле).

Таким образом, основа формата контекста полностью повторяет индексацию, представленную в формуле (1), а именно, комбинация значений полей `i`, `j` и `L` однозначно ассоциирует данное с i-м входом j-го нейрона в слое `L`. В дополнение к основным полям необходимо реализовать идентификацию данных, относящихся к различным тестовым образцам. Для этого в контекст было добавлено поле `d`.

Кроме того, для обеспечения корректной логики определения завершения вычислений, реализованной в узле `FrwF`, в контекст добавлено поле `n`, содержащее количество нейронов в определенном слое. В итоге формат контекста, являющийся общим для всех токенов задачи, включает пять полей: `d` (индекс тестового образца), `n` (количество нейронов в слое), `L` (индекс слоя), `j` (индекс нейрона) и `i` (номер входа).

Еще одним ключевым элементом потокового программирования является согласование начальных данных с программным кодом, заключающееся в обеспечении соответствия формата контекста и назначения (название программного узла и входа) данных создаваемых токенов. На рисунке 2*в* представлен псевдокод формирования тестовых образцов, весовых коэффициентов и коэффициентов смещения, формат контекста которых полностью совпадает с общим контекстом всех токенов задачи.

2.2. Описание программного кода алгоритма на языке высокого уровня

Для представления алгоритма в форме программного кода используется оригинальный язык высокого уровня DFL, специально разработанный в рамках проекта ППВС «Буран» для создания и исполнения на ППВС потоковых программ.

Графическое представление алгоритма (рисунок 2*а*) используется для облегчения понимания алгоритма и его последующая трансформация в программный код на языке высокого уровня DFL (рисунок 2*б*), являющимся точным выражением алгоритма, значительно упрощена благодаря формализации входных и выходных аргументов программных узлов, а также общему формату токенов. Единственной сложностью на этом этапе является определение параметров и типа взаимодействия формируемых токенов.

Описание программного узла состоит из заголовка и последовательного кода. Заголовок узла содержит имя узла, перечисление имен и типов данных входных аргументов, формат контекста и признак. Код узла – это последовательность команд обработки данных (аргументов узла, констант и литералов) и контекста, команд перехода (в пределах узла) и специальной команды формирования токенов.

Описание программного узла `FrwD`, отвечающего за перемножение весовых коэффициентов и данных тестовых образцов, расшифровывается следующим образом. Программный узел с именем `FrwD` содержит два вещественных аргумента a и b с контекстом формата `F1` (формат описан в начале кода программы после ключевого слова `type`) и имеет признак `dgrouped`. Код узла состоит из единственной команды формирования токенов, в которой результат перемножения входных аргументов a и b посылается на вход `a` узла `FrwF` с признаком `fsum` и заданным в фигурных скобках контекстом.

В данном коротком описании программного узла обращают на себя внимание такие элементы языка программирования как признак узла и формируемого токена, а также формат контекста и его формирование. Все эти элементы потокового программирования используются для отображения системы команд процессора сопоставлений, отвечающего за взаимодействие токенов и формирование пакетов.

Признак `dgrouped` (двойной групповой) обозначает, что токены, посылаемые на узел `FrwD`, взаимодействуют в соответствии с правилом «каждый с каждым». Это правило предполагает, что для каждой пары сопоставившихся токенов формируется один и только один пакет. В рамках реализуемого алгоритма токены этого типа используются для

оптимизации процесса взаимодействия одного значения из тестового образца с множеством весовых коэффициентов на определенном входе всех нейронов слоя.

В случае отсутствия признака `dgrouped` – пара сопоставившихся токенов будет порождать пакеты до исчерпания кратности одного из них, но это противоречит логике решения задачи. В текущей реализации (см. рисунок 2*в*) формируется только один набор значений для каждого тестового образца (раздел «Входные данные») и один набор весовых коэффициентов (раздел «Весовые коэффициенты») для всей задачи. При этом для токенов со значениями тестовых образцов задается кратность, соответствующая количеству нейронов во втором слое.

У токенов со значениями весовых коэффициентов маскируются все поля, кроме `L` (номер слоя) и `i` (номер входа), и задается бесконечная кратность. Поскольку маскирование поля исключает его из сравнения при сопоставлении контекстов токенов, положительное взаимодействие фиксируется только для токенов с идентичными значениями полей `L` и `i`. Таким образом, каждое входное значение из тестового образца формирует несколько пакетов (по числу нейронов во втором слое), содержащих все соответствующие ему весовые коэффициенты.

Признак `fsum` в команде `send` указывает на формирование специального токена «вещественное суммирование». При положительном взаимодействии двух токенов этого типа в процессоре сопоставлений, вместо генерации пакета, значение «нижнего» токена (хранящегося в памяти) увеличивается на значение «верхнего» токена (инициировавшего процесс сопоставления), после чего дальнейшее сопоставление прекращается. Формирование пакета на основе специального токена «суммирование» происходит после определенного числа взаимодействий, задаваемого в кратности «финализирующего» токена, посылаемого на второй вход вместе с дополнительным слагаемым – значением коэффициента смещения (поток данных `B` на рисунке 2).

Сформированный пакет, содержащий накопленную сумму, активирует программный узел `FrwF`. В текущей реализации алгоритма для суммирования всех произведений, относящихся к определенному нейрону, контекст однократного токена, формируемого в узле `FrwD`, составляется таким образом, что остаются неизменными значения полей `d` (для локализации вычислений в рамках заданного тестового образца), `n` (для сквозной передачи в узел `FrwF` значения числа нейронов в следующем слое, на основании которого принимается решение о завершении обработки тестового образца) и `L` (для сохранения принадлежности нейрона к заданному слою). При этом значение номера нейрона переносится в поле `i` с целью оптимизации распределения вычислений, осуществляемого по полям `L` и `i`.

