УДК 004.272 Научная статья

https://doi.org/2071-8985-2022-15-1-22-29

Оптимизация выполнения целочисленных арифметических операций на основе нейронных сетей

О. Т. Мохаммед, А. А. Пазников✉

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), Санкт-Петербург, Россия

✉ apaznikov@gmail.com

Аннотация. В настоящее время основные методы повышения производительности операций над числами с фиксированной и плавающей точками, выполняемых многоядерными вычислительными системами (ВС), представлены усложнением микроархитектуры систем и алгоритмов, реализованных в компиляторах и библиотеках. Отдельно следует выделить операции с длинными числами, которые превышают размер машинного слова, что обусловлено ограниченным размером регистров процессора и шины данных. В результате реализация традиционных операций – сложения, умножения, деления, в современных ВС усложняется. Для решения данной задачи в настоящий момент существует ряд алгоритмов, которые, однако, могут быть достаточно медленными, поскольку они последовательные и реализуют побитовое выполнение операций над числами. Для повышения производительности данного класса вычислений предложена модель на основе нейронной сети. Цель создания этой модели – уменьшение времени выполнения операций и загрузки центрального процессора. Разработана нейросетевая модель для выполнения арифметических операций. Предложен и экспериментально проверен метод генерации набора данных на основе обхода графа. Построены алгоритмы выполнения арифметических операций на основе предварительно обученной нейронной сети.

Ключевые слова: нейронные сети, машинное обучение, длинная арифметика, параллельные вычисления

Для цитирования: Мохаммед О. Т., Пазников А. А. Оптимизация выполнения целочисленных арифметических операций на основе нейронных сетей // Изв. СПбГЭТУ «ЛЭТИ». 2022. Т. 15, № 1. С. 22–29. doi: 10.32603/2071-8985-2022-15-1-22-29.

Конфликт интересов. Авторы заявляют об отсутствии конфликта интересов

Original article

Optimizing computation of arithmetic operations based on a pre-trained neural network

O. T. Mohammed, A. A. Paznikov✉

Saint Petersburg Electrotechnical University, Saint Petersburg, Russia

✉ apaznikov@gmail.com

Abstract. Nowadays, the main methods of improving the performance of arithmetic operations on fixed and floating numbers are performed by multicore computer systems (CS) are the improvement of the microarchitecture of systems and algorithms implemented in compilers and libraries. Separately, it is necessary to highlight the complex tasks of implementing operations with long numbers that exceed the size of a machine word. This is due to the limited size of the processor registers and the data bus. As a result, the implementation of traditional operations, such as addition, multiplication, division, become more difficult to implement in modern computer systems. To solve this problem, there are currently several algorithms. However, the available algorithms can be quite slow, since they are sequential and implement bitwise operations on the bits of a number.

In order to improve the performance of this class of calculations, a model based on a neural network is proposed. The purpose of this model is to reduce the execution time and CPU usage. A neural network model for performing arithmetic operations has been developed. A method of generating a data set based on graph traversal is proposed and tested experimentally. Algorithms for performing arithmetic operations based on a pretrained neural network are constructed.

Keywords: neural networks, machine learning, long arithmetic operations, parallel computing

For citation: Mohammed O. T., Paznikov A. A. Optimizing integer arithmetic operations based on neural networks // LETI Transactions on Electrical Engineering & Computer Science. 2022. Vol. 15, no. 1. P. 22–29. doi: 10.32603/2071-8985-2022-15-1-22-29.

Conflict of interest. The authors declare no conflicts of interest

Введение. Современные вычислительные системы обладают высокой производительностью при выполнении вычислений с числами традиционной размерности (32 или 64 бит) [1]. Однако при увеличении размерности чисел растет и время выполнения таких операций, как сложение, вычитание, умножение, деление и т. д. Данная проблема возникает вследствие аппаратных ограничений микроархитектуры процессора, а именно размера машинного слова [2]. Большинство стандартных типов данных в языках программирования не превосходят 64 бит (например, int, float, double в C/C++). Однако часто возникает потребность выполнения операций с числами, превосходящими данный размер. В настоящее время существует значительное число методов для выполнения длинной арифметики, большинство из них основаны на схожих принципах: во-первых, они представляют длинные числа как двоичные значения, после чего выполняют побитовые операции с двоичным представлением чисел [3], [4]. Например, арифметическое сложение может быть реализовано с помощью побитовых операций XOR или OR. Эти алгоритмы имеют сложность $O(n)$, где n представляет собой общее количество битов, составляющих каждый из операндов.

