УДК 004.31

EDN JFHHSE

Архитектура фон Неймана: актуальность, ограничения и пути развития в современных вычислениях

Д.Д. Рыбка[1*], П.М. Урвачев[2]

[1]Студентка ФГБОУ ВО «Кубанский Государственный Технологический Университет», Краснодар, Россия
[2]Старший преподаватель кафедры информационных систем и программирования ФГБОУ ВО «Кубанский Государственный Технологический Университет», Краснодар, Россия

[*]E-mail: rybkadasa8@gmail.com

Аннотация. Статья посвящена анализу актуальности и ограничений архитектуры фон Неймана в контексте современных вычислительных систем. В первом разделе представлена теоретическая основа архитектуры и сформулирована ключевая проблема – «фон-неймановское узкое место». Во втором разделе описывается методика экспериментального исследования, включающая разработку на языке C# тестов, которые сравнивают производительность при последовательном и случайном доступе к памяти, а также анализ результатов, показывающий проявление данного ограничения. На основе проведенного анализа в третьем разделе рассматриваются современные пути развития вычислительных систем, направленные на преодоление «узкого места». Статья подчеркивает, что, несмотря на свои фундаментальные ограничения, архитектура фон Неймана остается актуальной, а ее эволюция связана с развитием гибридных подходов и совершенствованием подсистемы памяти.

Ключевые слова: архитектура фон Неймана, фон-неймановское узкое место, производительность, оперативная память, кэш-память, сравнительный анализ, экспериментальное исследование, вычислительные системы.

Von Neumann architecture: relevance, limitations, and ways of development in modern computing

D.D. Rybka[1*], P.M. Urvachev[2]

[1]Student of Kuban State Technological University, Krasnodar, Russia
[2]Senior Lecturer of the Department of Information Systems and Programming Kuban State Technological University, Krasnodar, Russia

[*]E-mail: rybkadasa8@gmail.com

Abstract. The article is devoted to the analysis of the relevance and limitations of the von Neumann architecture in the context of modern computing systems. The first section presents the theoretical basis of the architecture and formulates the key problem, which is the "von Neumann bottleneck." The second section describes the methodology of the experimental study, which includes the development of specialized tests in the C# language that compare the performance of sequential and random access to memory. The third section presents and analyzes the results of the experiment, which clearly demonstrate the manifestation of this limitation. Based on the analysis, the article concludes by discussing current trends in the development of computing systems aimed at overcoming the "bottleneck." The article emphasizes that, despite its fundamental limitations, the von Neumann architecture remains relevant, and its evolution is driven by the development of hybrid approaches and improvements to the memory subsystem.

Keywords: von Neumann architecture, von Neumann bottleneck, performance, RAM, cache memory, comparative analysis, experimental research, and computing systems.

1. Введение

Архитектура фон Неймана, впервые детально описанная в 1945 году, заложила фундаментальные принципы организации практически всех современных универсальных вычислительных систем. Её ключевая концепция заключается в хранении как команд (программы), так и данных в единой оперативной памяти (ОЗУ). Этот подход стал революционным, позволив отказаться от аппаратной «прошивки» программ, требовавшей физической перекоммутации для внесения изменений.

Универсальность и долговечность архитектуры фон Неймана обеспечены тремя ключевыми принципами [1,2]:

1. Принцип однородности памяти: процессор обращается к памяти как к линейному адресному пространству, не делая различий между ячейками, хранящими команды и данные, что обеспечивает гибкость и простоту смены выполняемых задач.
2. Принцип хранимой программы: программа загружается в память перед выполнением, что делает вычислительную машину программно-управляемой.
3. Принцип последовательного программного управления: команды выполняются последовательно, а порядок их выполнения контролируется с помощью счётчика команд.

Однако та же простота и универсальность породили фундаментальное ограничение, известное как «фон-неймановское узкое место» (Von Neumann bottleneck) [3]. Проблема возникает из-за того, что процессор и память связаны единственным каналом передачи данных — системной шиной. В каждом цикле работы процессор может либо читать команду, либо читать/записывать данные. При этом производительность процессора исторически растёт экспоненциально быстрее, чем пропускная способность и скорость доступа к оперативной памяти [3,4]. В результате высокоскоростное вычислительное ядро значительную часть времени простаивает в ожидании доставки команд и данных из относительно медленной памяти, что делает это противоречие главным ограничивающим фактором общей производительности системы.

