Инженерный вестник Дона, №4 (2025)
ivdon.ru/ru/magazine/archive/n4y2025/9996

Адаптивная архитектура конвейера с разделяемой памятью и выборочной упорядоченностью для высокопроизводительной потоковой обработки данных

М.О. Антонов, И.О. Темкин, С.А. Дерябин

Национальный исследовательский технологический университет «МИСиС»

Аннотация: В данной работе представлена адаптивная архитектура вычислительного конвейера, позволяющая повысить пропускную способность и снизить задержку при обработке потоковых данных в реальном времени как в одно-, так и в многопроцессорных системах. В отличие от преимущественно концептуальных моделей или узко ориентированных алгоритмов, практический эффект проявляется в достижении измеримого роста производительности за счёт уменьшения числа повторных копирований данных и синхронизационных издержек или гибкой настройки порядка входных и выходных данных. Архитектура использует разделяемую память для исключения дублирования буферов, применяет каналы передачи данных в зависимости от потребности в упорядочивании, а также предусматривает репликацию процессов внутри или между ядрами центрального процессора. Эксперименты показали, что предложенная архитектура обеспечивает как высокую пропускную способность, так и малые задержки, вводя минимальные накладные расходы на пересылку данных и синхронизацию процессов. Тем самым архитектура служит гибкой и масштабируемой основой для широкого круга приложений реального времени — от систем видеонаблюдения и робототехники до распределённых платформ обработки больших массивов данных.
Ключевые слова: параллелизм, многопроцессорные вычисления, вычислительный конвейер, масштабирование производительности, очереди, разделяемая память.

Введение

В последние годы потоковая обработка данных становится неотъемлемой частью современных вычислительных систем от видеонаблюдения и промышленной автоматизации до высокопроизводительных научных расчётов. Неуклонный рост объёмов информации в социально-экономических и технологических сферах усиливает спрос на решения, способные одновременно обеспечивать высокую пропускную способность и низкую задержку при обработке входящих потоков в реальном времени [1]. Традиционные подходы к повышению производительности, такие как перенос вычислений на кластер или облако, нередко сопровождаются издержками: появляется необходимость многократной передачи больших объёмов данных и их сериализации, а также

строгих согласований между узлами распределённой системы, что затрудняет соблюдение жёстких временных ограничений реального времени.

Одним из ключевых способов повышения эффективности в потоковой обработке стали вычислительные конвейеры [2], которые по аналогии с производственными «конвейерами» разбивают задачу на ряд специализированных этапов. Каждый этап выполняется параллельно в собственном процессе или на выделенном ресурсе, что позволяет повысить общий уровень параллелизма за счёт одновременной обработки множества элементов потока. Тем не менее, у существующих конвейерных решений часто обнаруживаются недостатки, такие, как высокая стоимость копирования и сериализации данных, жёсткая зависимость этапов друг от друга, а также отсутствие гибких механизмов по сохранению или игнорированию порядка входных данных в зависимости от специфики задачи.

Например, Chun B., Ihm S., Maniatis P. и др. представили CloneCloud — систему, которая позволяет плавно перенаправлять часть выполнения задач с мобильных устройств на так называемые устройства-клоны [3] — виртуальные копии в вычислительном облаке. Поток информации или данных с мобильного устройства в отдельно взятый момент времени переносится на клон в облаке с последующим ожиданием результата. Предложенный подход, в целом, позволяет говорить о некоторой эффективности организации вычислений, однако он обладает рядом недостатков. Архитектура CloneCloud не позволяет осуществлять миграцию программного или аппаратного состояния работы приложения в виртуальную среду, а наличие виртуальных машин в облаке, в свою очередь, не позволяет в полной мере задействовать имеющиеся вычислительные ресурсы физических устройств за счет ограничений в потреблении и создания дополнительных уровней абстракций. Разнесенные отправка и получение результатов, а также отсутствие гарантии

доставки в отведенный для этого временной промежуток, ограничивают возможности достижения истинного параллелизма в задачах.

В качестве попытки устранения некоторых подобных проблем, Heo J., Bhardwaj K. и Gavrilovska A. предложили FleXR — гибко настраиваемую систему обработки распределенных потоков для систем дополненной реальности [4]. Основным компонентом системы являются ядра, объединение в конвейер которых осуществляется через так называемые порты — средства передачи входных и выходных данных между связанными ядрами. Дополнительно, к преимуществам подхода можно отнести возможность использования гетерогенных систем [5], применяя, например, графический процессор, аппаратные ускорители и др. Однако, в виду узкой специализации решения, направленного на применение в системах дополненной реальности, применение данной архитектуры для других задач не адаптировано и не может быть использовано напрямую.