В данной работе мы предлагаем модель машинного обучения, обученную на заданном наборе чисел, для предсказания точного результата конкретной арифметической операции. Цель этой модели – уменьшение времени выполнения и загрузки центрального процессора.

Статья организована следующим образом. Часть 2 посвящена существующим работам, связанным с разработкой алгоритмов для выполнения длинной арифметики. В ч. 2 рассматривается созданный метод генерации набора данных выполненных математических вычислений с помощью техники итерации узлов графа, описан пример сложения со схемой, которая иллюстрирует все арифметические операции (сложение, умножение, вычитание и деление). Описывается предлагаемый подход, заключающийся в выполнении арифметических вычислений с помощью предварительно обученной модели нейронной сети, которая может частично повысить производительность выполнения некоторых повторяющихся арифметических операций. Детально представлено выполнение операции сложения. В ч. 3 представлен экспериментальный анализ точности нейронной сети на примере вычисления числа «пи». Проводится сравнительный анализ времени вычисления числа «пи» при традиционном выполнении арифметических операций и предварительно обученной моделью после получения экспериментальных результатов. Приводятся результаты моделирования вычисления числа «пи» этими способами, показано различие между ними с точки зрения времени и полученных результатов. Представлены некоторые ограничения предлагаемого нами подхода. В заключении даны основные выводы по статье и план будущих работ.

1. Обзор существующих работ. В настоящий момент существует ряд библиотек, направленных на решение проблемы выполнения длинной целочисленной арифметики. Большинство из них не поддерживают распараллеливание и предназначены для работы в одноядерных системах [5]–[7]. Кроме того, подавляющая часть работ в данной области направлены на аппаратную оптимизацию выполнения операций.

Например, в [8] предлагается усовершенствовать сумматор с упреждающим переносом (look-ahead adder). В данном подходе используются распределенные вычислительные системы с массовым параллелизмом. Для распределения целых чисел между процессорами используется операция префиксной суммы, позволяющая выполнять

сложение больших чисел быстрее, чем если бы были установлены обычные машины.

В [9] предложена схема представления двоичных чисел, используемых в параллельных вычислениях. Чтобы избавиться от распространения последовательного переноса во время выполнения операций сложения и вычитания, в основу подхода положен метод репликации каждого входного операнда [10], [11].

Мы хотели бы подчеркнуть, что для решения этой проблемы было предложено множество решений, однако большинство из них выполняют арифметические вычисления на битовом уровне с использованием побитовых операций [12], [13]. Обычно такие алгоритмы имеют сложность $O(n)$: основная процедура выполняется n раз, что в конечном итоге равно количеству битов во входных данных большого целого числа.

2. Предлагаемый метод и модель. ***2.1. Метод обхода графа.*** Опишем предлагаемый нами метод генерации набора данных (dataset). Суть его заключается в том, что генерируется направленный граф операндов и результатов в виде узлов графа и ребер для направлений операций соответственно. Для каждой операции выполняется проход сгенерированного графа и поиск результата. Результат операции возвращается всякий раз, когда он был найден. Если результат не найден, это означает, что мы не выполняли такую операцию раньше, поэтому ее нужно выполнить традиционным способом и сохранить операнды и результаты как новые узлы в графе с указанными направленными ребрами для последующего использования.

После этого графы сохраняются в общей памяти с высокой скоростью доступа. На данном этапе возможна некоторая оптимизация, например предварительная загрузка (prefetching) данных в разделяемую кэш-память. Затем нужно построить эффективный алгоритм для обхода сгенерированного графа за минимальное время. Важно отметить, что каждый граф имеет два списка указателей: список направленных операций и список указателей на результаты.