Таким образом, «фон-неймановское узкое место» представляет собой не только теоретическую проблему, но и актуальное практическое ограничение, которое проявляется при каждой выборке данных или инструкций из основной памяти. Хотя современные методы (такие как многоуровневая кэш-память, предвыборка данных и конвейеризация) направлены

на минимизацию этих задержек, они не устраняют саму архитектурную проблему, а лишь маскируют её проявления.

Целью данной работы является экспериментальная демонстрация существования «фон-неймановского узкого места» путём сравнительного количественного анализа производительности при оптимальном (последовательном) и наихудшем (случайном) сценариях доступа к памяти.

2. Материалы и методы

Для экспериментального подтверждения существования «фон-неймановского узкого места» была разработана методика, основанная на сравнительном анализе производительности при различных паттернах доступа к оперативной памяти. Основной задачей было измерить, как эффективность работы с памятью влияет на общее время выполнения вычислительной задачи.

Эксперимент проводился на ноутбуке со следующей аппаратной конфигурацией:

- Процессор: AMD Ryzen 7 5700U (1.80 GHz, 8 ядер / 16 потоков)
- Оперативная память: 16.0 ГБ DDR4, частота 3200 МГц
- Графический адаптер: AMD Radeon Graphics (выделено 496 МБ)
- Накопитель: SSD объёмом 477 ГБ
- Операционная система: Windows 10

Методика и инструменты эксперимента:

В качестве инструмента исследования был выбран язык программирования C#, как обеспечивающий высокую производительность и точный контроль над выполнением программ. Для высокоточного замера времени использовался класс System.Diagnostics.Stopwatch.

3. Результаты и обсуждение

Были разработаны две тестовые программы, работающие с массивом размером в 10 миллионов целочисленных элементов.

3.1. Тест последовательного доступа

```
using System;
using System.Diagnostics;

class Program
{
   static void Main()
   {
      int size = 10_000_000;
      int[] data = new int[size];
      for (int i = 0; i < size; i++)
      {
         data[i] = i;
      }
      var stopwatch = Stopwatch.StartNew();
      long sum = 0;
      for (int i = 0; i < size; i++)
      {
         sum += data[i];
      }
      stopwatch.Stop();
      Console.WriteLine($"Последовательный доступ: {stopwatch.Elapsed.TotalSeconds:F4} секунд");
   }
}
```

Программа выполняет итерацию по массиву в порядке возрастания индексов, что соответствует оптимальному для кэш-памяти паттерну.

3.2. Тест случайного доступа

```
using System;
using System.Diagnostics;
using System.Linq;

class Program
{
   static void Main()
   {
      int size = 10_000_000;
      int[] data = new int[size];
      int[] indices = new int[size];
      for (int i = 0; i < size; i++)
      {
         data[i] = i;
         indices[i] = i;
      }
      Random rng = new Random();
      indices = indices.OrderBy(x => rng.Next()).ToArray();
      var stopwatch = Stopwatch.StartNew();
      long sum = 0;
      for (int i = 0; i < size; i++)
      {
         sum += data[indices[i]];
      }
```

```
        stopwatch.Stop();
        Console.WriteLine($"Случайный доступ: {stopwatch.Elapsed.TotalSeconds:F4} секунд");
    }
}
```

Программа обращается к элементам массива в заведомо перемешанном порядке. Такой паттерн делает невозможной эффективную предвыборку данных в кэш и приводит к постоянным кэш-промахам.

Оба алгоритма выполняют идентичный объем работы — суммирование всех элементов массива, что обеспечивает корректность сравнительного анализа.

3.3. Анализ результатов эксперимента

Результаты выполнения тестовых программ представлены ниже:

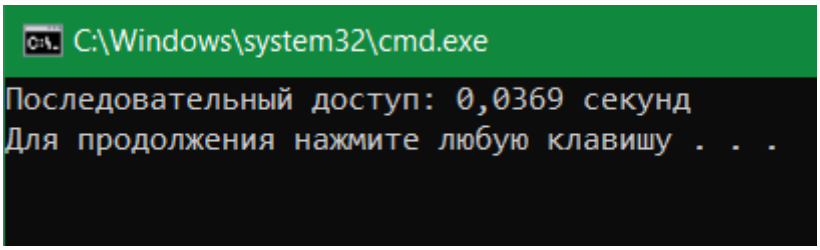

Рисунок 1. Время выполнения при последовательном доступе.

