RU EN ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ ISSN 2079-3316
научная статья ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ РАСПРЕДЕЛЕННЫХ И СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМ

УДК 004.021+004.272
10.25209/2079-3316-2026-17-3-237-273

Алгоритм обратного распространения ошибки в парадигме потока данных

Дмитрий Николаевич **Змеев**[1✉], Николай Николаевич **Левченко**[2✉], Аркадий Валентинович **Климов**[3✉]

[1–3] Национальный исследовательский центр «Курчатовский институт», Москва, Россия

[1✉] *zmejevdn@list.ru*, [2✉] *n2lev@yandex.ru*, [3✉] *arkady.klimov@gmail.com*

Аннотация. В статье рассматриваются вопросы разработки, реализации и исследования алгоритма обратного распространения ошибки в парадигме управления вычислениями потоком данных. Принципы управления вычислениями потоком данных (dataflow) существенно отличаются от принципов традиционного управления потоком команд (control-flow), как и сама потоковая парадигма программирования отличается от императивной.

Программы, разработанные в потоковой парадигме, изначально являются параллельными, поскольку параллелизм задачи извлекается автоматически из потока данных на аппаратном уровне. В статье представлено детальное описание потокового алгоритма обратного распространения ошибки, реализованного в потоковой программе, предназначенной для выполнения на параллельной потоковой вычислительной системе (ППВС) «Буран».

Потоковая программа отличается компактностью и универсальностью кода, автоматически масштабируется на всю систему и не содержит ссылок на библиотечные функции, основываясь исключительно на базовых арифметических операциях, таких как сложение, умножение и сравнение. Программа позволяет обучать многослойный персептрон произвольной размерности без модификации и перекомпиляции программного кода, при этом размер и структура персептрона определяются начальными данными.

В экспериментальной части статьи представлены результаты исследования поведения ППВС при выполнении задачи обучения, проведен анализ влияния размера обучающего батча на общую эффективность потоковой программы, а также даны оценки применения различных методов аппаратного распределения вычислений. Кроме того, рассмотрен потенциал использования ППВС для параллельного выполнения нескольких задач обучения одновременно.

Ключевые слова и фразы: алгоритм обратного распространения ошибки, параллельное программирование, 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 обратного распространения ошибки в парадигме потока данных* // Программные системы: теория и приложения. 2026. **Т. 17**. №3(72). С. 237–273. https://psta.psiras.ru/read/psta2026_3_237-273.pdf

Введение

Статья представляет собой логическое продолжение работы [1], в которой был описан процесс разработки, реализации и исследования алгоритма прямого распространения в парадигме потока данных (dataflow). В работе были сделаны выводы о компактности и универсальности программного кода, а также о естественном параллелизме программы, обеспечиваемом потоковой моделью вычислений с динамически формируемым контекстом. Экспериментальная часть работы продемонстрировала высокую адаптивность потоковой программы к обработке больших объемов данных, непрерывно поступающих на вход системы.

Следующим этапом в изучении нейросетевых алгоритмов на вычислительных системах с управлением потоком данных является разработка и исследование алгоритма обратного распространения ошибки в парадигме потока данных. В качестве основы для реализации алгоритма был выбран метод стохастического градиентного спуска [2], который является фундаментальным элементом обучения нейронных сетей.

Этот выбор обусловлен не только его широкой популярностью, но и его итерационным характером, который потенциально может затруднить автоматическое (средствами потоковой модели вычислений) выявление параллелизма в задаче. Регулярное обновление весовых коэффициентов при обратном проходе графа нейронной сети препятствует опережающему выполнению прямого прохода этого графа для новых тестовых образцов, что было отмечено в экспериментальной части работы [1], где выполнение алгоритма прямого распространения для следующего тестового образца начиналось до завершения обратного прохода предыдущего.

В связи с этим, необходимо определить цели данной работы. Во-первых, на текущем уровне развития нейросетевых технологий реализация базового алгоритма не представляет практического интереса. Однако, цели и задачи науки – это выработка новых знаний.

Современные аппаратные решения, основанные на концепции управления вычислениями потоком данных, подразделяются на два основных класса: специализированные и универсальные. К специализированным ускорителям относятся такие разработки как Graphcore [3], SambaNova [4], Cerebras [5] и Groq [6]. Эти системы характеризуются реализацией принципа преобразования нейросетевых алгоритмов в статический dataflow граф и его аппаратное декомпозирование на вычислительные элементы.

Основное преимущество данного подхода заключается в повышении эффективности решения задач, связанных с нейронными сетями. Однако стоит отметить, что эффективность этих систем в значительной степени зависит от качества компилятора, который преобразует программы, использующие стандартные библиотеки PyTorch и TensorFlow, в граф

выполнения программы и оптимизирует декомпозицию этого графа на вычислительные узлы с целью минимизации простоев, вызванных ожиданием поступления всех необходимых данных.

Более универсальным решением является ускоритель потока данных компании Maxeler, построенный на базе FPGA. Этот ускоритель также работает с статическим dataflow графом, но, в отличие от специализированных ускорителей, декомпозиция графа на FPGA позволяет выполнять более широкий спектр задач.

Недостатком данной реализации является вторичность ускорителя по отношению к основной программе, выполняемой на XOCT-системе. Это ограничение не позволило авторам серии работ по реализации алгоритма многослойного персептрона на ускорителях Maxeler [7, 8] реализовать алгоритм обратного распространения ошибки. По нашему мнению, сложность и практическая невозможность реализации этого алгоритма были обусловлены трудностями программирования устройства таким образом, чтобы изменения в движении по графу (персептрону) могли происходить без прерывания работы основной последовательной программы и перепрограммирования FPGA, что, в свою очередь, не позволило авторам реализовать одновременную работу алгоритмов прямого и обратного распространения.

Во всех описанных коммерческих решениях используется статический dataflow граф. В рамках научно-исследовательской работы отделения проблем проектирования в микроэлектронике Центра перспективной микроэлектроники Национального исследовательского центра «Курчатовский институт» разрабатывается потоковая модель вычислений с динамически формируемым контекстом.

Ключевое отличие динамического dataflow графа от статического заключается в том, что статический граф основывается на императивном алгоритме, преобразуемом в неизменяемый однонаправленный граф последовательности обработки данных, в то время как динамический граф строится в процессе выполнения программы на основе условий взаимодействия между данными. Ввиду отсутствия примеров реализации нейросетевых алгоритмов на системах с динамическим dataflow графом, можно утверждать, что предложенный в статье алгоритм обратного распространения ошибки в парадигме потока данных и результаты экспериментов по его нативному выполнению на архитектуре параллельной потоковой вычислительной системы (ППВС) «Буран» являются уникальными и не имеют аналогов в российской и мировой научной литературе.

Во-вторых, итерационная природа стохастического метода призвана выявить ограничения потоковой модели вычислений и протестировать

возможности аппаратного выявления параллелизма в, по сути, строго последовательной задаче.

В-третьих, подход к созданию потоковых алгоритмов, представленный в данной статье и работе [1], направлен на сокращение разрыва между программированием в парадигме потока команд и парадигме потока данных, а также на демонстрацию относительной простоты разработки программ в перспективной потоковой парадигме программирования.

Статья имеет следующую структуру. В первой главе кратко описывается потоковая модель вычислений и архитектура ППВС, а также приводятся их ключевые отличия от традиционной фон-неймановской модели вычислений и архитектуры. Вторая глава содержит детальный разбор потокового алгоритма обратного распространения ошибки – от разработки в виде потокового графа до реализации на языке высокого уровня и выбора эффективной функции распределения. Третья глава включает анализ результатов моделирования ППВС при выполнении программы обратного распространения ошибки и оценку перспектив применения ППВС для обучения нейросетей.

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

Ключевое и определяющее различие между потоковой (dataflow) моделью вычислений и традиционной (control-flow) заключается в активации вычислительных операций по готовности данных [10–12]. Такая формулировка подразумевает особую организацию вычислительного процесса, включающую методы хранения и обработки данных. В частности, она предполагает поиск всех данных, необходимых для выполнения конкретного вычислительного кванта, и активацию этого кванта только после успешного нахождения всех требуемых данных. Для эффективного поиска данных используется аппаратная ассоциативная память, обеспечивающая доступ к информации по содержимому (контексту), который определяет позицию данного в виртуальном адресном пространстве задачи (например, при индексации элементов матрицы).

Детальное описание потоковой модели вычислений, включая потоковую модель с динамически формируемым контекстом, представлено в [13]. Важно отметить, что динамическое формирование контекста позволяет изменять контекст и, соответственно, направление движения по dataflow графу в процессе вычислений. Это обеспечивает возможность плавного (без прерываний) перехода от выполнения алгоритма прямого распространения к алгоритму обратного распространения ошибки и обратно.

Очевидно, что особая организация вычислений требуются соответствующих архитектурных решений. Одним из таких решений является

архитектура параллельной потоковой вычислительной системы (ППВС) «Буран», которая реализует потоковую модель вычислений с динамически формируемым контекстом.

Архитектура ППВС (см. рисунок 1) представляет собой масштабируемую структуру, состоящую из набора вычислительных модулей, объединённых коммуникационной сетью для передачи сообщений (токенов).