В качестве примера проведем операцию для 1 + 2 = 3 (рис. 1). В этом примере мы находим первый рабочий узел в графе, который равен 1, после этого находим указатель на операцию (сложение «+») и второй операнд (узел 2). Затем из второго узла отыскивается указатель, который соответствует ребру «res + (from 1)», указывающему на результат операции, и получаем результат операции – узел «3». Таким образом отыскивается результат операции 1 + 2 = 3 обходом графа. При помощи описанного метода обхода узлов

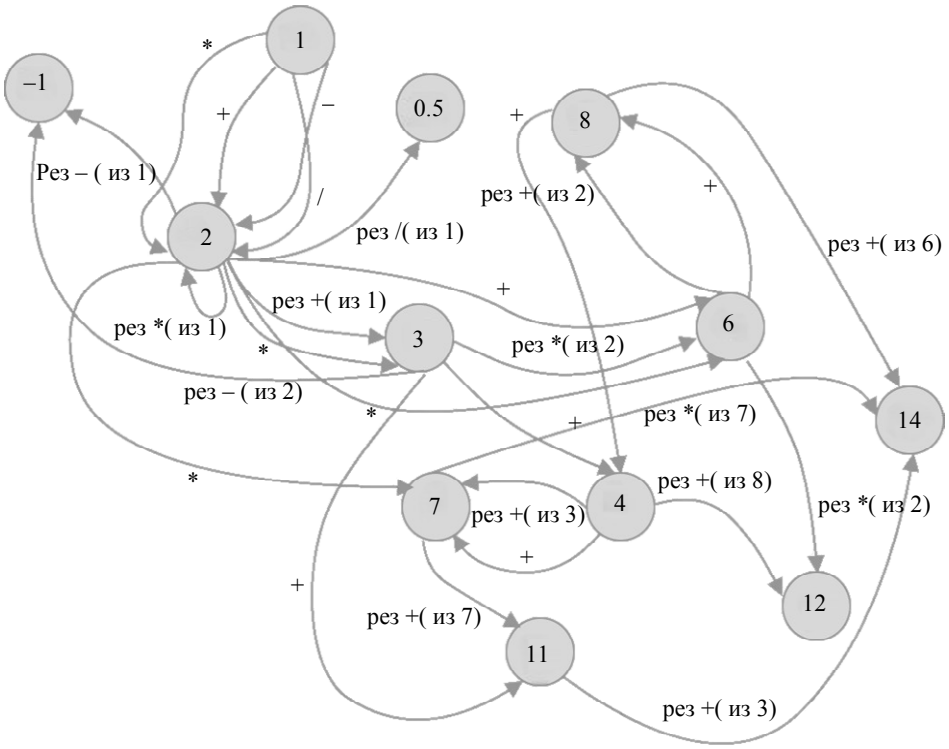


Рис. 1. Пример графа для выполнения арифметических операций
Fig. 1. Example of graph for arithmetical operations

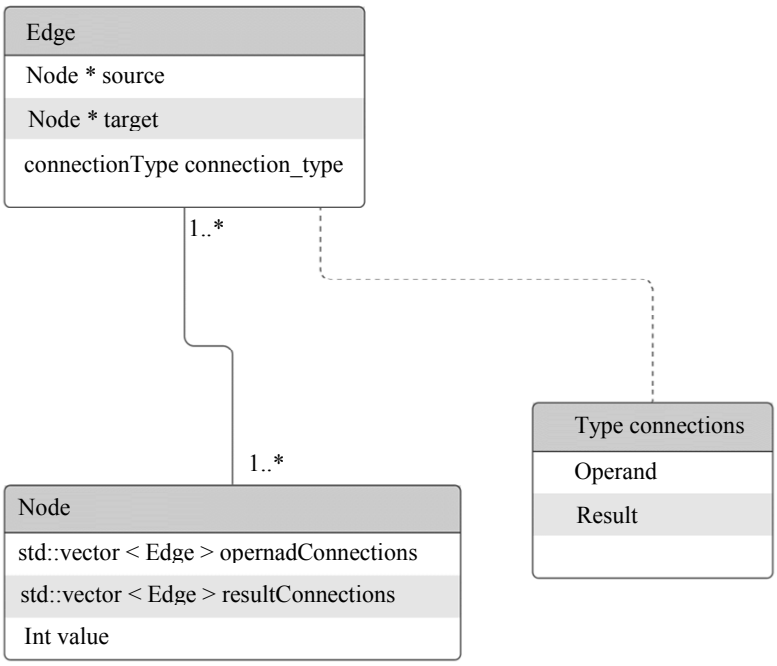


Рис. 2. Диаграмма классов
Fig. 2. Class diagram

было сгенерировано до 1 млн записей с соответствующими результатами всех арифметических операций. На рис. 2 представлен алгоритм генерации динамического набора данных:

Входные данные: Сгенерированный набор данных из графа обхода;
Выходные данные: Получить результат операции;
Получить операцию и операнды из входных данных;
if существует ли уже точный результат **then**;
найти кратчайший путь к результату;
получить значение;
else;
добавить новый узел;
выполнить операцию и добавить результат как узел результата;
добавить ребро между операндом и операционным узлом;
обновить набор данных графа новым результатом;
end if.