Контекст «финализирующего» токена со значением коэффициента смещения (рисунок 2*в*) составляется в схожей логике, за небольшим исключением. Во-первых, в поле `n` маскируется нулевое значение для обеспечения сквозной передачи в узел `FrwF` условий завершения вычислений (при дизъюнкции контекстов сохраняется значение поля `n` контекста токена, формируемого в узле `FrwD`). Во-вторых, кратность (значение `p`) – количество слагаемых аппаратного суммирования, устанавливается равной числу входов нейрона, соответствующему числу нейронов в предыдущем $(L-1)$ слое. В результате взаимодействия токенов «вещественное суммирование» узел `FrwF` активируется для обработки данных, содержащих накопленную сумму произведений, относящихся к определенному (j-му) нейрону конкретного слоя.

В программном узле `FrwF` реализованы расчет функции активации и контроль завершения работы программы. Описание узла, представленное на рисунке 2*б*, расшифровывается следующим образом. Программный узел с именем `FrwF` содержит один вещественный аргумент `a` с контекстом формата `F1`. Код узла состоит из объявления внутренней вещественной переменной `ReLU`, предназначенной для временного хранения результата вычисления функции активации, а также двух команд ветвления.

Первая команда ветвления определяет значение функции активации и сохраняет его в переменную `ReLU`. Вторая команда ветвления осуществляет контроль завершения программы и, в зависимости от результата, определяет направление передачи результата функции активации.

Контроль завершения программы осуществляется путем анализа значения поля `n`, устанавливаемого на этапе формирования весовых коэффициентов (рисунок 2*в*). Данное значение сохраняется без изменений до поступления в узел `FrwF`, что обеспечивается посредством механизма сквозной передачи, реализуемого через дизъюнкцию контекстов при формировании пакетов.

Поскольку значение поля `n` для весовых коэффициентов, относящихся к нейронам последнего слоя, задается равным нулю, то при положительной проверке соответствующего поля на нуль, формируется токен, в котором результат функции активации, без изменения контекста, посылается на ХОСТ-машину (используется ключевое слово `host` вместо названия целевого узла). В противном случае формируется токен, в котором результат функции активации посылается на вход `b` узла `FrwD` с измененным контекстом и кратностью n.

Кратность токена определяет максимальное количество пакетов, которое может быть создано токеном в результате положительных взаимодействий с другими токенами. В соответствии с выбранным

форматом контекста, поле `n` содержит информацию о количестве нейронов в следующем $(L+1)$-м слое. Это значение указывает на количество нейронов, использующих результат функции активации текущего нейрона, и именно поэтому оно задается в качестве кратности выходного токена.

Контекст токена составляется на основе той же логики, что и контекст начальных данных тестового образца (поток данных `X`), за исключением того, что значение функции активации должно взаимодействовать с весовыми коэффициентами на входе нейронов, относящихся к следующему слою. В связи с этим, значение поля `L` увеличивается на единицу, а значение номера нейрона, перенесенное на предыдущем этапе в поле `i` в целях оптимизации распределения вычислений, сохраняется неизменным в поле `i`, указывая на то, что номер нейрона текущего слоя соответствует номеру входа для всех нейронов следующего слоя.

2.3. Выбор функции распределения вычислений

Завершающим этапом реализации потоковой программы является выбор функции распределения. Функция распределения представляет собой аппаратную реализацию формулы, аргументом которой является контекст токена, а результатом — номер процессора сопоставлений, на вход которого должен поступить данный токен. Эта функция выполняется для каждого нового токена, включая начальные, и ее выбор существенно влияет на общую эффективность выполнения программы.

Выбор функции распределения осуществляется из трех доступных базовых: стандартная, «по полю» и блочная. Стандартная (`STD`) функция обеспечивает равномерное распределение токенов, присваивая каждому последующему по значению контекста токену номер следующего по индексу процессора сопоставлений. Функция распределения «по полю» (`FLD`) группирует токены с определенным количеством подряд идущих значений выбранного поля в один процессор сопоставлений. Блочная (`ZIP`) функция распределяет токены по процессорам сопоставлений блоками, составленными из значений двух, трех или четырех выбранных полей.

Основываясь на структуре общего контекста задачи и ориентируясь на потоковый граф алгоритма (см. рисунок 2*а*), при выборе функции распределения необходимо руководствоваться следующими принципами. Поскольку все функции распределения учитывают исключительно незамаскированные значения полей, распределение потоков данных на программный узел `FrwD` возможно только по полям `L` и `i`, а потоков данных на узел `FrwF` – по всем полям, за исключением поля `d`. Объединив эти ограничения, можно констатировать, что поля `L` и `i` являются единственными подходящими для распределения всех токенов задачи.

Определившись с набором полей, необходимо провести логическую оценку эффективности каждой из доступных функций распределения.

В функции `FLD` используется одно из полей контекста для расчета номера целевого процессора сопоставлений. Независимо от выбранного поля («номер слоя» или «номер входа нейрона»), равномерность распределения напрямую зависит от максимального значения в этом поле: чем больше максимальное значение поля, тем больше токенов с близкими значениями будут направляться в одно вычислительное ядро, а чем меньше размерность поля, тем более неравномерным будет распределение данных (вычислений) по вычислительным ядрам. Более того, если количество вычислительных ядер превышает максимальное значение в выбранном поле, то часть ядер может оставаться незадействованной, что приводит к их простою.

Функция `ZIP` имеет аналогичную проблему распределения, но она использует два поля для расчета, и в одно и тоже вычислительное ядро направляются токены, чьи комбинации значений полей образуют блок в виртуальном адресном пространстве задачи, который (блок) строится на основе выбранных для распределения полей.