Опираясь на результаты множества исследований, следует предположить, что потоковая обработка информации не только может быть организована с достаточной степенью эффективности [6, 7], но и обладать существенным преимуществом организации обработки данных в целом. Подобное преимущество может быть достигнуто посредством повышения производительности вычислительного конвейера при организации работы с каждым отдельно взятым ядром аппаратных вычислительных компонентов системы [8]. В то же время, необходимо учитывать, что эффективность использования архитектуры конвейеров для потоковой обработки данных сталкивается с препятствиями в виде коммуникационных затрат — расходов на перемещение данных между вычислительными узлами [9].

Построение архитектуры вычислительного конвейера

Конвейеры — это динамические одно- и многопроцессорные вычислительные последовательности, демонстрирующие следующие закономерности в своем составе:

- **задачи** или **результаты** в зависимости от того, рассматриваются ли они как входные или выходные данные;
- вычислительный **этап**, олицетворяющий **процесс** и выполняющий преобразование или вычисление на заданном наборе данных;
- этапы, образующие **поток данных**, под которым стоит понимать набор связей, формирующих однонаправленные деревья;
- набор связей, олицетворяющий собой интерфейс передачи данных, поддерживающий методы записи и чтения, образуя **поток управления** и олицетворяя процесс трансформации данных в конечный ожидаемый результат.

Время обработки данных с помощью такой программы зависит от числа этапов. **Сбалансированный** или **синхронный** конвейер состоит из этапов, имеющих одинаковое время обработки одного элемента входных данных и одинаковое время записи результата выполнения операции над ним.

Вычислительный конвейер состоит из конечной последовательности (1) каналов передачи данных c_i, где $0 \leq i \leq p+1$, а также вычислительных этапов p_i в соответствующем диапазоне $0 \leq i \leq p$. Здесь p — некоторое произвольное и олицетворяющее количество вычислительных этапов положительное целое число. Интерфейс связи c_0 называется входным интерфейсом конвейера, а c_{p+1} — выходным.

$$c_0 \xrightarrow{p_0} c_1 \xrightarrow{p_1} c_2 \rightarrow \cdots c_p \xrightarrow{p_i} c_{p+1} \quad (1)$$

Вычислительный конвейер называется **асинхронным** или **несбалансированным**, если управление синхронизацией работы вычислительных этапов реализовано при помощи самого потока данных. Иными словами, каждый вычислительный этап выполняет следующий набор операций:

- ожидание готовности данных для чтения из интерфейса c_i;
- прием единицы обрабатываемой информации из интерфейса c_i;
- выполнение операции над считанной информацией;
- запись результата вычислительной операции в выходной интерфейс c_{i+1}.

Таким образом, если на вход вычислительного конвейера поступает n единиц данных, то для каждого из p этапов число таких операций составит n.

Обозначив через $r(c_i)$ время чтения из входного интерфейса, приняв $\tau(p_i)$ как длительность выполнения вычислительной операции, выполняемой этапом i, а $w(c_i)$ — время записи результатов в выходной интерфейс, получим суммарное время обработки (2) одной единицы данных с помощью этапа i в диапазоне $0 \leq i \leq p$, иными словами — задержку этапа i.

$$\tau_i = r(c_i) + \tau(p_i) + w(c_{i+1}) \quad (2)$$

Обозначив через μ задержки каждого из этапов, получим формулу (3) для вычисления времени обработки n элементов данных p-мерным синхронным вычислительным конвейером.

$$\tau(n) = (p + n - 1)\mu \quad (3)$$

В случае использования **несбалансированного** или **асинхронного** вычислительного конвейера, выработанные ранее обозначения теряют свою силу из-за неравномерности распределения времени выполнения между этапами вследствие синхронизации работы самим потоком данных. Таким образом, полное время обработки единицы информации составит сумму всех задержек работы вычислительного этапа i без учета параллелизма (4).

$$\sigma = \sum_{i=0}^{p} (r(c_i) + \tau(p_i) + w(c_{i+1})) \quad (4)$$

Обозначив через μ, вычислим максимальное время задержки в случае применения **асинхронного** подхода (5).

$$\mu = \max_{0 \le i \le p} (r(c_i) + \tau(p_i) + w(c_{i+1})) \quad (5)$$

Используя значение полного времени обработки единицы информации, а также максимальное время задержки этапа, получим минимальное время обработки n входных элементов с помощью **асинхронного** вычислительного конвейера (6).

$$\tau(n) = \sigma + (n - 1)\mu \quad (6)$$

Работа далеко не каждой вычислительной системы может являться эффективной несмотря на следование принципам потоковой обработки, в

результате чего преимущества высокопроизводительного параллелизма становятся его недостатками вследствие возникновения **структурных конфликтов** или же **конфликтов потоков управления** и **данных**.