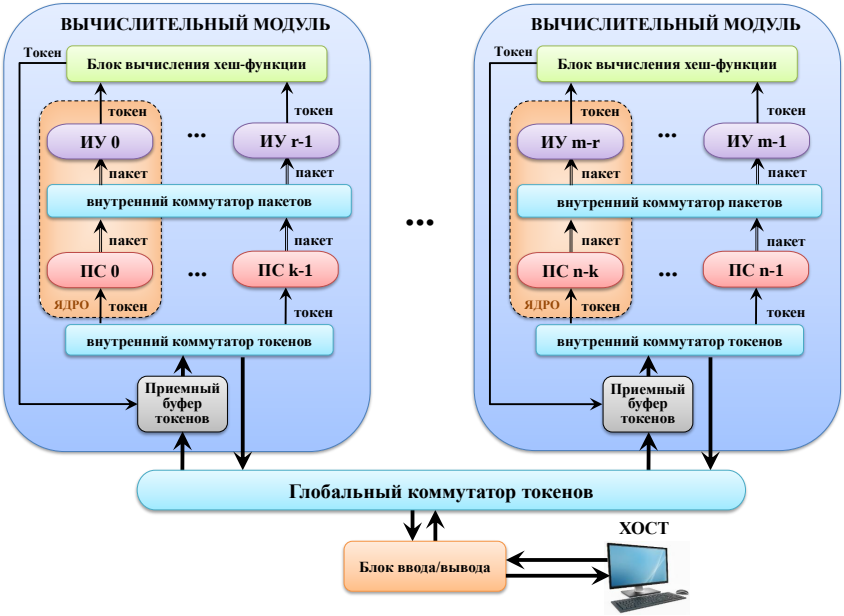

Рисунок 1. Архитектура параллельной потоковой вычислительной системы «Буран»

Токен представляет собой структуру, включающую данные (операнды), контекст и набор признаков. Вычислительный модуль состоит из одного или нескольких процессоров сопоставлений (ПС), коммутатора пакетов, одного или нескольких исполнительных устройств (ИУ) и блоков формирования хэш-функций (БХФ).

Процессор сопоставлений выполняет функции ассоциативной памяти: хранение токенов, поиск парных токенов по контексту и формирование пакетов. Исполнительное устройство является простым вычислителем, предназначенным для выполнения программного кода, указатель на который содержится в пакете, сформированном процессором сопоставлений. Указатель на программный код, представляющий собой номер узла,

является частью ключа сопоставления, который также включает контекст, присутствующий в каждом токене. Программный код представляет собой последовательность команд для обработки входных данных пакета и формирования новых токенов.

Блок формирования хэш-функций предназначен для расчёта номера процессора сопоставлений, в который должен поступить каждый токен, сформированный исполнительным устройством. Этот расчёт осуществляется на основе ключа токена, состоящего из контекста и номера целевого узла. Более подробное описание назначения и логики работы блока формирования хэш-функций представлено в [14]. Подробное описание параллельной потоковой вычислительной системы в целом представлено в [15].

При описании модели вычислений и архитектуры также следует учитывать следующий аспект. Специфическая организация вычислительного процесса и уникальная структурно-функциональная архитектура требуют особого подхода к разработке программного обеспечения для заданного устройства. В рамках проекта ППВС «Буран» был разработан оригинальный язык высокого уровня DFL (DataFlow Language), предназначенный для создания программ в парадигме потока данных. Принципы программирования на языке DFL подробно изложены в работах [16–18] и в [1], где детально описан процесс разработки потоковой программы, реализующей алгоритм прямого распространения.

2. Разработка алгоритма обратного распространения в потоковой парадигме и его реализация на потоковом языке высокого уровня

В работе [1] был использован следующий подход для описания алгоритма прямого распространения. В начале была представлена формула расчета и выполнена ее декомпозиция на элементарные операции. Затем эти операции были отображены в потоковой парадигме, и составлено графическое представление потокового алгоритма. Полученный потоковый граф был транслирован в программный код на языке DFL и составлен псевдокод для формирования начальных данных, согласованный с кодом программы. Аналогичный подход будет применен для описания алгоритма обратного распространения ошибки.

Алгоритм обратного распространения ошибки заключается в корректировке каждого весового коэффициента нейронной сети. Приведенная ниже формула описывает коррекцию весового коэффициента с использованием метода пакетного градиентного спуска.

(1)
$$W^{b+1} = W^b + \frac{\alpha}{q} * \sum_{k=1}^{q} \overline{W}_k^b$$
$$W \equiv w_{ji}^L$$
$$\overline{W} \equiv \overline{w}_{ji}^L$$
$$\overline{w}_{ji}^L = H_i^{L-1} * \overline{S}_j^L$$
$$\overline{S}_j^L = f'\left(S_j^L\right) * \sum_{i=1}^{n}\left(\overline{S}_i^{L+1} \cdot w_{ij}^{L+1}\right)$$
$$H_j^L = f\left(S_j^L\right)$$
$$S_j^L = \left(\sum_i w_{ji}^L \cdot H_i^{L-1}\right) + B_j^L, \quad \text{где}$$

L – индекс слоя,
j – индекс нейрона в слое L,
i – индекс нейрона в слое $(L-1)$, он же индекс входа в нейроне j слоя L
n – количество нейронов в слое L
w – весовой коэффициент i-го входа j-го нейрона в слое L,
B – коэффициент смещения j-го нейрона в слое L,
S – взвешенная сумма входных сигналов нейрона j слоя L,
f – функция активации взвешенной суммы входных сигналов нейрона j слоя L,
f' – производная от функции активации для нейрона j слоя L,
H – результат функции активации нейрона j слоя L,
$\overline{S}$ – ошибка нейрона j слоя L,
$\overline{w}$ – ошибка весового коэффициента,
b – индекс батча (пакета),
k – порядковый номер тестового образца в батче (пакете),
q – количество тестовых образцов в одном батче (пакете),
α – коэффициент скорости обучения,
$\overline{W}$ – ошибка весового коэффициента w_{ji}^L k-го образца b-го батча,
W – весовой коэффициент w_{ji}^L b-го батча.

Детальный анализ формулы (1) выявляет четыре ключевых этапа вычислений: алгоритм прямого распространения, расчет ошибки нейрона, расчет ошибки весового коэффициента и обновление весового коэффициента. Каждый из этапов вычислений, в свою очередь, делится на простые

операции сложения и умножения, и дополняется расчетами функции активации и производной функции активации.

В традиционной парадигме программирования задача решается посредством системы вложенных циклов. Параллелизм задачи ограничен алгоритмической итерационной зависимостью между этапами обратного и прямого распространения по графу. Обновление значений весовых коэффициентов начинается с нейронов последнего слоя, и вычисление прямого распространения с новыми значениями возможно только по завершении этапа обратного распространения, при расчете новых значений весовых коэффициентов для нейронов первого промежуточного слоя.

Для решения проблемы ограниченного параллелизма применяется метод пакетного обновления весовых коэффициентов. В рамках этого метода весь набор тестовых образцов делится на батчи, представляющие собой подмножества тестовых примеров. Для каждого элемента батча выполняется полный цикл прямого и обратного распространения, а обновление весовых коэффициентов осуществляется на основе результатов вычислений всех элементов батча. Это позволяет параллельно выполнять задачу обратного распространения ошибки для всех элементов внутри одного батча.

Термин «батч», используемый в западной профильной литературе для обозначения подмножества тестовых образцов, в отечественной литературе традиционно переводится как «пакет». Однако, в контексте описания потоковой модели вычислений и архитектуры ППВС, термин «пакет» обозначает структуру данных, составленную из сопоставленных токенов, которая передается для вычисления на исполнительное устройство. Поэтому далее для обозначения набора, подмножества или пакета тестовых образцов будет применяться термин «батч».

Переходя к реализации задачи обратного распространения в потоковой парадигме программирования следует отметить, что механизм автоматического извлечения явного и неявного параллелизма, заложенный в потоковую модель вычислений, не способен полностью устранить итерационную зависимость, связанную с обновлением весов после каждой итерации. В связи с этим, в качестве основы для разработки потокового алгоритма был выбран метод пакетного обновления весовых коэффициентов, при котором обновления весов производятся один раз после обработки батча.

Представленный на рисунках 2–4 потоковый алгоритм пакетного градиентного спуска состоит из девяти программных узлов и использует аппаратные возможности ППВС «Буран», включая аппаратное суммирование вещественных значений в процессоре сопоставлений без задействования исполнительного устройства. Алгоритм поддерживает любую структуру многослойного персептрона, размер которого ограничен разрядностью используемых полей контекста.