Функции `FLD` и `ZIP` обладают свойством локализации данных «порциями», что является преимуществом в задачах с фиксированной размерностью полей, достигаемое за счет снижения пересылок токенов между вычислительными ядрами. Однако в задачах с вариативным заполнением полей, таких как реализуемая задача прямого распространения с различным числом нейронов в слоях, это свойство приводит к неравномерному распределению данных, что негативно сказывается на эффективности работы вычислительной системы при решении задачи.

Функция `STD` лишена указанных преимуществ и недостатков. Ее отличительной чертой является равномерное распределение токенов, которое достигается за счет генерации следующего по порядку номера вычислительного ядра при обработке следующего по значению контекста. Таким образом, выбор стандартной функции для распределения всех токенов задачи по полям `L` и `i` контекста обеспечивает равномерное распределение токенов по вычислительным ядрам, что способствует повышению эффективности работы системы при решении задачи.

2.4. Оценка программного кода

В заключение описания потоковой программы необходимо отметить следующие аспекты:

(*1*) Программный код не зависит от структуры нейронной сети (число слоев и нейронов в каждом слое персептрона), поскольку она

(структура нейросети) определяется токенами весовых коэффициентов при инициализации данных на XOCT-машине.

(2) В программном коде отсутствуют вызовы библиотек функций, которые часто применяются при разработке параллельных алгоритмов для традиционных вычислительных систем.

(3) Потоковый алгоритм прямого распространения представляет собой прямую реализацию формулы расчета функции активации, при этом сам программный код отличается простотой и компактностью.

При сравнении представленного программного кода с кодом, реализующим описание нейронных сетей в традиционных фреймворках, таких как TensorFlow и PyTorch, можно выделить следующие различия. В широко используемых традиционных фреймворках каждый слой нейронной сети, как правило, определяется посредством одного библиотечного вызова, включающего множество параметров, таких как количество нейронов, количество входов, тип функции активации и другие.

В представленном алгоритме этот библиотечный вызов заменен парой узлов `FrwD` и `FrwF`, которые выполняют умножение на весовые коэффициенты, сложение произведений и применение заданной функции активации. В отличие от традиционных фреймворков эти программные узлы не содержат вызовов библиотечных функций и являются регулярными элементами языка DFL с четко определенной семантикой. В данном примере все слои персептрона имеют однотипную структуру и могут иметь единое описание, параметризованное номером слоя.

3. Исследование работы алгоритма прямого распространения на модели ППВС

Переходя к экспериментальной части представленного исследования, необходимо подчеркнуть, что основной целью проведения экспериментов являлась проверка функциональности разработанного потокового нейросетевого алгоритма и анализ его поведения на вычислительной системе с управлением потоком данными. Задача оценки эффективности алгоритма в контексте решения конкретной задачи не ставилась, что обусловлено отсутствием аппаратной реализации ППВС и значительным разнообразием существующих высокоскоростных аппаратных и программных решений для нейросетевых задач. Для проведения экспериментов была выбрана следующая методика.

Серия экспериментов выполнялась на поведенческой блочно-регистровой модели (ПБРМ) [**15**], предназначенной для моделирования поведения ППВС при выполнении потоковых программ. В рамках данной модели исследовалось поведение системы при выполнении нейросетевого

алгоритма для различных структур персептрона и при различных объемах входных данных.

Каждый эксперимент был проведен для архитектур с 1-м, 2-мя, 4-мя, 8-ю, 16-ю, 32-мя, 64-мя и 128-ю вычислительными ядрами, каждое из которых состоит из одного процессора сопоставлений и одного исполнительного устройства. Временные параметры моделируемой архитектуры оставались неизменными для всех экспериментов.

Для обеспечения чистоты результатов и исключения влияния второстепенных факторов, при проведении моделирования не учитывалось время формирования и ввода начальных данных в систему. Однако время передачи этих данных через коммуникационную сеть было учтено, так как данный процесс является неотъемлемой частью вычислительного процесса.

В рамках исследования были выбраны и проанализированы три различные структуры персептрона, охватывающие широкий спектр стандартных конфигураций многослойных нейронных сетей, которые здесь обозначены так:

равномерная состоит из 20 слоев, каждый из которых содержит 20 нейронов.

нисходящая также состоит из 20 слоев, однако количество нейронов в каждом слое уменьшается на один по мере продвижения от входного к выходному слою. Таким образом, входной слой содержит 20 нейронов, а выходной — один нейрон.

нисходяще-восходящая объединяет элементы двух предыдущих структур. Первые 10 слоев следуют нисходящей схеме, уменьшая количество нейронов на один элемент, после чего структура переходит к восходящей схеме, увеличивая количество нейронов на один элемент в каждом последующем слое, начиная с 12-го и заканчивая выходным слоем, который также содержит 20 нейронов.

Объем данных, обрабатываемых программой, определяется не только структурой персептрона, включающей количество весовых коэффициентов и нейронов во входном слое, но и числом тестовых образцов, которые последовательно вводятся в систему. В рамках проведенных экспериментов использовались выборки, состоящие из 1, 10 и 100 образцов, поступающих на вход системы без промежуточных задержек.

Каждый образец содержал данные нейронов входного слоя и значения коэффициентов смещения всех слоев. В токенах образцов и коэффициентов смещения в поле `d` их контекста задавался порядковый номер образца. А поле `d` контекста весовых коэффициентов маскировалось для обеспечения их взаимодействия с токенами всех образцов.

В рамках исследования поведения архитектуры ППВС при решении задачи прямого распространения проведено три серии экспериментов, каждая из которых включала 24 эксперимента для каждого типа структуры персептрона. В таблице 1 представлены числовые результаты проведенных экспериментов.