Под **структурными конфликтами** подразумеваются условия, в рамках которых обработка потоковой информации осуществляется неравномерно, а сам вычислительный конвейер считается **несбалансированным**. В случае возникновения структурных конфликтов продолжительность работы системы ограничена временем вычисления в рамках самой медленной стадии, что в свою очередь приводит к накоплению очередей из необработанных данных.

Конфликты потока управления могут возникать при изменении порядка чтения или записи данных отдельно взятым этапом последовательности, что подразумевает необходимость соблюдения потоковой безопасности. Данный тип наиболее распространен при формировании параллельных ветвей, каждая из которых функционирует независимо от другой, обрабатывая идентичный набор входных данных.

В случае **конфликта потоков данных** наблюдается низкопроизводительная передача информации между вычислительными этапами за счет как необходимости многократного копирования при использовании независимых параллельных ветвей, так и вносящей дополнительный вклад во время выполнения сериализации.

Оценка эффективности или погрешность подразумевают определение разности эталонного и фактического времени работы при увеличении количества элементов конвейера и связей между ними по отношению к эталонному значению времени, если бы одноименная задача выполнялась вне структуры вычислительной системы (7).

$$ERR = \frac{REAL - CALC}{CALC} 100\% \quad (7)$$

ERR — погрешность расчета затраченного времени [%];

$REAL$ — фактически затраченное время [с];

$CALC$ — эталонное время выполнения [с].

С целью повышения производительности, а также во избежание появления **структурных конфликтов**, в рамках настоящей работы применена техника реплицирования процессов внутри этапа, где каждый из процессов является полной и атомарной копией друг друга.

Упорядоченный этап поддерживает последовательность выполнения среди своих процессов, причем каждый из них синхронизирован как с предыдущим, так и с последующим его «соседом», таким образом образуя «круг». Данный подход гарантирует, что входные данные будут обработаны в соответствии с той последовательностью, с которой те были размещены внутри конвейера. С целью достижения описываемого поведения, в состав этапа введены дополнительные примитивы синхронизации, обеспечивающие поддержание последовательности выполнения.

В рамках **неупорядоченного этапа** симметрии взаимодействия между рабочими процессами не наблюдается и каждый процесс начинает обработку следующей ближайшей доступной задачи сразу, как только выполнение текущей будет завершено, делая доступным результат в виде выходных данных своего этапа.

Принимая во внимание необходимость разрешения **конфликта потока данных**, в качестве набора передаваемых данных предлагается использование строго битовой последовательность во избежание сериализации. Учитывая необходимость в чтении или записи битовых последовательностей целиком, а также в соблюдении потоковой безопасности во избежание повреждения данных, стоит принимать во внимание ограничения на работу с интерфейсами

передачи данных и потребность во введении и применении примитивов синхронизации. В качестве интерфейсов передачи данных используются однонаправленные каналы и очереди. И однонаправленные каналы, и очереди идентично используются для отправки и получения данных между процессами, но в отличие от очередей каналы являются механизмами более низкого уровня, требующими явного создания соединений между двумя процессами, а затем явной отправки и получения данных между ними. Очереди, одновременно с этим, могут быть рассмотрены как локальная структура данных, которая может быть случайно использованной между множеством процессов за счет наличия встроенных в нее примитивов синхронизации. Соответственно, наличие таких примитивов в составе делает очереди менее производительными по отношению к однонаправленным каналам, замедляя скорость потока данных, как показали наши внутренние тесты производительности, практически в два раза. Учитывая обнаруженное, было принято решение использовать одновременно как слабые, так и сильные стороны каждого из интерфейсов передачи данных, а именно, за счет наличия потоковой безопасности использовать очереди для неупорядоченных стадий и однонаправленные каналы для упорядоченных соответственно.

Во избежание повреждения данных при использовании однонаправленных каналов, а также необходимости поддерживать очередность чтения и записи для корректной работы упорядоченного этапа, в качестве примитивов синхронизации были введены семафоры, разрешающие работу на чтение или запись каждому из процессов внутри этапа. Таким образом, процесс, вычитав данные до конца, разблокирует идентичную операцию чтения последующему, который в данный момент ожидает разрешения и, получив его, автоматически блокирует себе доступ к интерфейсу до момента повторного разрешения от предшественника.