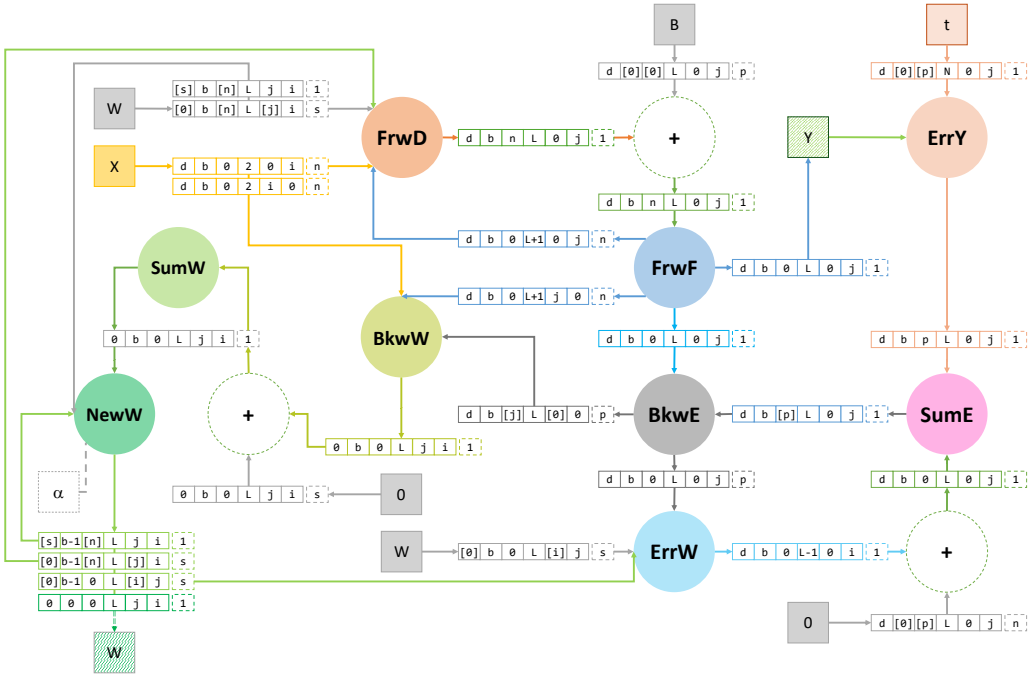

Рисунок 2. потоковый граф алгоритма обратного распространения ошибки

```
module BackPropagation;

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

const a = 0.01;

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

node FrwF (x : double) : F1;
var ReLU, dReLU : double;
begin
  if x<0 then ReLU:=0; dReLU:=0;
         else ReLU:=x; dReLU:=1;
  end;

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

  send dReLU to BkwE.dh {d,b,0,L,0,i};
end;

node ErrY(y, t : double) : F1;
begin
  send <start> t-y to SumE {d,b,n,L,0,i};
end;

node BkwE(e, dh : double) : F1;
begin
              send c*dh to BkwW.e {d,b,[i],L,[0],0} <<n>>;
  if L>2 then send c*dh to ErrW.e {d,b, 0, L, 0, i} <<n>>; end;
end;

node ErrW(w, e : double) : F1; dgrouped;
begin
  send <fsum> e*w to SumE.e {d,b,0,L-1,0,j};
end;

node SumE(e : double) : F1;
begin
  send e to BkwE.e {d,b,[n],L,0,i};
end;

node BkwW(h, e : double) : F1; dgrouped;
begin
  send <fsum> h*e to SumW.dw {0,b,0,L,n,j};
end;

node SumW(dw: double) : F1;
begin
  send dw to NewW.dw;
end;

node NewW(w, dw : double) : F1;
var nw : double;
begin
  nw:=w+dw*a/d;

  if b=0 then send nw to host;
         else send nw to FrwD.w {[0],b-1,[n],L,[j],i} <<d>>;
              send nw to ErrW.w {[0],b-1, 0, L,[i],j} <<d>>;
              send nw to NewW.w {[d],b-1,[n],L, j, i};
  end;
end;

endmod.
```

Рисунок 3. программный код алгоритма обратного распространения ошибки на языке DFL

```
      // D                          - количество образцов
      // E                          - количество пакетов обучения (число батчей)
      // S                          - размер пакета обучения (размер батча)
      // 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
      // T[1..D][1..L[N]            - массив целевых значений для каждого образца
      // W[2..N][1..L[1]][1..L[l-1]] - матрица весовых коэффициентов  [слой][нейрон][вход]
      // B[2..N][1..L[B]]           - матрица коэффициентов смещения [слой][нейрон]

      //Входные данные
      for d:=1 to X.size do
        for i:=1 to X[d].size do
<1>       send X[d][i] to FrwD.h {d,(D-d) div S,0,2,0,i} <<L[2]>>;
<2>       send X[d][i] to BkwW.h {d,(D-d) div S,0,2,i,0} <<L[2]>>;

      //Целевые значения
      for d:=1 to T.size do
<3>     for j:=1 to L[N] do send T[d][j] to ErrY.t {d,[0],[L[N-1]],N,0,j};

      //Весовые коэффициенты
      for l:=2 to L.size-1 do
        for j:=1 to L[l] do
          for i:=1 to L[l-1] do
<4>         send W[l][j][i] to FrwD.w {[0],D/S+1,[L[l+1]],l,[j],i} <<S>>;
<5>         send W[l][j][i] to NewW.w {[S],D/S+1,[L[l+1]],l, j, i};
<6>         send W[l][j][i] to ErrW.w {[0],D/S+1, 0,      l,[i],j} <<S>>;

      //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[l] do
<7>         send <fsum> B[l][j] to FrwF {d,[0],[0],l,0,j} <<L[l-1]>>;
        for l:=2 to L.size-2 do
          for j:=1 to L[l] do
<8>         send <fsum> 0 to SumE {d,[0],[L[l-1]],l,0,j} <<L[l+1]>>;
      for e:=1 to E do
        for l:=2 to L.size-2 do
          for j:=1 to L[l] do
            for i:=1 to L[l-1] do
<9>           send <fsum> 0 to SumW {0,E-e,0,l,j,i} <<S>>;
```

Рисунок 4. псевдокод формирования начальных данных алгоритма обратного распространения ошибки

2.1. Графическое представление алгоритма

Графическое представление (рисунок 2) – это особая форма визуализации, предназначенная для упрощенного формирования и представления алгоритма.

Правила и формы составления потокового графа соответствуют тем, которые были описаны для алгоритма прямого распространения

в работе [1].

Входные и выходные данные обозначаются в виде закрашенных квадратов с указанием названия в центральной части (квадраты W, X, B, Y, t и 0).

Программные узлы представляются в виде закрашенных окружностей с указанием названия узла в центральной части (`FrwD`, `BkwW` и другие).

Специальные функции, реализованные в процессоре сопоставлений, обозначаются белыми кругами с пунктирными границами («+» символизирует аппаратное суммирование заданного числа данных).

Константы представляются в виде квадратов с пунктирными границами и именами в центральной части (α) и соединяются пунктирными линиями с программными узлами, использующими их значения в расчетах.

Потоки данных между элементами отображаются с помощью линий со стрелками, на которых указаны контекст и кратность токенов.

Контекст токена представлен набором полей с значениями, обозначенными в клеточках со сплошной границей на линии связи.

Значения полей контекста, заключенные в квадратные скобки, указывают на то, что эти поля замаскированы и не учитываются при сопоставлении токенов.

Значение кратности токена указывается справа от контекста в отдельном квадрате с пунктирной границей (также используется для указания числа слагаемых аппаратного суммирования).

Представленный на рисунке 2 потоковый граф отображает следующий алгоритм вычислений:

Первый этап алгоритма (прямое распространение).

Производится вычисление H_j^L (1) для каждого нейрона в каждом слое нейронной сети, относящееся к каждому отдельному тестовому образцу в рамках текущего батча.

- Потоки данных W (весовые коэффициенты) и X (значения тестовых образцов) поступают на узел `FrwD`, где выполняется их умножение.
- Результат умножения передается на узел `FrwF` посредством операции аппаратного суммирования, представленного узлом «+», дополнительным аргументом которого является поток данных B (коэффициент смещения в значении операнда и количество слагаемых аппаратного суммирования, указанное в позиции кратности).
- Результат сложения указанного количества произведений и коэффициента смещения инициирует активацию узла `FrwF`, который выполняет вычисление функции активации, ее производной, а также проверку условий завершения первого этапа алгоритма.

- Результат вычисления функции активации передается в качестве второго множителя в узел `FrwD` для последующего взаимодействия с весовыми коэффициентами на входах нейронов следующего слоя.

Второй этап алгоритма включает вычисление ошибки нейрона $\overline{S}_j^L$ (1) для каждого нейрона в каждом слое, относящихся к каждому отдельному тестовому образцу в пределах текущего батча. Расчет осуществляется путем умножения производной функции активации на взвешенную сумму ошибок.

- Результат вычисления производной функции активации (узел `FrwF`) передается в качестве первого аргумента узлу `BkwE`.
- В качестве второго аргумента узлу `BkwE` передается либо ошибка выходного нейрона, либо взвешенная сумма ошибок, полученная от входов нейронов следующего слоя (здесь и далее под «следующим» понимается слой с большим порядковым номером).
- Ошибка выходного слоя вычисляется в узле `ErrY`. На первый вход узла `ErrY` поступает поток данных t (целевые значения), а на второй вход – данные из узла `FrwF` (значения функции активации для нейронов последнего слоя). В узле `ErrY` вычисляется функция ошибки, в описываемой реализации алгоритма выбрана функция разности между целевым и выходным значениями.
- Взвешенная сумма ошибок от входов нейронов следующего слоя вычисляется путем аппаратного суммирования данных, поступающих из узла `ErrW`. В узле `ErrW` выполняется операция умножения ошибки нейрона следующего слоя (дублируемой из узла `BkwE`) на значение соответствующего весового коэффициента (поток данных W). Дополнительным аргументом для аппаратного суммирования является поток нулевых значений, задающий (в кратности) количество слагаемых.
- Значение ошибки выходного слоя или взвешенной суммы поступает на промежуточный узел `SumE`, который передает полученный поток данных на второй вход узла `BkwE`.
- В узле `BkwE` производится операция умножения производной функции активации на ошибку. Результат операции дублируется: направляется на второй вход узла `BkwW` для расчета ошибки весового коэффициента и на узел `ErrW` для следующей итерации вычислений второго этапа.

На третьем этапе алгоритма производится вычисление ошибки весового коэффициента $\overline{w}_{ji}^L$ (1) для каждого входа в каждом нейроне в каждом слое, относящихся к каждому отдельному тестовому образцу в текущем батче. Вычисление осуществляется путем умножения значения нейрона (которое представляет собой результат функции активации) на величину ошибки нейрона.