Таблица 1. Время (в условных тактах) выполнения задачи прямого распространения для персептронов с различной структурой

	Архитектура ППВС, количество вычислительных ядер							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Персептрон с равномерной структурой								
1 образец	146,769	73,345	37,286	24,889	21,480	16,145	18,475	18,091
10 образцов	1,191,129	595,525	297,759	160,689	91,341	58,998	29,701	27,773
100 образцов	11,622,889	5,811,657	2,906,261	1,532,762	850,388	540,256	233,978	230,738
Персептрон с «нисходящей» структурой								
1 образец	52,679	27,787	16,864	13,805	14,489	12,921	15,617	15,043
10 образцов	430,484	222,944	112,732	61,330	36,129	31,330	19,415	18,555
100 образцов	4,208,864	2,179,724	1,096,930	576,982	315,947	239,150	112,196	111,110
Персептрон с «нисходяще-восходящей» структурой								
1 образец	89,504	45,938	25,198	18,998	18,640	14,747	17,220	16,540
10 образцов	728,018	368,219	189,226	105,618	57,055	45,614	23,670	22,100
100 образцов	7,114,468	3,584,420	1,838,879	1,029,624	550,260	374,459	178,288	178,749

Анализ полученных результатов демонстрирует постепенное уменьшение времени выполнения задачи при увеличении вычислительных ресурсов. Эффективность (отношение ускорения к числу ядер) наиболее высока при использовании небольшого количества ядер и большого объема данных (количества пакетов), и снижается по мере достижения «избыточности» в вычислительных ресурсах. Схожее поведение ППВС, при котором при достаточном объеме данных и правильно выбранной функции распределения достигается практически линейное ускорение, неоднократно подтверждалось на задачах различных классов[16, 17].

Следует отметить значение, указывающее на замедление выполнения одного образца на 64-х ядрах. Отмеченная аномалия обусловлена тем, что количество вычислительных ядер достигло уровня, при котором объем полезных вычислений на одном ядре уже недостаточен для сокрытия задержек при передаче данных через коммуникационную среду. Этот эффект полностью компенсируется увеличением объема данных.

Более значимыми результатами проведенных серий экспериментов является оценка масштабируемости и аппаратной конвейеризации задачи. Масштабируемость задачи определяется по тому, при каком уровне увеличения вычислительных ресурсов перестает наблюдаться пропорциональный рост ускорения времени выполнения одной и той же задачи.

На рисунке 3 представлены графики эффективности выполнения задачи по сравнению с одноядерным запуском одного образца. Графики показывают изменение времени выполнения одного образца из набора относительно времени выполнения одиночного образца на одном ядре (2), что позволяет оценить среднюю эффективность одного ядра при увеличении объема данных и числа ядер.

(2) $$E_N^C = \frac{T_1^1}{T_N^C} * \frac{N}{C}, \quad \text{где}$$

N – число образцов в выборке,

C – число вычислительных ядер,

T_N^C – время выполнения задачи с N образцами на C вычислительных ядрах,

E_N^C – эффективность расчета N образцов на C ядрах относительно одноядерного запуска одного образца.

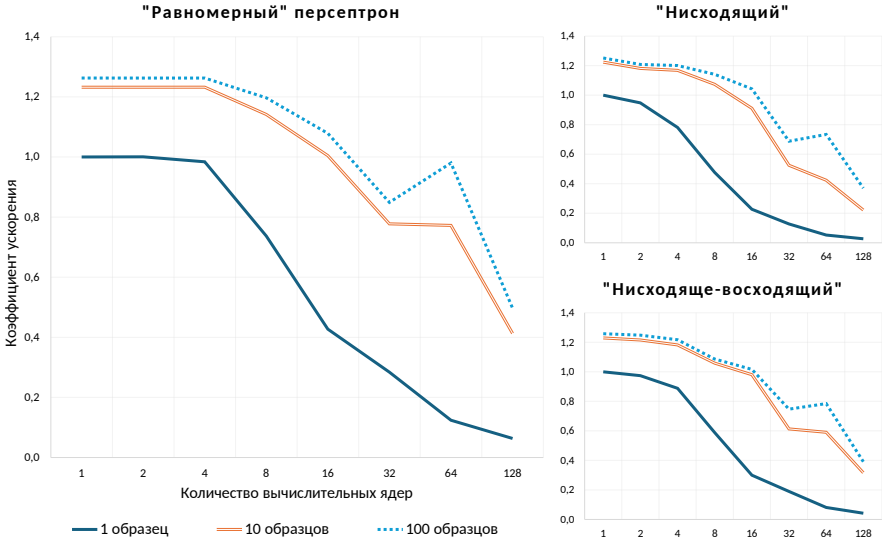


Рисунок 3. Эффективность выполнения задачи прямого распространения на одном ядре ППВС с разным числом ядер для различных структур многослойного персептрона и различных размеров пакетов

Анализ представленных графиков позволяет сделать следующие выводы:

(*1*) Размерность задачи, определяемая числом нейронов в каждом слое, оказывает прямое влияние на масштабируемость. Для «равномерного» персептрона, в котором количество нейронов остается неизменным во всех слоях, эффективность увеличения вычислительных ресурсов начинает снижаться при 8 ядрах (когда на одно ядро приходится менее трех нейронов на слой). В случае «нисходящего» персептрона с неравномерным распределением нейронов и наименьшим суммарным числом нейронов, снижение эффективности наблюдается уже после 2 ядер.

Полученное снижение является вполне ожидаемым и объясняется нехваткой параллельной работы внутри каждого слоя для заданного числа ядер. Это позволяет констатировать, что повышение числа слоев и увеличение количества нейронов в каждом слое позволяет задействовать больше вычислительных ресурсов без потери их эффективности.