Напоследок, предлагается отказаться от многократного копирования между промежуточными буферами для значительного увеличения производительности системы. Отсутствие многократного копирования достигается за счет применения разделяемой памяти, что является наиболее быстрым средством обмена данными между процессами [10].

Результаты тестирования

Тестируемый вычислительный конвейер представляет собой **неупорядоченный** этап, в рамках которого запускается от 2 до 32 процессов, а сама задача имитирует вычислительную операцию на центральном процессоре путем тактирования в заданном временном интервале пустых NOP-операций — однобайтовых инструкций во время исполнения которых ничего не происходит, за исключением того, что они используют один тактовый цикл центрального процессора во время которого счетчик команд переходит к следующей.

Перед началом каждой итерации тестирования осуществляется вычисление количества NOP-операций за 1 секунду, после чего полученное в качестве эталонного значение передается на вход основной программе для формирования вычислительного конвейера. Учитывая, что нам известно время работы, количество тактов и алгоритм, на базе которого произведен подсчет, то, зная количество обрабатываемых задач, а также перенеся сам алгоритм в тело вычислительного этапа, можно извлечь как эталонное время работы конвейера, так и реальное, сформировав олицетворяющий количество накладных расходов на потоки данных и управления, процент отклонения. Например, при длительности работы алгоритма в 1 секунду, 64 задачах на входе и 32 процессах в рамках неупорядоченного этапа, суммарное эталонное время работы всего конвейера составит 2 секунды.

В качестве оборудования для тестирования производительности использовался сервер на базе двух Intel Xeon E5-2699 v4 под управлением ОС

Debian GNU/Linux 11 с версией ядра 5.10.0-28. Как следует из представленных в таблице 1 результатов, погрешность составляет не более 2.7 % от эталонного расчетного времени для однопроцессорных систем. В частности, вместо прохождения всего вычислительного конвейера за 10.67 секунды при запущенных 6 процессах внутри одиночного этапа, цепочка вычислений завершает свою работу за 10.96 секунды.

Таблица № 1

Оценка производительности предлагаемого решения

Процессы, ед.	Время факт., с	Время эталон., с	Погрешность, %
2	30,84	32,00	-3,6
4	16,33	16,00	2,1
6	10,96	10,67	2,7
8	8,14	8,00	1,8
10	6,47	6,40	1,1
12	5,23	5,33	-1,9
14	4,65	4,57	1,7
16	3,95	4,00	-1,4
18	3,51	3,56	-1,1
20	3,36	3,20	5,1
22	3,12	2,91	7,2
24	2,96	2,67	11,1
26	2,75	2,46	11,8
28	2,57	2,29	12,6
30	2,48	2,13	16,2
32	2,37	2,00	18,3

Данная погрешность обусловлена набегающей ошибкой вычисления тактирования NOP-операций из-за разницы профиля нагрузки операционной

системы и наличием дополнительных соединений для осуществления межпроцессного взаимодействия. Отрицательная же погрешность при запущенных двух процессах обусловлена особенностью работы центральных процессоров, у которых есть два быстрых ядра с поддержкой максимальных частот. Средняя же погрешность составляет 1.88 %, а приведенные результаты истинны при условии, что задействованное для вычислений оборудование поддерживает только один центральный процессор. Рассматривая же использование вычислительного конвейера на 22 и более ядрах, что является самым простым и экономически выгодным способом масштабирования системы [11], можно наблюдать, что увеличение количества процессов не приводит к наблюдаемому раньше паритету производительности и времени выполнения, а погрешность только растет.

Литература

1. Роби Р. Параллельные и высокопроизводительные вычисления / пер. с англ. А. В. Логунова. М.: ДМК Пресс, 2022. 800 с.

2. Хорошевский В.Г., Курносов М.Г., Мамойленко С.Н., Павский К.В., Ефимов А.В., Пазников А.А., Перышкова Е.Н. Масштабируемый инструментарий параллельного мультипрограммирования пространственно распределённых вычислительных систем // Вестник СибГУТИ. 2011. № 4. С. 3-18.

3. Chun B.-G., Ihm S., Maniatis P., Naik M. CloneCloud: boosting mobile device applications through cloud clone execution. Proc. of the 6th conference on Computer systems (EuroSys'11), 2011, pp. 301-314. doi: 10.1145/1966445.1966473.