- На вход узла `BkwW` поступают два потока данных: один с узла `FrwF` представляющий собой дублирование результата вычисления функции активации, и второй с узла `BkwE`, содержащий информацию об ошибке нейрона.
- В узле `BkwW` осуществляется расчет ошибки весового коэффициента посредством умножения значения функции активации нейрона предыдущего слоя на величину ошибки нейрона текущего слоя.
- Результат вычисления ошибки весового коэффициента передается на узел `SumW` посредством аппаратного суммирования. Дополнительным аргументом для аппаратного суммирования является поток нулевых значений, задающий (в кратности) количество слагаемых, равное числу тестовых образцов в батче.

Четвертый этап алгоритма включает в себя обновление весового коэффициента W^{b+1} (1) для каждого входа в каждом нейроне каждого слоя. Расчет нового весового коэффициента осуществляется посредством корректировки текущего значения на усредненную ошибку этого весового коэффициента, рассчитанную по отдельности для каждого тестового образца в текущем батче.

- Результат аппаратного суммирования ошибок, рассчитанных для каждого тестового образца в текущем батче, инициирует активацию промежуточного узла `SumW`, который пересылает поток данных на второй вход узла `NewW`.
- На первый вход узла `NewW` поступают актуальные значения весовых коэффициентов.
- Новое значение весового коэффициента определяется путем модификации текущего значения на величину, равную произведению усредненной ошибки на коэффициент обучения α.
- Результат коррекции передается на соответствующие входы узлов `FrwD`, `ErrW` и `NewW`, что обновляет поток данных на эти узлы новыми значениями. При этом, в процессе взаимодействия предыдущих коэффициентов с соответствующими данными происходит постепенное снижение их кратности, что приводит к удалению этих коэффициентов непосредственно перед обновлением значений. Это предотвращает накопление «мусорных» данных в памяти.
- Итоговым результатом четвертого этапа и всего алгоритма является передача на XOCT-машину обновленных значений весовых коэффициентов, рассчитанных для последнего батча.

Алгоритм может быть описан следующим образом: для каждого тестового образца (поток данных X) выполняется процедура прямого распространения (узлы `FrwD` и `FrwF`), результатом которой является поток данных Y. Этот результат сравнивается с целевым значением (поток

данных t) тестового образца, после чего узел `ErrY` вычисляет ошибку выходного слоя.

Ошибка затем последовательно распространяется в обратном направлении от последнего к первому промежуточному слою. Процесс обратного распространения ошибки для каждого тестового образца осуществляется поэтапно, где для каждого слоя рассчитываются ошибки нейронов (`BkwE`, `ErrW`, `ErrY` и `SumE`) и ошибки весовых коэффициентов (`BkwW`). Обновление каждого весового коэффициента (узел `NewW`) выполняется независимо и суммируется (узел `SumW`) на основе ошибок данного веса, рассчитанных для каждого тестового образца в текущем обрабатываемом батче.

Важным аспектом любого потокового алгоритма является управление контекстом формируемых и поступающих данных. В работе [1] подчеркивается значимость и сложность разработки структуры контекста, которая, однако, может быть упрощена при использовании формул в качестве основы для формирования контекста.

Анализ формулы (1) указывает на необходимость включения в контекст следующих полей: L для идентификации слоя нейрона, j для идентификации нейрона, i для идентификации входа нейрона, b для идентификации батча и d для идентификации тестового образца. Кроме того, механизм «сквозной» передачи значений, используемый в алгоритме прямого распространения, также применяется на различных этапах алгоритма обратного распространения ошибки, что требует добавления в контекст отдельного поля для маскирования и передачи закодированных значений до целевых узлов.

Итоговый формат контекста, применимый ко всем токенам задачи, как начальным, так и формируемым в процессе выполнения программы, включает шесть полей: d (индекс тестового образца), b (индекс батча), n (поле для «сквозной» передачи данных), L (индекс слоя), j (индекс нейрона) и i (номер входа).

2.2. Описание программного кода алгоритма на языке высокого уровня

На рисунке 3 представлен программный код алгоритма обратного распространения ошибки, реализованный на языке высокого уровня DFL.

Структура кода на языке DFL включает блок описания типов и констант, а также список независимых программных узлов. Каждый программный узел состоит из заголовка и последовательности команд обработки данных (аргументов узла, констант и литералов) и контекста,

команд перехода (внутри узла) и специальной команды формирования токенов, предназначенной для формирования и передачи данных в другой узел.

Одной из задач работы [1] была демонстрация процесса реализации потокового алгоритма на языке DFL. В работе объяснялась логика применения аппаратных возможностей ППВС и различных типов взаимодействия токенов, что позволяет упростить программный код и оптимизировать его выполнение. Эти возможности также использовались при реализации алгоритма обратного распространения ошибки.

Аппаратное суммирование данных в процессоре сопоставлений применяется для накопления произведений на входе нейрона, для накопления произведений для расчета ошибки нейрона и для сбора ошибок весового коэффициента (рассчитанных для заданного числа тестовых образцов) с целью определения нового значения этого весового коэффициента при обработке тестовых образцов в следующем батче. Признак `dgrouped` («двойной групповой») программного узла реализует схему взаимодействия двух токенов по правилу «каждый с каждым» (при сопоставлении двух токенов формируется только один пакет) и используется для вычисления только одного произведения для пары аргументов на узлах `FrwD` (веса и значения нейронов), `ErrW` (веса и ошибки нейронов) и `BkwW` (значения и ошибки нейронов).

Важной частью реализации программного кода является согласованность форматов контекста и имен программных узлов с начальными данными, псевдокод формирования которых представлен на рисунке 4. При описании программного кода для потоков начальных данных, направленных на входы узлов, в угловых скобках указывается номер строки на рисунке 4, где закодирована логика построения этого потока данных.

Этап прямого распространения. Описание программных узлов `FrwD` и `FrwF` идентично описанию алгоритма прямого распространения. Узел FrwD выполняет операцию умножения весовых коэффициентов на данные тестовых образцов и значения нейронов. Он имеет два вещественных аргумента w и h с контекстом формата `F1`, определенным в начале кода программы после ключевого слова *type*. Узел обладает признаком «двойной групповой».

Код узла `FrwD` состоит из единственной команды, формирующей токен. В этой команде результат умножения входных аргументов w и h передается на вход узла `FrwF` посредством аппаратного суммирования (признак `fsum`). Контекст генерируемого токена идентичен контексту

пакета, активировавшего программный узел, за исключением обнуления поля j и переноса значения индекса нейрона в поле i. Это изменение осуществляется для оптимизации распределения вычислений, которые проводятся по полям L и i.

Активация узла `FrwD` на начальной стадии обеспечивается посредством взаимодействия весовых коэффициентов и тестовых данных, относящихся ко второму (первому промежуточному) слою. Код для генерации начальных значений весовых коэффициентов приведен в <4>. В формируемом токене явно указываются индексы батча, слоя и входа нейрона.

Номер тестового образца обнуляется и маскируется для обеспечения группового взаимодействия со всеми тестовыми образцами указанного батча. Индекс нейрона также обнуляется и маскируется для обеспечения взаимодействия на указанных входах всех нейронов заданного слоя. Резервное поле n маскируется и используется для «сквозной» передачи значения числа нейронов в следующем слое. Кратность формируемого токена устанавливается равной размеру батча, что гарантирует его удаление после взаимодействия с соответствующими данными всех тестовых образцов в батче.

Код для генерации тестовых данных приведен в <1>. В формируемом контексте явно указываются индексы тестового образца и батча, индекс слоя устанавливается равным 2, а порядковый номер данного из массива тестовых данных указывается в поле i, поскольку для нейронов второго слоя этот номер относится к предыдущему (первому) слою. Кратность тестовых данных устанавливается равной числу нейронов во втором слое.

Таким образом, в <4> реализуется формирование всех весовых коэффициентов для взаимодействия с данными всех нейронов, полученными в процессе обработки всех тестовых образцов начального батча, а в <1> реализуется формирование данных всех тестовых образцов для взаимодействия с весовыми коэффициентами, относящимися к нейронам второго слоя.

Программный модуль `FrwF`, помимо вычисления функции активации, также используется для расчета производной функции активации. Этот пример наглядно иллюстрирует различия между традиционной и потоковой моделями вычислений.

В традиционной модели вычислений инициатором вычислений является потребитель данных - для получения значения функции активации вызывается соответствующая функция, которая возвращает результат, рассчитанный на основе переданного в нее аргумента; для получения

значения производной функции активации вызывается другая функцию, которая выполняет иной набор вычислений для того же аргумента. Таким образом, в традиционной программе для двух различных последовательностей обработки одного и того же аргумента используются две разные функции: одна для вычисления функции активации, другая – для вычисления ее производной.

В потоковой парадигме понятие запроса данных отсутствует, а инициатором вычислений выступает источник данных. При расчете функции активации и ее производной указываются потребители обрабатываемых значений, что позволяет интегрировать в одном программном модуле две различные операции над одним и тем же аргументом. Это позволяет оптимизировать вычисления, добавив в код программного узла `FrwF` определение значения переменной `dReLU` (производная функции активации `ReLU`), передачу этого значения на узел `BkwE` и дублирование значения функции активации `ReLU` не только на узел `FrwD`, но и на узлы `ErrY` и `BkwW`.

Контекст токенов, формируемых в узле `FrwF`, составляется в соответствии с формулой (1). Общая часть контекста для всех четырех токенов включает принадлежность данных к тому же батчу (поле b) и тестовому образцу (поле d), что и сумму произведений на входе узла. Основные различия касаются принадлежности передаваемых данных к слою (поле L) и их позиционирования в этом слое – относятся ли они к нейрону в этом слое или к входам каждого нейрона слоя. При передаче значения функции активации в узел `FrwD` производится расчет H_j^L, относительно индексации которого расчет функции активации выполнялся на предыдущем слое.