2.2. Модель глубокого обучения. Искусственные нейронные сети и модели глубокого обучения к самым мощным инструментам прогнозирования, которые может предложить машинное обучение. В данной статье мы попытались использовать данный инструментарий для оптимизации выполнения часто выполняющихся операций с использованием длинной арифметики и предложили соответствующий алгоритм.

Приведем пример: центральный процессор (ЦП) может легко сложить два обычных числа, в то время как вычисление суммы двух больших чисел, превышающих 64-битовый целочисленный диапазон значений, требует иного представления чисел (например, в виде побитового массива) и более сложных алгоритмов (и, как следствие, более трудоемких) для выполнения арифметических операций.

Для простоты понимания: в данной статье мы обучили нейронную сеть вычислению арифметических операций с двумя числами, а затем попытались предсказать результат операций. Структура нашего набора данных состоит из двух входов и одного выхода. Модель была обучена на 1 млн записей целочисленного типа с соответствующими целевыми выходами от 0 до 10^6 и была протестирована для получения точных результатов для целых чисел в диапазоне от 10^6 до 10^7 результатов независимо от самого обученного набора.

В результате мы создали модель, способную точно предсказать арифметическое вычисление двух целых чисел в диапазоне от нуля до 10^7, которая готова избавить ЦП от выполнения арифметических вычислений в этом диапазоне, особенно когда ЦП требуется выполнять большое количество аналогичных задач неоднократно.

Следует отметить, что время, необходимое ЦП для суммирования двух небольших целых чисел, незначительно превышало время выполнения операций на основе созданной модели для

прогнозирования вычисления тех же чисел. Тем не менее, основная цель заключается не в том, чтобы конкурировать с возможностями процессора, а в том, чтобы помочь работать быстрее.

На рис. 3 приведен пример нейронной сети без скрытых слоев и с функцией идентичности $f(x) = x$ в качестве функции активации. Следует отметить, что данную нейронную сеть можно представить в виде линейной регрессии. Сеть обучается выполнению операций посредством поиска весовых коэффициентов W длиной N. Сети также могут обучаться арифметическим операциям, однако линейная регрессия уже не подходит для этой задачи.

В данной статье мы использовали более сложные структуры нейронных сетей, а именно многоуровневый персептрон (multi-layer perceptrone, MLP), имеющий входной слой, три скрытых слоя, каждый из которых имеет 25 нейронов, и выходной слой.