4. Heo J., Bhardwaj K., Gavrilovska A. FleXR: A system enabling flexibly distributed extended reality. Proc. of the 14th ACM Multimedia Systems Conf., 2013, pp. 1-13. doi: 10.1145/3587819.3590966.

5. Gschwind M. The cell broadband engine: exploiting multiple levels of parallelism in a chip multiprocessor. Inter. Journal of Parallel Programming, 2007, vol. 35, pp. 233–262. doi: 10.1007/s10766-007-0035-4.

6. Zheng Z., Oh C., Zhai J., Shen X., Yi Y., Chen W. VersaPipe: A Versatile Programming Framework for Pipelined Computing on GPU. Inter. Proc. of the 50th Annual IEEE MICRO-50, Cambridge, MA, USA, 2017, pp. 587-599. doi: 10.1145/3123939.3123978.

7. Cucinotta T., Subramanian V. Characterization and analysis of pipelined applications on the Intel SCC. Conf.: Proc. of the 4th Many-core Applications Research Community (MARC) Symposium, 2012, (55), pp. 37-42.

8. Padmanabhan S., Chen Y., Chamberlain R. D. Optimal design-space exploration of streaming applications. Proc. IEEE Inter. Conf. on Application-specific Systems, Architectures and Processors (ASAP), 2011, pp. 227-230. doi: 10.1109/ASAP.2011.6043274.

9. Meyerovich L., Rabkin A. Empirical analysis of programming language adoption. Inter. Proc. of the 2013 ACM SIGPLAN - conference on Object oriented programming systems languages & applications, 2013, pp. 1-18. doi: 10.1145/2509136.2509515.

10. Колисниченко Д.Н. Разработка Linux-приложений. СПб.: БХВ-Петербург. 2012. 430 с.

11. Stallings W. Operating Systems: Internals and Design Principles. 7th Edition, Prentice Hall, 2011, 816 p.

References

1. Robi R., Zamora Dzh. Parallel and High Performance Computing"; [Eng. Transl. A. V. Logunova]. Moskva, 2022. 800 p.

2. Horoshevskij V.G., Kurnosov M.G., Mamojlenko S.N., Pavskij K.V., Efimov A.V., Paznikov A.A., Peryshkova E.N. Vestnik SibGUTI. 2011. №4. pp. 3-18.

3. Chun B.-G., Ihm S., Maniatis P., Naik M. CloneCloud: boosting mobile device applications through cloud clone execution. Proc. of the 6th conference on Computer systems (EuroSys'11), 2011, pp. 301-314. doi: 10.1145/1966445.1966473.

4. Heo J., Bhardwaj K., Gavrilovska A. FleXR: A system enabling flexibly distributed extended reality. Proc. of the 14th ACM Multimedia Systems Conf., 2013, pp. 1-13. doi: 10.1145/3587819.3590966.

5. Gschwind M. Inter. Journal of Parallel Programming, 2007, vol. 35, pp. 233–262. doi: 10.1007/s10766-007-0035-4.

6. Zheng Z., Oh C., Zhai J., Shen X., Yi Y., Chen W. VersaPipe: A Versatile Programming Framework for Pipelined Computing on GPU. Inter. Proc. of the 50th Annual IEEE MICRO-50, Cambridge, MA, USA, 2017, pp. 587-599. doi: 10.1145/3123939.3123978.

7. Cucinotta T., Subramanian V. Characterization and analysis of pipelined applications on the Intel SCC. Conf.: Proc. of the 4th Many-core Applications Research Community (MARC) Symposium, 2012, (55), pp. 37-42.

8. Padmanabhan S., Chen Y., Chamberlain R. D. Proc. IEEE Inter. Conf. on Application-specific Systems, Architectures and Processors (ASAP), 2011, pp. 227-230. doi: 10.1109/ASAP.2011.6043274.

9. Meyerovich L., Rabkin A. Empirical analysis of programming language adoption. Inter. Proc. of the 2013 ACM SIGPLAN - conference on Object oriented programming systems languages & applications, 2013, pp. 1-18. doi: 10.1145/2509136.2509515.

10. Kolisnichenko D.N. Razrabotka Linux-prilozhenij [Linux application development]. Sankt-Peterburg, BHV-Peterburg, 2012. 430 p.

11. Stallings W. Operating Systems: Internals and Design Principles. 7th Edition, Prentice Hall, 2011, 816 p.

Дата поступления: 5.02.2025

Дата публикации: 26.03.2025