В результате номер слоя (L) инкрементируется, а значение номера нейрона помещается в поле i для привязки к входам нейрона. Аналогичный инкремент номера слоя осуществляется при передаче значения в узел `BkwW` для вычисления ошибки весового коэффициента $\overline{w}_{ji}^L$, при этом номер нейрона размещается в соответствующем поле j.

Кратность обоих токенов, передаваемых в узлы `FrwD` и `BkwW`, определяется значением поля n, через которое при формировании весовых коэффициентов <4> передается количество нейронов, обозначающее число взаимодействий передаваемого значения с весовыми коэффициентами на следующем слое. Индексация поля L при передаче токенов на узлы `ErrY` и `BkwE` остается неизменной, так как эти узлы участвуют в расчете ошибки нейрона $\overline{S}_j^L$, где индекс L с обеих сторон равенства идентичен. Кратность обоих токенов равна единице, поскольку оба передаваемых значения (результат прямого распространения на узел `ErrY` и производная функции

активации на узел BkwE) используются в каждом заданном нейроне слоя только один раз.

Этап вычисления ошибки нейрона. Ошибка нейрона $\overline{S}_j^L$ вычисляется для каждого нейрона каждого слоя один раз и используется для расчета ошибки всех нейронов предыдущего слоя в процессе обратного распространения по графу персептрона. Начальный этап расчета ошибок нейронов определяется вычислением ошибки выходного слоя.

Этот расчет выполняется в узле ErrY, на один из входов которого поступает результат вычисления функции активации для нейронов последнего слоя в узле FrwF (поле n установлено в нуль, что указывает на отсутствие нейронов в следующем слое, следовательно, текущий слой является последним). На другой вход узла ErrY подается целевое значение <3>. Контекст целевого значения формируется таким образом, чтобы исключить влияние номера батча (поле b маскируется), а поле n используется для передачи значений, в конкретном случае, для передачи в узел BkwE количества нейронов в предпоследнем слое.

Последующий расчет ошибок нейронов осуществляется на основании значений ошибок, полученных на предыдущей итерации замкнутого цикла, который состоит из последовательности узлов BkwE → ErrW → SumE → BkwE. Каждая итерация цикла начинается с передачи в узел ErrW значения ошибки нейрона, вычисленного на предыдущей итерации, и последующего умножения этой ошибки на текущее значение соответствующего весового коэффициента.

Контексты и кратность ошибки в узле BkwE и весового коэффициента <6> формируются таким образом, чтобы обеспечить взаимодействие каждой ошибки нейрона со всеми весовыми коэффициентами на входах данного нейрона. Для этого кратность ошибки нейрона устанавливается равной числу нейронов предыдущего слоя (значение фиксируется в поле n), а индекс нейрона, с целью оптимизации распределения, размещается в поле i. Кроме того, в узле BkwE у весового коэффициента маскируется номер тестового образца, а кратность устанавливается равной размеру батча. Это позволяет локализовать и дублировать передаваемое значение при расчетах всех тестовых образцов текущего батча.

Результаты выполнения узла ErrW группируются по нейронам предыдущего слоя с целью последующего суммирования произведений и передачи итогового значения на следующую итерацию цикла. Группировка произведений осуществляется посредством декремента номера слоя и фиксации номера нейрона в поле i (что фактически было выполнено на

предыдущем этапе, когда значения номеров нейрона и входа нейрона были переставлены местами для обеспечения корректного взаимодействия с ошибкой нейрона).

Количество слагаемых в аппаратном суммировании определяется в <8> и соответствует числу нейронов в предыдущем слое относительно нейрона, для которого вычислена ошибка. Полученная сумма передается без изменений через узел `SumE` в узел `BkwE`, где она умножается на производную функции активации, рассчитанную для соответствующего слоя, инициируя начало следующей итерации цикла вычисления ошибок нейронов.

Этап вычисления ошибки весового коэффициента. Ошибка весового коэффициента $\overline{w}_{ji}^{L}$ является частью процесса вычисления коррекции весового коэффициента, которая реализуется на последующем этапе посредством определения средней ошибки этого весового коэффициента для всей совокупности тестовых образцов текущего батча. Для расчета индивидуальной ошибки весового коэффициента применительно к каждому тестовому образцу в узле `BkwW` осуществляется операция умножения ошибки указанного нейрона на значение функции активации, соответствующее нейрону предыдущего слоя и ассоциированное с указанным нейроном весовым коэффициентом.

В соответствии с формулой (1), индексация ошибки весового коэффициента формируется на основе комбинации индексов функции активации (i) и ошибки нейрона (j). Таким образом, каждый элемент из потока данных H (функции активации) взаимодействует с каждым элементом из потока данных S (ошибки нейронов).

Этот тип взаимодействия «каждый с каждым» аналогичен узлам `FrwD` и `ErrW` и обеспечивается аппаратным признаком `dgrouped`, а также следующими правилами составления контекстов и кратностей для обоих потоков данных. Номер тестового образца (поле d) и индекс батча (поле b) остаются неизменными для обоих потоков данных. Номер слоя (поле L) для потока ошибок нейрона также остаётся неизменным относительно номера слоя, для которого рассчитывается ошибка весового коэффициента. Для потока функций активации номер слоя инкрементируется, так как требуется приведение данных к единому адресному пространству, а соответствующие рассчитанные значения функций активации относятся к нейронам предыдущего слоя, что подтверждается индексом $(L-1)$ при индексации H в формуле (1).

Более сложной задачей является комбинация индексов результирующего произведения (фактически, индекса входа нейрона) из номера

нейрона, для которого рассчитана функция активации, и номера нейрона, для которого рассчитана ошибка. Для первого потока данных H значение номера нейрона помещается в поле j, а для второго потока данных $S-$ в поле n. При этом поля n и j маскируются, чтобы исключить их из условий взаимодействия и гарантировать активацию узла `BkwW` для пар данных с различными комбинациями полей i и j. Кратности обоих потоков данных задаются равными числу нейронов в соответствующих слоях и, как уже было неоднократно отмечено, в обоих случаях значения берутся из поля n, используемого для «сквозной» передачи значений.

Результаты выполнения узла `BkW` явно позиционируют (по полям L, j и i) ошибки весового коэффициента как элементы коррекции конкретного весового коэффициента. Эти элементы группируются (через обнуление номера тестового образца) по индексу батча для последующего аппаратного суммирования. Количество слагаемых ошибок каждого весового коэффициента определяется в <9> и соответствует размеру батча. Результат суммирования всех ошибок весового коэффициента, через промежуточный узел `SumW`, передается на узел `NewW`, где осуществляется заключительный этап обновления коэффициента.

Этап обновления весового коэффициента. Обновление весового коэффициента W^{b+1} в соответствии с формулой (1) предполагает корректировку текущего значения весового коэффициента на величину средней ошибки весового коэффициента, рассчитанной для всех тестовых образцов текущего батча, умноженную на значение коэффициента скорости обучения. Полученное значение затем дублируется для всех узлов, использующих этот весовой коэффициент в своих вычислениях.

Все операции, связанные с обновлением весового коэффициента, осуществляются в узле `NewW`. На вход этого узла поступают текущее значение весового коэффициента и сумма ошибок весового коэффициента, которая была вычислена на предыдущем этапе. Расчет нового значения весового коэффициента выполняется в первой строке программного кода узла `NewW` (переменная nw). В ней к текущему значению весового коэффициента (w) добавляется сумма ошибок (dw), которая делится на размер батча (значение поля d) и умножается на коэффициент скорости обучения (α).

Значение размера батча доступно для расчетов в узле `NewW` благодаря «сквозной» передаче через поле d, которое было задано при формировании контекста начального весового коэффициента <5> или при дублировании нового значения весового коэффициента на узел `NewW`. Перед отправкой

нового значения весового коэффициента проводится анализ текущего индекса батча (поле b).

В рамках оптимизации кода и алгоритма используется схема изменения индекса батча от большего значения к меньшему.

В случае, если текущий индекс батча равен нулю, алгоритм считается завершенным, и рассчитанное значение весового коэффициента передается на XOCT-машину. Для этого используется ключевое слово `host` вместо названия целевого узла.

В остальных ситуациях формируются токены на узлы `FrwD`, `ErrW` и `NewW`. Контекст и кратность этих токенов определяются в соответствии с правилами формирования начальных данных для соответствующих узлов: на узел `FrwD` в соответствии с <4>, на узел `ErrW` в соответствии с <6> и на узел `NewW` в соответствии с <5>.

2.3. Выбор функции распределения вычислений

Функция распределения является ключевым элементом распараллеливания вычислений в системах с управлением потоком данных. Функция распределения представляет собой аппаратную реализацию алгоритма расчета номера вычислительного ядра на основе контекста токена. В работе [1] были подробно рассмотрены различные варианты функций распределения, и обоснованно выбран стандартный (`STD`) алгоритм для распределения всех токенов задачи по полям L и i контекста. Выбор обусловлен тем, что только данная комбинация параметров обеспечивает равномерное распределение токенов по вычислительным ядрам, что способствует повышению эффективности функционирования системы при выполнении задачи прямого распространения.

Контекст токенов в задаче обратного распространения ошибки идентичен контексту задачи прямого распространения, за исключением добавленного поля b (индекс батча), которое не подлежит учету при распределении, поскольку его значение изменяется итеративно, что может негативно повлиять на равномерность распределения токенов. В связи с этим, логичным выбором функции распределения для задачи обратного распространения ошибки является та же стандартная функция распределения по полям L и i. Однако, учитывая итеративный характер выполнения задачи и сложность алгоритма по сравнению с задачей прямого распространения, выбор в пользу стандартной функции должен быть обоснован серией экспериментов, в ходе которых проводилось сравнение времени выполнения задачи с использованием всех доступных

функций распределения: стандартной (STD), распределения «по полю» (FLD) и блочного распределения (ZIP).