(*2*) Поступление в систему новых образцов до завершения обработки предыдущих образцов не только увеличивает масштабируемость задачи («избыточность» вычислительных ресурсов проявляется на значительно большем числе ядер), но и приводит к аппаратной конвейеризации вычислений на каждом отдельном вычислительном ядре. Согласно соотношению линий на графиках, коэффициент конвейеризации составляет приблизительно 25%, что означает, что каждое вычислительное ядро способно выполнить в 1.25 раза больше задач прямого распространения при непрерывной подаче новых данных.

Описанный эффект конвейеризации обусловлен структурой вычислительного ядра и этапами его работы (прием и обработка токенов в процессоре сопоставлений, формирование пакета и его последующая обработка в исполнительном устройстве).

Коэффициент конвейеризации, определенный на основании данных, представленных на рисунке 3, имеет ограниченную область применения. Его значение является релевантным исключительно в контексте одноядерных систем. Преимуществом вычислительных систем с управлением потоком данных является их способность эффективно использовать большое количество ядер, которые параллельно и автономно выполняют свою часть задачи.

Поэтому более корректным подходом к оценке конвейеризации является анализ системы (ППВС) в целом. Такая оценка может быть получена посредством сравнения результатов выполнения задачи при различном количестве образцов в пакете («пакетный» запуск). На рисунке 4 представлены графики, показывающие отношение времени выполнения одиночного образца к времени выполнения одного образца из пакета (3),

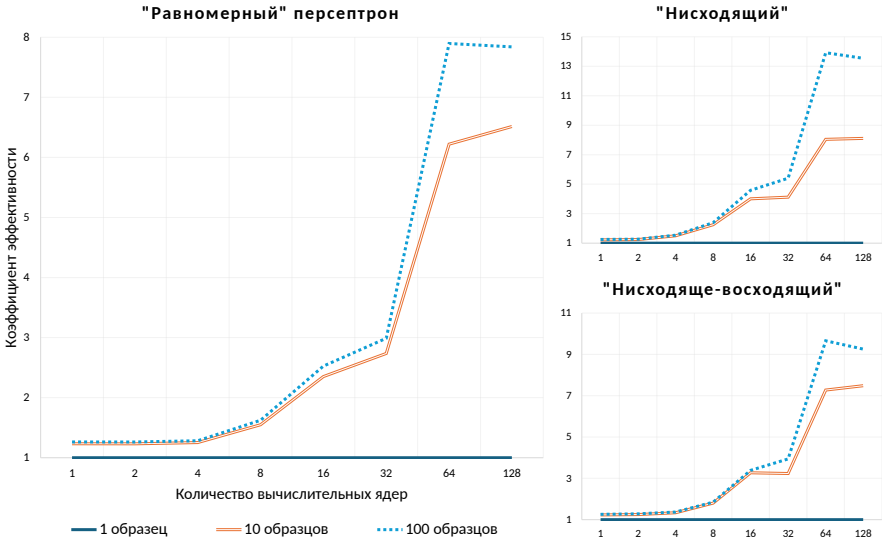


Рисунок 4. Эффективность «пакетного» выполнения задачи прямого распространения на ППВС для различных структур многослойного персептрона

что позволяет оценить динамику ускорения «пакетного» выполнения задачи по сравнению с эталонным увеличением времени выполнения задачи.

$$U_N^C = \frac{N * T_1^C}{T_N^C}, \quad \text{где} \tag{3}$$

N – число образцов в выборке,
C – число вычислительных ядер,
T_N^C – время выполнения задачи с N образцами на C вычислительных ядрах,
U_N^C – удельное ускорение времени расчета N образцов на C ядрах.

«Пакетное» выполнение задачи предполагает «непрерывную» подачу нескольких образцов, при этом каждый следующий образец поступает на вход системы без временных задержек после предыдущего и без ожидания завершения обработки предыдущего образца. Эталонное время выполнения задачи определяется как произведение времени обработки одного образца на число образцов.

Ускорение относительно эталонного времени выполнения задачи определяется как отношение времени выполнения задачи с N образцами к

суммарному времени выполнения N задач с одним образцом в каждой. Анализ представленных графиков демонстрирует следующее:

- Результаты полностью коррелируют с оценкой масштабируемости задачи (см. рисунок 3).
- До достижения точки «избыточности» вычислительных ресурсов, характеризующейся наличием достаточного объема данных для загрузки имеющихся вычислительных ресурсов, относительное ускорение составляет приблизительно 1.25, что указывает на ранее определенный коэффициент конвейеризации.
- После достижения точки «избыточности», когда накладные расходы начинают превышать полезные вычисления, добавление дополнительных образцов обеспечивает необходимую загрузку ресурсов. Это приводит к результатам, представленным на графиках: кратное увеличение объема данных вызывает непропорциональное (в несколько раз меньшее) увеличение времени выполнения задачи (для примера см. таблицу 1, где увеличение числа образцов в 100 раз увеличивает время выполнения задачи на 128-ми ядрах не в 100 раз, а всего примерно в 13 раз).

В результате проведенных экспериментов можно сделать следующие выводы. Эффективность задачи прямого распространения значительно повышается в системах с управлением потоком данных, включая ППВС, при увеличении количества слоев и нейронов в структуре нейронной сети, а также при увеличении числа тестовых образцов в наборе данных, используемых для вычислений. Наибольшая эффективность достигается при обеспечении непрерывного потока тестовых образцов.

Заключение

На сегодняшний день уровень внедрения вычислительных систем с управлением потоком данных можно охарактеризовать как крайне низкий. Большинство таких систем используются в качестве аппаратных ускорителей и выполняют вспомогательные функции. Проекты вычислительных систем, полностью реализующих потоковую модель вычислений, представлены единичными примерами, одним из которых является ППВС «Буран».