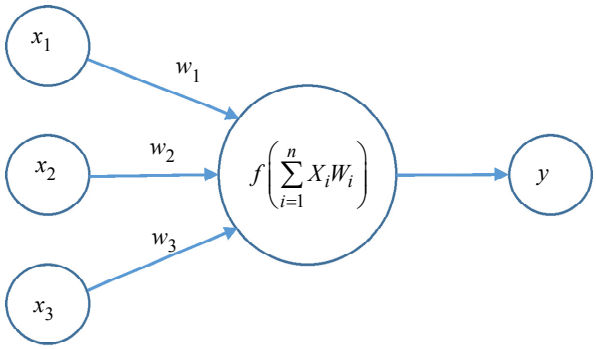


Рис. 3. Пример нейронной сети
Fig. 3. Neural network example

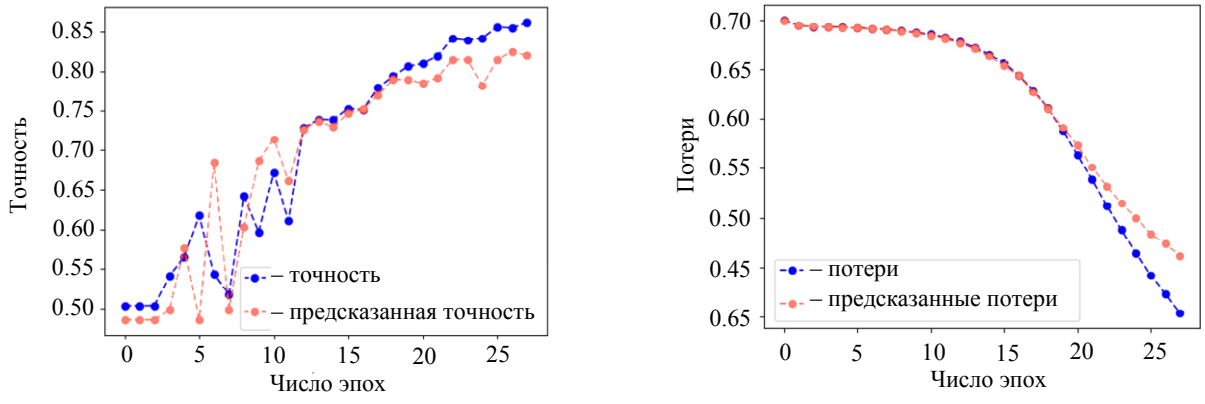
Нейронная сеть прямого распространения функционирует по следующему алгоритму:

Входные данные: подготовленный набор данных;

Выходные данные: обученная нейронная сеть;
инициализировать нейронную сеть;
настроить входные, скрытые и выходные слои;
настроить связи между слоями;
while epoch < max_epoch **do**;
получить обучающую и целевую выборки из набора данных;
if обучающая выборка ≠ размер входного слоя **then**;
прервать выполнение цикла;
else;
выполнить прямое распространение для получения прогнозируемого вывода;
вычислить разницу между прогнозируемым выходом и целевой выборкой, чтобы получить значение потерь;
выполнить обратное распространение, чтобы минимизировать значение потерь;
end if;
epoch++;
end while.

3. Результаты экспериментов. ***3.1. Обучение и оценка модели нейронной сети.*** Сгенерированный набор данных был разделен таким образом, что 80 % данных были назначены для обучения, а остальные 20 % – для тестового набора. Набор обучающих данных подается в четырехуровневую нейронную сеть, причем первые два слоя содержат по двадцать пять нейронов каждый, а выходной слой – один нейрон.

На рис. 4 представлены графики: точности обучения и прогнозируемой точности (рис. 4, *а*); потерь при обучении и прогнозируемых потерь нашей обученной модели (рис. 4, *б*) – в зависимости от числа эпох. «Точность обучения» представляет собой точность прогнозируемых выходных данных обучающих выборок в наборе дан-



а *б*

Рис. 4. Значения точности для модели нейронной сети:
а – точность обучения и предсказанная точность; *б* – потери и предсказанные потери
Fig. 4. Accuracy and loss of the model after training the model:
а – accuracy graph of the trained model; *б* – loss graph of the trained model

ных, а «прогнозируемая точность» – это прогнозируемая точность тестовых выборок в наборе данных. С другой стороны, потеря при обучении – это представление значения потерь за одну эпоху (одна итерация обучающих выборок), в то время как «прогнозируемая потеря» отображает значение потерь за одну эпоху тестовых выборок в наборе данных. Стоит отметить, что конечная цель любой нейронной сети – повысить точность и уменьшить потери.

Полученную точность более 87 % можно считать положительной для модели, обученной всем арифметическим операциям. График на рис. 4, *б* показывает потери при обучении по сравнению с прогнозируемыми потерями за одинаковое количество эпох. Как показано на основе архитектуры нашей модели, значение потерь или ошибок уменьшается до 0.12, что также является удовлетворительным результатом. Тем не менее, следует отметить, что рассматриваемые показатели могут быть оптимизированы при новых архитектурных решениях.

3.2. Экспериментальное исследование на примере вычисления числа π. При исследовании созданной модели для простоты использовался метод вычисления числа π при помощи уравнения быстро сходящегося ряда

$$\frac{1}{\pi}=\frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k)!(1103+26390k)}{(k!)^4\left(396^{4k}\right)}.$$

Одним из часто используемых методов для вычисления числа π является вычисление с помощью ряда Грегори–Лейбница (мы считаем его интересным примером для экспериментального моделирования построенной модели, хотя многие арифметические операции в нем выполняются неоднократно):

$$\pi=\frac{4}{1}-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\frac{4}{9}-\frac{4}{11}+\frac{4}{13}-\frac{4}{15}.$$

После обучения модели на входных данных (10^6 чисел) мы измерили среднее время выполнения одного вычисления на вышеописанной задаче. Эксперимент проводился на базе операционной системы Windows 10 с использованием библиотеки timeit на языке программирования Python. Вычислительная система была укомплектована процессором Intel Core i7-8750H 2.2 ГГц (12 процессорных ядер), 16 Гбайт оперативной памяти.