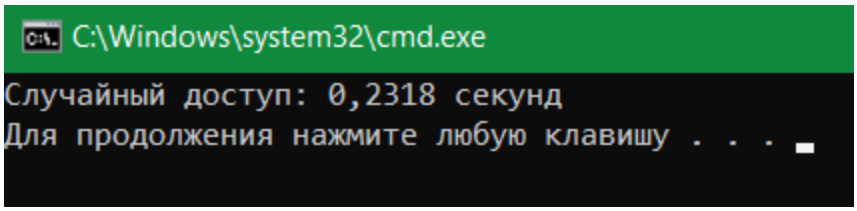

Рисунок 2. Время выполнения при случайном доступе.

Как видно из результатов, случайный доступ к памяти выполняется примерно в 6 раз медленнее последовательного. Эта разница возникает потому, что при последовательном доступе механизмы кэширования и предвыборки данных работают эффективно, предугадывая следующие запросы процессора. При случайном доступе эти механизмы не справляются, и процессор вынужден постоянно обращаться к медленной оперативной памяти, что и является прямым доказательством существования «фон-неймановского узкого места».

3.4. Актуальность архитектуры фон Неймана и современные пути преодоления «узкого места»

Несмотря на экспериментально подтвержденное фундаментальное ограничение, архитектура фон Неймана сохраняет свою актуальность благодаря универсальности и масштабной программной экосистеме [1,2]. Однако современные вычислительные системы используют комплексный подход для минимизации влияния «узкого места»:

- Совершенствование иерархии памяти. Результаты эксперимента демонстрируют критическую важность кэш-памяти. Развитие идет по пути увеличения объемов и уровней кэша (L1-L4) и применения интеллектуальных алгоритмов предвыборки данных [5].
- Распараллеливание вычислений. Многоядерные архитектуры позволяют распределять нагрузку между ядрами: пока одно ядро ожидает данные из памяти, другие выполняют полезную работу.
- Гибридные архитектуры. Универсальные CPU все чаще работают в связке со специализированными акселераторами (GPU, NPU), которые используют гарвардскую архитектуру с раздельными каналами для команд и данных.
- Вычисления в памяти. Перспективное направление, предполагающее размещение вычислительных элементов непосредственно в чипе памяти, что исключает необходимость передачи данных.

4. Заключение

Таким образом, современные подходы к развитию архитектуры направлены не на замену фон-неймановской модели, а на ее эффективное дополнение и усовершенствование. Это обеспечивает сохранение данной архитектуры в качестве фундаментальной основы универсальных вычислений, поскольку:

1. Фон-неймановская архитектура доказала свою исключительную универсальность и способность адаптироваться к меняющимся вычислительным парадигмам.
2. Накопленная за десятилетия экосистема программного обеспечения и инструментов разработки создает мощный стимул для сохранения совместимости.

3. Гибридные решения позволяют сочетать проверенные временем принципы фон Неймана с современными специализированными подходами, максимально используя преимущества каждой архитектуры.

Эволюционный путь развития, при котором фон-неймановская модель не заменяется, а обогащается новыми решениями, оказывается наиболее практичным и эффективным для обеспечения прогресса в области вычислительных технологий.

Список литературы

1. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – СПб.: Питер, 2022. – 826 с.
2. Hennessy J. Computer Architecture: A Quantitative Approach / J. Hennessy, D. Patterson. – 6th ed. – Morgan Kaufmann, 2019. – 936 p.
3. Wulf W. Hitting the Memory Wall: Implications of the Obvious / W. Wulf, S. McKee [Электронный ресурс] // ACM SIGARCH Computer Architecture News. – 1995. – URL: https://dl.acm.org/doi/10.1145/216585.216588 (дата обращения: 30.09.2025).
4. Backus J. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs // Communications of the ACM. – 1978. – Vol. 21. – No. 8. – P. 613-641.
5. Smith A.J. Cache Memories // ACM Computing Surveys. – 1982. – Vol. 14. – No. 3. – P. 473-530.