В статье представлено детальное описание потокового алгоритма прямого распространения. Потоковая программа, реализующая данный алгоритм, отличается высокой степенью компактности и универсальности. Программа в своей основе является параллельной и не имеет ограничений на используемые вычислительные ресурсы, автоматически масштабируясь на всю систему. Параллелизм программы обеспечивается на уровне нейронной сети (нейронов и слоев), в отличие от систем, основанных на

традиционной модели вычислений, где параллелизм достигается на уровне тестовых образцов.

Код программы не содержит ссылок на библиотечные функции и основывается исключительно на базовых арифметических операциях, таких как сложение, умножение и сравнение. Максимальный размер персептрона, обрабатываемого программой, определяется исключительно разрядностью полей контекста. Это единственный аспект задачи, который может потребовать модификации и перекомпиляции программного кода. В остальных случаях программа способна функционировать с любым объемом данных без необходимости внесения изменений в исходный код.

Экспериментальная часть статьи показала высокую адаптивность потоковой программы к обработке больших объемов непрерывно поступающих данных, что указывает на потенциал применения потоковых вычислительных систем в задачах нейросетевой обработки, включая обработку потоков фото- и видеоданных. Результаты экспериментов также подтвердили наличие в архитектуре ППВС «эффекта конвейеризации» вычислений внутри каждого ядра.

Проявившийся эффект позволяет аппаратно извлекать дополнительный скрытый параллелизм, что, в свою очередь, обеспечивает ускорение по сравнению с традиционным параллельным выполнением нескольких задач на нескольких ядрах. Другими словами, одно ядро ППВС способно выполнить N однотипных подзадач за время, которое превышает время выполнения одной подзадачи менее чем в N раз.

На данном этапе исследований не представляется возможным провести сравнительный анализ эффективности выполнения реализованной потоковой программы с использованием существующих аппаратных платформ, таких как графические процессоры (GPU) или программируемые логические матрицы (FPGA), поскольку ППВС «Буран» на данный момент существует исключительно в виде программной модели, которая позволяет оценить время выполнения потоковых алгоритмов в условных тактах. Содержание результатов, полученных на программной модели, главным образом заключается в их внутренних свойствах, таких как масштабируемость и извлекаемый параллелизм. Содержание же статьи в большей степени нацелено на демонстрацию написания алгоритма и механизма его работы.

Для оценки применимости потоковой модели вычислений в более широком спектре нейросетевых задач, а также для изучения перспектив ее использования в процессах обучения нейросетевых моделей, требуется проведение дополнительных исследований. Так, в рамках представленного исследования не был рассмотрен вопрос ресурсных ограничений ассоциативной памяти ключей (АПК). В приведенных экспериментах все

совместно обрабатываемые данные загружались в систему таким образом, что это быстро приводило к исчерпанию объема АПК.

В статье [18] эта проблема была рассмотрена в контексте задачи умножения матриц, и было предложено комбинированное решение, включающее использование на входе процессора сопоставлений обычной прямо-адресуемой памяти большого объема, в которой хранятся токены с заданным приоритетом, определяющим очередность подачи на обработку. В контексте рассматриваемой задачи при определении приоритета на основе «номера тестового образца» рассмотренный подход позволит обеспечить поэтапное введение новых образцов в систему по мере завершения обработки введенных раньше. Это позволит поддерживать оптимальную загрузку АПК и вычислительных устройств. Описанный метод показал свою высокую эффективность на разных задачах в плане сокращения объема потребной АПК.

Список использованных источников

[1] Kotlar M., Babovic Z., Milutinovic V. *Implementation of perception algorithm using DataFlow paradigm // 2016 13th Symposium on Neural Networks and Applications (NEUREL)* (Belgrade, Serbia, 22–24 November 2016).– IEEE.– 2016.– ISBN 978-1-5090-1531-3.– С. 1–5. ↑52

[2] Milutinovic V., Kotlar M., Stojanovic M., Dundic I., Trifunovic N., Babovic Z. *Implementing Neural networks by using the DataFlow paradigm // DataFlow Supercomputing Essentials. Computer Communications and Networks*, Cham: Springer.– 2017.– ISBN 978-3-319-66124-7.– Pp. 3–44. ↑52

[3] *Machine Learning with Multiscale Dataflow Computing for High Energy Physics.*– Maxeler Technologies. Maximum Performance Computing.– 103 с. ↑52

[4] Böhm A. P. W. *Dataflow and hybrid dataflow architecture summary // Parallel Computer Systems: Performance Instrumentation and Visualization*, ред. R. Koskela, M. Simmons, New York: ACM.– 1990.– ISBN 978-0-201-50937-3.– С. 281–286. ↑54

[5] Arvind, Brobst S. *The evolution of dataflow architectures: from static dataflow to P-RISC* // International Journal of High Speed Computing.– 1993.– Т. **05**.– № 02.– С. 125–153. ↑54

[6] Lee B., Hurson A. R. *Dataflow architectures and multithreading* // Computer.– August 1994.– Т. **27**.– № 8.– С. 27–39. ↑54

[7] Климов А. В., Левченко Н. Н., Окунев А. С., Стемпковский А. Л. *Вопросы применения и реализации потоковой модели вычислений*, МЭС-2016 (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.), Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС).– №2, М.: ИППМ РАН.– 2016.– С. 100–106. ↑54

[8] Стемпковский А. Л., Левченко Н. Н., Окунев А. С., Цветков В. В. *Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов* // Информационные технологии.– 2008.– № 10.– С. 2–7. ↑54

[9] Smith K. C., Sedra A. S. *Associative memory* // *Encyclopedia of Computer Science*, Chichester: John Wiley and Sons Ltd.– 2003.– ISBN 978-0-470-86412-8.– С. 105–106. ↑55

[10] Soleimani P., Capson D. W., Li K. F. *A partitioned CAM architecture with FPGA acceleration for binary descriptor matching* // ACM Transactions on Reconfigurable Technology and Systems.– 2024.– Т. **17**.– № 1.– ид. 10.– 21 с. ↑55