2.4. Оценка программного кода

Программный код алгоритма обратного распространения ошибки превосходит по объему и сложности программный код алгоритма прямого распространения. Однако, несмотря на это, ему присущи следующие характеристики:

- Независимость от структуры нейронной сети (количество слоев и нейронов в каждом слое), поскольку она определяется начальными значениями весовых коэффициентов.
- Отсутствие вызовов внешних библиотечных функций; код состоит исключительно из элементарных операций.
- Прямое отображение математической формулы, что облегчает чтение и понимание кода благодаря идентичной индексации.

Сравнение представленного программного кода с традиционными фреймворками для обучения нейросетей, такими как TensorFlow и PyTorch, не имеет существенного значения, поскольку функциональные возможности этих фреймворков значительно превосходят представленный код. Сравнение в этом контексте сводится к количественному подсчету строк кода, что не является продуктивным подходом. Остается лишь отметить, что представленный программный код реализует базовый алгоритм обучения и состоит из элементарных арифметических операций, таких как сложение, умножение и сравнение.

3. Исследование работы алгоритма обратного распространения ошибки на модели ППВС

Основной целью экспериментов является выявление закономерностей поведения вычислительной системы с управлением потоком данных при решении задачи обучения нейронной сети, включая оценку влияния размера батча и функции распределения на эффективность выполнения рассматриваемой задачи. Задача оценки эффективности потоковой программы по сравнению с существующими аппаратными решениями не ставилась. Кроме того, учитывая различия в уровне проработки и внедрения архитектуры ППВС и реализованного алгоритма, из целей проведения экспериментов было исключено сравнение с современными решениями, направленными на ускорение вычислений нейросетевых задач.

Серия экспериментов проводилась на поведенческой блочно-регистровой модели (ПБРМ) [19], предназначенной для моделирования поведения ППВС при выполнении потоковых программ. Выбор метода проведения экспериментов основывался на результатах, полученных в работе [1]. Все эксперименты проводились над персептроном с равномерной структурой, которая показала наилучшую устойчивость к масштабируемости архитектуры, что выражается в более позднем снижении эффективности при увеличении вычислительных ресурсов.

Параметры модели были определены с учетом технических возможностей ПБРМ. Нейронная сеть состоит из 16 слоев (1 входной слой, 1 выходной слой и 14 промежуточных слоев), каждый из которых содержит по 16 нейронов. Количество тестовых образцов для экспериментов составило 64.

На основании установленных целей проведения экспериментов, выбраны следующие параметры, влияющие на анализ поведения ППВС: количество вычислительных ядер архитектуры (в диапазоне от 1 до 128 с шагом степени двойки) и размер батча (в диапазоне от 1 до 64 с шагом степени двойки).

Основная серия экспериментов была выполнена с использованием стандартной функции распределения по полям L (индекс слоя) и i (индекс входа нейрона). Для оценки влияния функции распределения на эффективность выполнения нейросетевого алгоритма была проведена дополнительная серия экспериментов на задаче с 64 тестовыми образами, размером батча 8 и функциями распределения `Fld(L)`, `Fld(i)` и `Zip(L, i)`.

В результате моделирования предполагалось подтвердить эффективность стандартной функции вычисления и выявить снижение масштабируемости задачи, обусловленное итерационной природой алгоритма. Снижение масштабируемости проявляется в суммарной недоиспользованности вычислительных ресурсов при их увеличении для решения одной и той же задачи, то есть время решения задачи должно уменьшаться непропорционально росту числа доступных вычислительных ядер.

С учётом возможного получения такого результата, была запланирована серия дополнительных экспериментов, направленных на исследование мультизадачности. В рамках этой серии запланировано одновременное выполнение двух и четырёх задач обучения нейронной сети, состоящей из 8 слоёв по 8 нейронов в каждом слое, на вычислительной системе с числом ядер от 1 до 128 (с шагом степени двойки). Ожидаемым результатом

серии экспериментов является повышение общей масштабируемости задач, что будет выражаться в увеличении времени выполнения каждой отдельной задачи при одновременном сокращении их суммарного времени выполнения.

В таблице 1 представлены основные результаты моделирования поведения ППВС «Буран» для задачи обратного распространения ошибки. Проведенный анализ влияния размера батча на продолжительность выполнения одной задачи выявил тенденцию к снижению «удельной» эффективности выполнения задачи по мере увеличения размера батча и количества задействованных вычислительных ядер.

Таблица 1. Время в условных тактах выполнения задачи обратного распространения ошибки с разными значениями размера батча q

q	**Архитектура ППВС, количество вычислительных ядер**							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1	27 994 674	15 307 952	8 368 446	5 800 393	3 839 077	3 213 984	3 619 138	2 348 938
2	20 006 162	10 206 851	5 711 849	4 175 801	2 768 634	2 204 262	2 064 873	1 330 401
4	16 009 938	8 272 894	4 858 631	3 202 291	2 301 754	1 736 085	1 561 022	1 024 543
8	14 008 994	7 342 211	4 331 266	2 933 746	2 123 807	1 557 581	1 340 374	785 593
16	13 000 098	6 899 167	4 060 547	2 855 261	2 035 752	1 400 161	1 256 583	714 904
32	12 479 906	6 651 188	3 867 280	2 835 562	1 997 408	1 357 488	1 210 678	639 692
64	12 188 322	6 615 743	3 784 012	2 797 306	1 992 760	1 381 806	1 202 621	618 751

Под «удельной» эффективностью здесь понимается эффективность выполнения задачи с размером батча q на N вычислительных ядрах по сравнению с выполнением задачи с размером батча 1 на одном ядре. Показатель рассчитывается по следующей формуле:

$$U_q^N = T_1^1 \Big/ \frac{T_q^N}{N}, \quad \text{где} \tag{2}$$

q – размер батча,

N – число вычислительных ядер,

T_q^N – время выполнения задачи с размером батча q на архитектуре ППВС «Буран» с N вычислительными ядрами,

U_q^N – «удельная» эффективность выполнения задачи с размером батча q на N вычислительных ядрах относительно выполнения задачи с размером батча 1 на 1 ядре.

«Удельная» эффективность позволяет оценить аппаратную и программную масштабируемость задачи, а именно: эффективность увеличения вычислительных ресурсов для решения одной и той же задачи и эффективность «дробления» тестовых образцов на батчи. Детальный анализ результатов позволяет сделать вывод о том, что увеличение размера батча не приводит к пропорциональному ускорению выполнения задачи, однако обеспечивает значительное повышение эффективности уже при значении двух тестовых образцов на батч. Это повышение достигается за счет параллельного выполнения каждым отдельным вычислительным ядром нескольких прямых и обратных проходов (для нескольких тестовых образцов) по графу персептрона (эффект конвейеризации вычислений, описанный в [1]).

Значение «удельной» эффективности меньше единицы (начиная с 16 ядер, см. рисунок 5) свидетельствует о наличии избыточных вычислительных мощностей для выполнения задач заданной размерности. Таблица 2

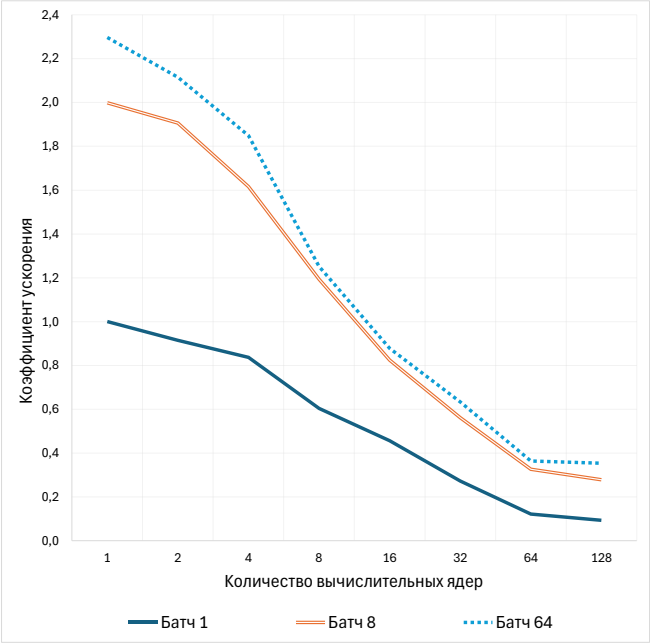

Рисунок 5. «Удельная» эффективность выполнения задачи обратного распространения ошибки с разным размером батча на ППВС с разным числом ядер

представляет результаты серии экспериментов с размерностью по 64 тестовых образца и размером батча 8, проверяющих предположение о возможности положительного эффекта от загрузки неиспользованных ресурсов обработкой дополнительных задач.