Результаты экспериментов представлены в таблице. Время, которое потребовалось ЦП для выполнения этого вычисления, в среднем на 0.26 нс больше, чем время выполнения нашей модели на основе нейронной сети. Мы считаем, что причина получения таких результатов связана с методом, используемым ЦП для вычисления операций, который характеризуется большей сложностью в сравнении с нашим методом нейронной сети.

Результаты моделирования
Experimental results

Реализация	Результат	Среднее время выполнения, нс
CPU	3.141592653589793	9.99
Предварительно обученная модель AI	3.141592653589793	9.73

В качестве ограничений применимости нашего метода следует отметить, что модель должна быть точно и специально обучена, при этом в процессе обучения требуется применение больших объемов структурированных обучающих данных и обучение должно контролироваться. Кроме того, для обучения требуются автономные пакеты данных. Также отметим, что сеть не может эффективно обучаться в реальном времени.

Заключение и план дальнейших исследований. Разработана нейросетевая модель для выполнения арифметических операций. Предложен и проверен экспериментально метод генерации набора данных на основе обхода графа. Построены алгоритмы выполнения арифметических операций на основе предварительно обученной нейронной сети. Мы видим, что наш общий набор данных имеет некоторые ограничения, поскольку он не может охватить все возможные целые числа, чем больше целых чисел мы охватываем, тем более сложной станет модель. В результате проведенных экспериментов время выполнения операций ЦП в среднем превышает на 0.26 нс время выполнения операций на основе построенной нейронной сети.

В заключение отметим, что использование нейронных сетей для выполнения арифметических операций является активной областью исследования, и данный проект может стать одним из первых шагов в этом направлении. Будущие исследования могут улучшить полученные нами результаты, а также добавить другие математические функции (например, тригонометрические). Кроме того, возможно распараллеливание данного подхода в многоядерных вычислительных системах с общей и разделяемой памятью.

Финансирование. Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00784.

Список литературы

1. Maxfield C., Brown A. The definitive guide to how computers do math: featuring the virtual DIY calculator. New Jersey: Hoboken John Wiley & Sons, 2005.

2. Hennessy J., Patterson D. Computer architecture: a quantitative approach. San Francisco: Morgan Kaufmann, 2006. 4rd ed.

3. Knuth D. Art of computer programming. Vol. 2: Seminumerical Algorithms. United States: Addison-Wesley Longman, 1997. 3rd ed.

4. Koren I. Computer Arithmetic Algorithms. New York: AK Peters, 2001. 2nd ed.

5. Paznikov A., Shichkina Y. Algorithms for optimization of processor and memory affinity for remote core locking synchronization in multithreaded applications // Information. 2018. Vol. 9, no. 1. P. 1–12.

6. Tabakov A., Paznikov A. Algorithms for optimization of relaxed concurrent priority queues in multicore systems // Proc. of the 2019 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). St. Petersburg, 2019. P. 360–365.

7. Paznikov A. A., Smirnov V. A., Omelnichenko A. R. Towards efficient implementation of concurrent hash tables and search trees based on software transactional memory // 2019 Intern. Multi-Conf. on Industrial Engineering and Modern Technologies (FarEastCon). Vladivostok: IEEE, 2019. P. 1–5.

8. Fagin B. Fast addition of large integers // IEEE Tr. Com. 1992. No. 41. P. 1069–1077.

9. Avizienis A. Signed-digit number representations for fast parallel arithmetic // IRE Trans. Elec. Comput. 2003. No. 10. P. 737–740.

10. Bassil Y., Barbar A. Sequential and parallel algorithms for the addition of big-integer numbers // Intern. J. of Computational Science. 2010. No. 10. P. 52–69.

11. Paznikov A. A., Gurin A. V., Kupriyanov M. S. Implementation in actor model of leaderless decentralized atomic broadcast // 2020 9th Mediterranean Conf. on Embedded Computing (MECO). Vladivostok: IEEE, 2020. P. 1–4.