[11] Sandhie Z. T., Ahmed F. U., Chowdhury M. H. *Ternary content addressable memory* // *Beyond Binary Memory Circuits*, Synthesis Lectures on Digital Circuits & Systems, Cham: Springer.– 2022.– ISBN 978-3-031-16194-0.– С. 87–95. ↑55

[12] Змеев Д. Н., Климов А. В., Левченко Н. Н. *Средства распределения вычислений в ППВС «Буран» и варианты реализации блока выработки хэш-функций*, МЭС-2016 (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.), Проблемы разработки перспективных микро- и наноэлектронных систем.– №2, М.: ИППМ РАН.– 2016.– С. 107–113. ↑56

[13] Змеев Д. Н., Левченко Н. Н. *Особенности создания параллельных программ в потоковой парадигме программирования* // Информационные технологии.– 2022.– Т. **28**.– № 11.– С. 597–606. ↑57

[14] Змеев Д. Н., Левченко Н. Н. *Особенности создания параллельных программ для параллельной потоковой вычислительной системы «Буран»* // Информационные технологии.– 2023.– Т. **29**.– № 10.– С. 529–539. ↑57

[15] Змеев Д. Н. *Средства проектирования высокопроизводительных потоковых вычислительных систем*, МЭС-2016 (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.), Проблемы разработки перспективных микро- и наноэлектронных систем.– №2, М.: ИППМ РАН.– 2016.– С. 159–163. ↑66

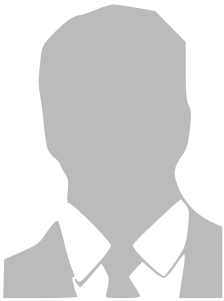
[16] Змеев Д. Н., Окунев А. С. *Разработка и исследование алгоритма задачи перемножения разреженных матриц для параллельной потоковой вычислительной системы «Буран»*, МЭС-2018 (Москва, Зеленоград, Россия, 1 октября–5 октября 2018 г.), Проблемы разработки перспективных микро- и наноэлектронных систем.– №3, М.: ИППМ РАН.– 2018.– С. 16–23. ↑68

[17] Стемпковский А. Л., Левченко Н. Н., Окунев А. С., Климов А. В., Змеев Д. Н. *Программирование задачи «молекулярная динамика» в потоковой модели вычислений* // Информационные технологии.– 2017.– Т. **23**.– № 12.– С. 859–867. ↑68

[18] Климов А. В., Левченко Н. Н., Окунев А. С., Стемпковский А. Л. *Суперкомпьютеры, иерархия памяти и потоковая модель вычислений* // Программные системы: теория и приложения.– 2014.– Т. **5**.– № 1(19).– С. 15–36. ↑74

Поступила в редакцию 15.07.2025;
одобрена после рецензирования 25.08.2025;
принята к публикации 25.08.2025;
опубликована онлайн 07.09.2025.

Информация об авторах:

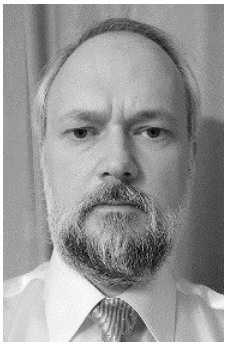


Дмитрий Николаевич Змеев

научный сотрудник Отделения проблем проектирования в микроэлектронике Центра перспективной микроэлектроники Национального исследовательского центра «Курчатовский институт». Научные интересы: вычислительные системы с управлением потоком данных, параллельные вычисления

0000-0001-5544-968X

e-mail: *zmejevdn@list.ru*



Николай Николаевич Левченко

к.т.н., ведущий научный сотрудник Отделения проблем проектирования в микроэлектронике Центра перспективной микроэлектроники Национального исследовательского центра «Курчатовский институт». Научные интересы: суперкомпьютерные архитектуры, параллельные вычисления, перспективные архитектуры вычислительных систем

0000-0003-3550-7381

e-mail: *nick@burcom.ru*

Аркадий Валентинович Климов

старший научный сотрудник Отделения проблем проектирования в микроэлектронике Центра перспективной микроэлектроники Национального исследовательского центра «Курчатовский институт». Научные интересы: нетрадиционные модели параллельных вычислений

0000-0002-7030-1517

e-mail: *arkady.klimov@gmail.com*

Вклад авторов: *Д. Н. Змеев* – 80% (программное обеспечение, проведение экспериментов, написание черновой версии, доработка и редактирование, визуализация); *Н. Н. Левченко* – 10% (доработка и редактирование, администрирование); *А. В. Климов* – 10% (валидация, доработка и редактирование).

Декларация об отсутствии личной заинтересованности: *благополучие авторов не зависит от результатов исследования.*

EN RU PROGRAM SYSTEMS: THEORY AND APPLICATIONS ISSN 2079-3316
Research Article ARTIFICIAL INTELLIGENCE, INTELLIGENCE SYSTEMS, NEURAL NETWORKS

UDC 004.021+004.272
10.25209/2079-3316-2025-16-4-51-79

Forward propagation algorithm in the dataflow paradigm

Dmitry Nikolayevich **Zmejev**[1], Nikolay Nikolayevich **Levchenko**[2], Arkady Valentinovich **Klimov**[3]

National Research Center "Kurchatov Institute", Moscow, Russia

[1]✉*zmejevdn@list.ru*, [2]✉*nick@burcom.ru*, [3]✉*arkady.klimov@gmail.com*

Abstract. The article discusses the issue of developing and implementing the forward propagation algorithm in the dataflow paradigm. The principles of dataflow computing differ significantly from traditional control-flow computing, just as the dataflow programming paradigm differs from the imperative one. Programs created in the dataflow paradigm are initially parallel. The article provides a detailed description of the forward propagation dataflow algorithm and the program that runs on the parallel dataflow computing system "Buran".