Таблица 2. Суммарное время в условных тактах одновременного выполнения задач обратного распространения ошибки

Число задач	Архитектура ППВС, количество вычислительных ядер							
	1	2	4	8	16	32	64	128
1	1 775 906	946 208	552 174	387 406	295 255	225 948	209 983	170 786
2	3 550 844	1 839 441	1 089 083	724 305	545 701	334 532	308 979	218 021
4	7 098 778	3 677 397	2 179 446	1 446 578	1 027 156	531 825	461 685	334 790

На основании представленных в таблице 2 данных построен график (рисунок 6), демонстрирующий «удельную» эффективность выполнения

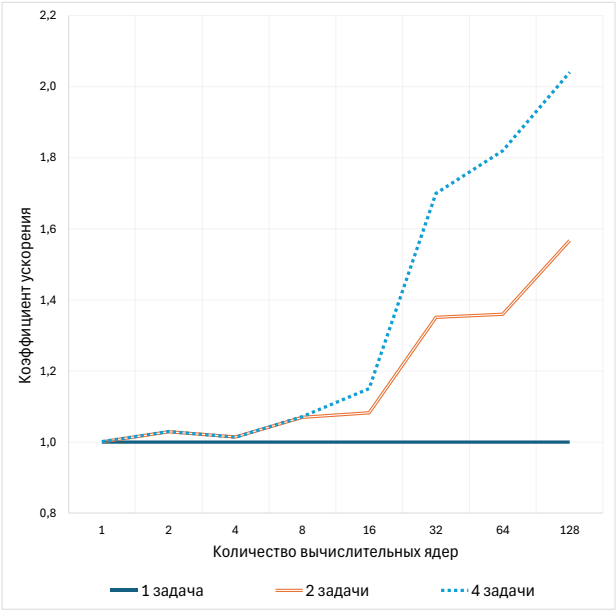

Рисунок 6. «Удельная» эффективность одновременного выполнения 1/2/4 задач обратного распространения ошибки на ППВС с разным числом ядер

задач в условиях многозадачности. Показатель рассчитан по формуле

$$Z_M^N = T_1^N \Big/ \frac{T_M^N}{M}, \quad \text{где} \tag{3}$$

M – количество одновременно выполняемых задач одинаковой размерности,

N – число вычислительных ядер,

T_M^N – время выполнения M задач на архитектуре ППВС «Буран» с N вычислительными ядрами,

Z_M^N – «удельная» эффективность одновременного выполнения M задач на N вычислительных ядрах относительно последовательного выполнения M задач на том же числе ядер.

Эффективность выполнения задач в многозадачном режиме характеризуется показателем «удельной» эффективности, который отражает соотношение суммарного времени параллельного выполнения нескольких задач к суммарному времени их последовательного выполнения. Этот показатель позволяет оценить достаточность вычислительных ресурсов для решения одной задачи, а также возможность увеличения количества одновременно выполняемых задач при имеющихся ресурсах. Анализ графика, представленного на рисунке 6, показывает, что до четырех ядер «удельная» эффективность параллельного выполнения задач близка к единице, что свидетельствует о полном использовании доступных ресурсов для выполнения задачи размером 8 слоев с 8 нейронами в каждом слое и 64 тестовыми образцами.

С увеличением количества ядер до восьми наблюдается повышение эффективности выполнения нескольких задач, что указывает на более рациональное использование вычислительных ресурсов. Однако следует отметить, что время выполнения каждой отдельной задачи увеличивается с ростом числа параллельно исполняемых задач, так как все задачи начинаются и завершаются практически одновременно. Поэтому представленная оценка эффективности имеет ограниченную область применения, что будет отражено в выводах.

Третьим элементом экспериментальной части исследования является верификация эффективности выбранной функции распределения. Представленные в таблице 3 результаты демонстрируют преимущество применения стандартной функции, обеспечивающей равномерное распределение данных. Это достигается путем генерации следующего по порядку номера вычислительного ядра при обработке следующего по значению контекста (а именно полей L и i контекста).

Результаты моделирования, проведенного при распределении данных исключительно по конкретному полю контекста `FLD(L)` или `FLD(i)`, а также

Таблица 3. Время (в условных тактах) выполнения задачи обратного распространения ошибки с разными функциями распределения

Функции распределения	Архитектура ППВС, количество вычислительных ядер							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
STD(L,i)	14 008 994	7 342 211	4 331 266	2 933 746	2 123 807	1 557 581	1 340 374	785 593
FLD(L)	14 008 994	11 446 996	8 045 655	5 321 602	3 809 108	3 851 363	3 806 283	3 806 283
FLD(i)	14 008 994	8 727 130	6 086 032	4 768 805	4 114 216	4 114 885	4 114 263	4 114 263
ZIP(L,i)	14 008 994	8 727 130	5 637 964	4 098 392	4 172 340	3 684 011	3 549 927	3 571 432

при блочном распределении по двум полям `Zip(L, i)`, подтверждают теоретические положения, изложенные в работе [1]. Функции `FLD` и `ZIP` характеризуются свойством локализации данных «порциями», что в контексте задачи прямого и обратного прохода по графу означает последовательную концентрацию данных, относящихся к текущему слою, в одной группе вычислительных ядер при распределении по полю L. При переходе на следующий или предыдущий слой данные концентрируются в другой группе вычислительных ядер.

При распределении по полю i наблюдается снижение эффективности из-за увеличения объема пересылок между ядрами, что обусловлено частыми изменениями значения поля и, в частности, обнулением данного поля. Обнуление поля приводит к концентрации всего потока данных, направленного на один узел, в одном вычислительном ядре, что нарушает равномерность распределения нагрузки на вычислительные ресурсы.

На основании анализа результатов проведенных экспериментов можно сделать следующие выводы. Во-первых, значительное увеличение размера батча не приводит к пропорциональному ускорению выполнения задачи, однако даже минимальное увеличение его размера существенно повышает эффективность решения задачи обратного распространения ошибки. Во-вторых, модель вычислений и архитектура ППВС предусматривают возможность масштабирования системы до практически любого объема за счет аппаратного распределения вычислительных процессов. Это позволяет использовать ППВС для параллельного выполнения нескольких задач без необходимости дополнительной подготовки программы для работы в многозадачном режиме, в отличие от, например, *MPI*, где программист должен учитывать объем выделенных аппаратных ресурсов для оптимизации алгоритма. В результате параллельное выполнение нескольких задач может сократить их суммарное время выполнения, причем эффективность многозадачного режима увеличивается с ростом количества вычислительных ядер в архитектуре ППВС.

Заключение

Перед началом работ по оценке эффективности выполнения нейросетевых задач в рамках модели вычислений с управлением потоком данных авторами был проведен анализ актуальности описанной работы, учитывая текущий уровень развития исследований в области искусственного интеллекта и традиционных подходов к решению подобных задач.

Dataflow системы начали активно применяться для решения нейросетевых задач. Однако большинство существующих систем этого типа представляют собой специализированные устройства, реализующие концепцию статического dataflow графа, который формируется в результате компиляции стандартной программы. Применение таких систем продемонстрировало высокую эффективность использования концепции управления потоками данных, особенно в контексте задач инференса.

Выше представлен алгоритм обучения многослойного персептрона, реализованный в концепции динамического dataflow, и предназначенный для выполнения на универсальной системе, поддерживающей управление вычислениями потоком данных. Проведенные исследования подтвердили высокую эффективность предложенного подхода к решению задач инференса, а также выявили небольшое преимущество при решении задач обучения.

Несмотря на относительную простоту представленного в статье алгоритма по сравнению с современными разработками, полученные результаты и продемонстрированные подходы могут представлять интерес для профильных специалистов. Эти материалы могут быть использованы для оценки потенциала применения нестандартных методов в области программирования и выполнения задач, связанных с нейронными сетями.

В рамках исследования выявлены следующие преимущества перехода к решению нейросетевых задач в модели вычислений с управлением потоком данных:

1. Простота и универсальность программного кода:
   - Программирование в парадигме потока данных существенно отличается от традиционного подхода, который может восприниматься как более сложный. Однако программный код потоковой программы является компактным и универсальным, не требующим регулярной адаптации под размерность задачи или доступные вычислительные ресурсы. Размер персептрона определяется исключительно размерностью полей контекста, а распределение вычислений осуществляется аппаратурой в соответствии с заранее заданной функцией распределения, которая может изменяться независимо от программного кода.

- Ключевой особенностью текущей реализации алгоритма обратного распространения ошибки является отсутствие библиотечных функций и использование исключительно простых операций сложения, умножения и сравнения. Даже такая низкоуровневая реализация не оказала негативного влияния на компактность программного кода. Более того, потоковая модель вычислений, реализованная в ППВС, позволяет разрабатывать потоковые программы, состоящие только из формирования начальных данных, без необходимости составления программного кода.
  Примером подобной реализации является работа [**16**], где программный код для задачи перемножения матриц состоит всего из одного узла выдачи результата на XOCT. Учитывая, что в алгоритмах прямого и обратного распространения также присутствуют операции умножения и сложения произведений, можно утверждать о возможности создания соответствующих потоковых программ без программного кода, где алгоритм обработки данных задается с помощью типов токенов при формировании начальных данных.

2. Перспективы применения в системах реального времени:
   - Полученные экспериментальные данные позволяют говорить о перспективности применения систем с управлением потоком данных и алгоритма прямого распространения в системах реального времени для обработки непрерывно поступающих данных, таких как распознавание лиц, номеров автомобилей и т.п. При этом обеспечивается возможность простого аппаратного и программного масштабирования.
   - Системы с управлением потоком данных не продемонстрировали значимого преимущества по сравнению с традиционными подходами в контексте обучения, что связано с невозможностью аппаратного разрешения итерационной зависимости, заложенной в алгоритме обратного распространения ошибки. Однако, применительно к динамическому dataflow можно говорить о возможности обновления весов по результатам обработки одних образцов асинхронно с выполнением прямого прохода последующих образцов.
     Это не только поможет избавиться от вынужденных простоев между батчами, но и позволит устранить саму необходимость делить набор тестовых образцов на батчи. Подобный подход, являющийся промежуточным между методами стохастического и пакетного градиентного спуска, зависит от относительных времен выполнения отдельных операций и требует отдельного исследования.
   - Кроме того, преимущество применения потоковых систем для обучения нейронных сетей проявляется в режиме многозадачности, где аппаратное распределение вычислений позволяет эффективно загрузить доступные вычислительные ресурсы несколькими независимыми задачами обучения.