12. Mohammed O., Heidari M., Paznikov A. Mathematical computations based on a pre-trained AI model and graph traversal // 9th Mediterranean Conf. on Embedded Computing (MECO). St. Petersburg, 2020. P. 1–4.

13. Towards optimization of big numbers computation through an AI pre-trained model and graph traversal / O. Mohammed, M. Heidari A. Paznikov, M. Kupriyanov // 2020 XXIII Intern. Conf. on Soft Computing and Measurement (SCM). St. Petersburg. 2020. P. 153–156.

Информация об авторах

Пазников Алексей Александрович – канд. техн. наук, доцент Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» им. В. И. Ульянова (Ленина).
E-mail: apaznikov@gmail.com
http://orcid.org/ 0000-0002-3735-6882

Мохаммед Омар Таха – аспирант Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» им. В. И. Ульянова (Ленина).
E-mail: omar.taha.mohammed@gmail.com
http://orcid.org/ 0000-0001-5939-9296

References

1. Maxfield C., Brown A. The definitive guide to how computers do math: featuring the virtual DIY calculator. New Jersey: Hoboken John Wiley & Sons, 2005.

2. Hennessy J., Patterson D. Computer Architecture: A Quantitative Approach. San Francisco: Morgan Kaufmann, 2006. 4rd ed.

3. Knuth D. Art of Computer Programming. Vol. 2: Seminumerical Algorithms. United States: Addison-Wesley Longman, 1997. 3rd ed.

4. Koren I. Computer Arithmetic Algorithms. New York: AK Peters, 2001. 2nd ed.

5. Paznikov A., Shichkina Y. Algorithms for Optimization of Processor and Memory Affinity for Remote Core Locking Synchronization in Multithreaded Applications // Information. 2018. Vol. 9, no. 1. P. 1–12.

6. Tabakov A., Paznikov A. Algorithms for Optimization of Relaxed Concurrent Priority Queues in Multicore Systems // Proc. of the 2019 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). St. Petersburg, 2019. P. 360–365.

7. Paznikov A. A., Smirnov V. A., Omelnichenko A. R. Towards efficient implementation of concurrent hash tables and search trees based on software transactional memory // 2019 Intern. Multi-Conf. on Industrial Engineering and Modern Technologies (FarEastCon). Vladivostok: IEEE, 2019. P. 1–5.

8. Fagin, B. Fast Addition of Large Integers // IEEE Tr. Com. 1992. No. 41. P. 1069–1077.

9. Avizienis A. Signed-digit number representations for fast parallel arithmetic // IRE Trans. Elec. Comput. 2003. No. 10. P. 737–740.

10. Bassil Y., Barbar A. Sequential and Parallel Algorithms For the Addition of Big-Integer Numbers // Intern. J. of Computational Science. 2010. No. 10. P. 52–69.

11. Paznikov A. A., Gurin A. V., Kupriyanov M. S. Implementation in Actor Model of Leaderless Decentralized Atomic Broadcast] // 2020 9th Mediterranean Conf. on Embedded Computing (MECO). Vladivostok: IEEE, 2020. P. 1–4.

12. Mohammed O., Heidari M., Paznikov A. Mathematical Computations Based on a Pre-trained AI Model and Graph Traversal // 9th Mediterranean Conf. on Embedded Computing (MECO). St. Petersburg, 2020. P. 1–4.

13. Mohammed O., Heidari M., Paznikov A., Kupriyanov M. Towards optimization of big numbers computation through an AI pre-trained model and graph traversal // 2020 XXIII Intern. Conf. on Soft Computing and Measurement (SCM). St. Petersburg. 2020. P. 153–156.

Information about the authors

Alexey A. Paznikov, Cand. Sci. (Eng.), Associate Professor of Saint Petersburg Electrotechnical University.
E-mail: apaznikov@gmail.com
http://orcid.org/ 0000-0002-3735-6882

Omar T. Mohammed, postgraduate student of Saint-Petersburg Electrotechnical University.
E-mail: omar.taha.mohammed@gmail.com
http://orcid.org/ 0000-0001-5939-9296

Статья поступила в редакцию 22.11.2021; принята к публикации после рецензирования 12.12.2021; опубликована онлайн 30.01.2022

Submitted 22.11.2021; accepted 12.12.2021; published online 30.01.2022.