The dataflow program is compact and versatile in its code. It automatically scales to the entire system, and its program code does not contain references to library functions and is based solely on basic arithmetic operations such as addition, multiplication, and comparison. The program is capable of processing perceptrons of any dimension without modification and recompilation of the program code. The size and structure of the processed perceptron is determined by the initial data. The experimental part of the article demonstrates the high adaptability of the dataflow program to processing large amounts of continuously incoming data. An assessment of the applicability of the dataflow computing model for solving neural network problems is given. (*In Russian*).

Key words and phrases: forward propagation, parallel programming, dataflow computing model, dataflow programming paradigm, parallel dataflow computing system

2020 *Mathematics Subject Classification*: 68Q09; 68T01, 68W10

Acknowledgments: The work was carried out within the state assignment of NRC «Kurchatov institute»

For citation: Dmitry N. Zmejev, Nikolay N. Levchenko, Arkady V. Klimov. *Forward propagation algorithm in the dataflow paradigm*. Program Systems: Theory and Applications, 2025, **16**:4(67), pp. 51–79. (*In Russ.*). https://psta.psiras.ru/read/psta2025_4_51-79.pdf

References

[1] Kotlar M., Babovic Z., Milutinovic V.. "Implementation of perception algorithm using DataFlow paradigm", *2016 13th Symposium on Neural Networks and Applications (NEUREL)* (Belgrade, Serbia, 22–24 November 2016), IEEE, 2016, ISBN 978-1-5090-1531-3, pp. 1–5.

[2] Milutinovic V., Kotlar M., Stojanovic M., Dundic I., Trifunovic N., Babovic Z.. "Implementing Neural networks by using the DataFlow paradigm", *DataFlow Supercomputing Essentials. Computer Communications and Networks*, Springer, Cham, 2017, ISBN 978-3-319-66124-7, pp. 3–44.

[3] *Machine Learning with Multiscale Dataflow Computing for High Energy Physics*, Maxeler Technologies. Maximum Performance Computing, 103 pp.

[4] A. P. W. Böhm. "Dataflow and hybrid dataflow architecture summary", *Parallel Computer Systems: Performance Instrumentation and Visualization*, eds. R. Koskela, M. Simmons, ACM, New York, 1990, ISBN 978-0-201-50937-3, pp. 281–286.

[5] , Brobst S.. "The evolution of dataflow architectures: from static dataflow to P-RISC", *International Journal of High Speed Computing*, **05**:02 (1993), pp. 125–153.

[6] Lee B., Hurson A. R.. "Dataflow architectures and multithreading", *Computer*, **27**:8 (August 1994), pp. 27–39.

[7] Klimov A. V., Levchenko N. N., Okunev A. S., Stempkovskij A. L.. "The application and implementation issues of dataflow computing system", MES-2016 (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem (MES), 2, IPPM RAN, M., 2016, pp. 100–106 (In Russian).

[8] Stempkovskij A. L., Levchenko N. N., Okunev A. S., Cvetkov V. V.. "Parallel dataflow computing system — the further development of architecture and the structural organization of the computing system with automatic distribution of resources", *Informacionnye texnologii*, 2008, no. 10, pp. 2–7 (In Russian).

[9] Smith K. C., Sedra A. S.. "Associative memory", *Encyclopedia of Computer Science*, John Wiley and Sons Ltd, Chichester, 2003, ISBN 978-0-470-86412-8, pp. 105–106.

[10] Soleimani P., Capson D. W., Li K. F.. "A partitioned CAM architecture with FPGA acceleration for binary descriptor matching", *ACM Transactions on Reconfigurable Technology and Systems*, **17**:1 (2024), id. 10, 21 pp.

[11] Sandhie Z. T., Ahmed F. U., Chowdhury M. H.. "Ternary content addressable memory", *Beyond Binary Memory Circuits*, Synthesis Lectures on Digital Circuits & Systems, Springer, Cham, 2022, ISBN 978-3-031-16194-0, pp. 87–95.

[12] Zmeev D. N., Klimov A. V., Levchenko N. N.. "The tools for computation distribution in the PDCS "Buran" and hash-functions block implementation options", MES-2016 (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 2, IPPM RAN, M., 2016, pp. 107–113 (In Russian).

[13] Zmeev D. N., Levchenko N. N.. “Aspects of creating parallel programs in dataflow programming paradigm”, *Informacionnye texnologii*, **28**:11 (2022), pp. 597–606 (In Russian).

[14] Zmeev D. N., Levchenko N. N.. “Features of creating parallel programs for the parallel dataflow computing system "Buran"”, *Informacionnye texnologii*, **29**:10 (2023), pp. 529–539 (In Russian).

[15] Zmeev D. N.. “Design tools of high-performance dataflow computing systems”, MES-2016 (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 2, IPPM RAN, M., 2016, pp. 159–163 (In Russian).

[16] Zmeev D. N., Okunev A. S.. “Development and investigation of algorithm of sparse matrices multiplication task for the parallel dataflow computing system "Buran"”, MES-2018 (Moskva, Zelenograd, Rossiya, 1 oktyabrya–5 oktyabrya 2018 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 3, IPPM RAN, M., 2018, pp. 16–23 (In Russian).

[17] Stempkovskij A. L., Levchenko N. N., Okunev A. S., Klimov A. V., Zmeev D. N.. “Programming of the molecular dynamics task in the dataflow computing model”, *Informacionnye texnologii*, **23**:12 (2017), pp. 859–867 (In Russian).

[18] Klimov A. V., Levchenko N. N., Okunev A. S., Stempkovskij A. L.. “Supercomputers, memory hierarchy and dataflow computation model”, *Program Systems: Theory and Applications*, **5**:1(19) (2014), pp. 15–36 (In Russian).