Список использованных источников

[1] 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 ↑238, 240, 242, 248, 251, 252, 258, 260, 262, 265

[2] Википедия *Стохастический градиентный спуск.* ↑238

[3] *IPU hardware overview*, Graphcore, https://docs.graphcore.ai/projects/ipu-programmers-guide/en/latest/about_ipu.html. ↑238

[4] Prabhakar R., Sivaramakrishnan R., Gandhi D., Du Y., Wang M., Song X., Zhang K., Gao T., Wang A., Li X., Sheng Y., Brot J., Sokolov D., Vivek A., Leung C., Sabnis A., Bai J., Zhao T., Gottscho M., Jackson D., Luttrell M., Shah M. K., Chen Z., Liang K., Jain S., Thakker U., Huang D., Jairath S., Brown K. J., Olukotun K. *SambaNova SN40L: Scaling the AI memory wall with dataflow and composition of experts* // *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Austin, TX, USA, 02–06 November 2024).– IEEE.– 2024.– ISBN 979-8-3503-5057-9.– Pp. 1353–1366. ↑238

[5] Lie S. *Cerebras architecture deep dive: first look inside the hardware/software co-design for deep learning* // IEEE Micro.– 2023.– Vol. **43**.– No. 3.– Pp. 18–30. ↑238

[6] Abts D., Ross J., Sparling J., Wong-VanHaren M., Baker M., Hawkins T., Bell A., Thompson J., Kahsai T., Kimmell G., Hwang J., Leslie-Hurd R., Bye M., Creswick E. R., Boyd M., Venigalla M., Laforge E., Purdy J., Kamath P., Maheshwari D., Beidler M., Rosseel G., Ahmad O., Gagarin G., Czekalski R., Rane A., Parmar S., Werner J., Sproch J., Macias A., Kurtz B. *Think Fast: A Tensor Streaming Processor (TSP) for accelerating deep learning workloads* // *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (Valencia, Spain, 30 May 2020–03 June 2020).– IEEE.– 2020.– ISBN 978-1-7281-4661-4.– Pp. 145–158. ↑238

[7] 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. ↑239

[8] 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. ↑239

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

[10] Böhm A. P. W. *Dataflow and hybrid dataflow architecture summary* // *Parallel Computer Systems: Performance Instrumentation and Visualization*, eds. R. Koskela, M. Simmons, New York: ACM.– 1990.– ISBN 978-0-201-50937-3.– Pp. 281–286. ↑240

[11] Arvind, Brobst S. *The evolution of dataflow architectures: from static dataflow to P-RISC* // International Journal of High Speed Computing.– 1993.– Vol. **05**.– No. 02.– Pp. 125–153. ↑240

[12] Lee B., Hurson A. R. *Dataflow architectures and multithreading* // Computer.– August 1994.– Vol. **27**.– No. 8.– Pp. 27–39. ↑240

[13] Климов А. В., Левченко Н. Н., Окунев А. С., Стемпковский А. Л. *Вопросы применения и реализации потоковой модели вычислений* (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС).– 2016.– № 2.– С. 100–106. ↑240

[14] Змеев Д. Н., Климов А. В., Левченко Н. Н. *Средства распределения вычислений в ППВС «Буран» и варианты реализации блока выработки хэш-функций* (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2016.– № 2.– С. 107–113. ↑242

[15] 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. ↑242

[16] Змеев Д. Н., Окунев А. С. *Разработка и исследование алгоритма задачи перемножения разреженных матриц для параллельной потоковой вычислительной системы «Буран»* (Москва, Зеленоград, Россия, 1 октября–5 октября 2018 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2018.– № 3.– С. 16–23. ↑242, 267

[17] Змеев Д. Н., Левченко Н. Н. *Особенности создания параллельных программ в потоковой парадигме программирования* // Информационные технологии.– 2022.– Т. **28**.– № 11.– С. 597–606. ↑242

[18] 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. ↑242

[19] Змеев Д. Н. *Средства проектирования высокопроизводительных потоковых вычислительных систем* (Москва, Зеленоград, Россия, 3 октября–7 октября 2016 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2016.– № 2.– С. 159–163. ↑260

Поступила в редакцию 30.06.2026;
одобрена после рецензирования 30.07.2026;
принята к публикации 31.07.2026;
опубликована онлайн 18.09.2026.

Рекомендовал к публикации *к.ф.-м.н. С. А. Романенко*

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: *n2lev@yandex.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 HARDWARE AND SOFTWARE FOR DISTRIBUTED AND SUPERCOMPUTER SYSTEMS

UDC 004.021+004.272
10.25209/2079-3316-2026-17-3-237-273

Backpropagation algorithm in the dataflow paradigm

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

[1–3] National Research Center "Kurchatov Institute", Moscow, Russia

[1]✉*zmejevdn@list.ru*, [2]✉*n2lev@yandex.ru*, [3]✉*arkady.klimov@gmail.com*

Abstract. The article discusses the issue of developing and implementing backpropagation 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 intrinsically parallel, since the task parallelism is extracted automatically from the data flow at the hardware level. Programs created in the dataflow paradigm are initially parallel. The article provides a detailed description of the backpropagation dataflow algorithm and the program that runs on the parallel dataflow computing system (PDCS) «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 training perceptrons of any dimension without modification and recompilation of the program code. The size and structure of the trainable perceptron is determined by the initial data.

The experimental part of the article presents the results of PDCS behavior study when executing the backpropagation program, analyzes the effect of the trainable batch size on the overall efficiency of the dataflow program, and evaluates the use of various methods of hardware distribution of computations. In addition, the potential of using the PDCS for parallel execution of several training tasks at the same time is considered. (*In Russian*).

Key words and phrases: backpropagation algorithm, 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. *Backpropagation algorithm in the dataflow paradigm*. Program Systems: Theory and Applications, 2026, **17**:3(72), pp. 237–273. (*In Russ.*). https://psta.psiras.ru/read/psta2026_3_237-273.pdf

References

[1] D. N. Zmeev, N. N. Levchenko, A. V. Klimov. “Forward propagation algorithm in the dataflow paradigm”, *Program Systems: Theory and Applications*, **16**:4(67) (2025), pp. 51–79 (in Russian).

[2] Vikipediya. *Stochastic gradient descent* (in Russian).

[3] *IPU hardware overview*, Graphcore, https://docs.graphcore.ai/projects/ipu-programmers-guide/en/latest/about_ipu.html.

[4] R. Prabhakar, R. Sivaramakrishnan, D. Gandhi, Y. Du, M. Wang, X. Song, K. Zhang, T. Gao, A. Wang, X. Li, Y. Sheng, J. Brot, D. Sokolov, A. Vivek, C. Leung, A. Sabnis, J. Bai, T. Zhao, M. Gottscho, D. Jackson, M. Luttrell, M. K. Shah, Z. Chen, K. Liang, S. Jain, U. Thakker, D. Huang, S. Jairath, K. J. Brown, K. Olukotun. “SambaNova SN40L: Scaling the AI memory wall with dataflow and composition of experts”, *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Austin, TX, USA, 02–06 November 2024), IEEE, 2024, ISBN 979-8-3503-5057-9, pp. 1353–1366.

[5] Lie S.. “Cerebras architecture deep dive: first look inside the hardware/software co-design for deep learning”, *IEEE Micro*, **43**:3 (2023), pp. 18–30.

[6] D. Abts, J. Ross, J. Sparling, M. Wong-VanHaren, M. Baker, T. Hawkins, A. Bell, J. Thompson, T. Kahsai, G. Kimmell, J. Hwang, R. Leslie-Hurd, M. Bye, E. R. Creswick, M. Boyd, M. Venigalla, E. Laforge, J. Purdy, P. Kamath, D. Maheshwari, M. Beidler, G. Rosseel, O. Ahmad, G. Gagarin, R. Czekalski, A. Rane, S. Parmar, J. Werner, J. Sproch, A. Macias, B. Kurtz. “Think Fast: A Tensor Streaming Processor (TSP) for accelerating deep learning workloads”, *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (Valencia, Spain, 30 May 2020–03 June 2020), IEEE, 2020, ISBN 978-1-7281-4661-4, pp. 145–158.

[7] 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.

[8] 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.

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

[10] 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.

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

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

[13] A. V. Klimov, N. N. Levchenko, A. S. Okunev, A. L. Stempkovskij. “The application and implementation issues of dataflow computing system” (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), *Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem (MES)*, 2016, no. 2, pp. 100–106 (In Russian).

[14] D. N. Zmeev, A. V. Klimov, N. N. Levchenko. “The tools for computation distribution in the PDCS "Buran" and hash-functions block implementation options” (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), *Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem*, 2016, no. 2, pp. 107–113 (In Russian).

[15] A. L. Stempkovskij, N. N. Levchenko, A. S. Okunev, V. V. Cvetkov. “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).

[16] D. N. Zmeev, A. S. Okunev. “Development and investigation of algorithm of sparse matrices multiplication task for the parallel dataflow computing system "Buran"” (Moskva, Zelenograd, Rossiya, 1 oktyabrya–5 oktyabrya 2018 g.), *Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem*, 2018, no. 3, pp. 16–23 (In Russian).

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

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

[19] D. N. Zmeev. “Design tools of high-performance dataflow computing systems” (Moskva, Zelenograd, Rossiya, 3 oktyabrya–7 oktyabrya 2016 g.), *Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem*, 2016, no. 2, pp. 159–163 (In Russian).