

ИНФОРМАЦИОННЫЕ
ТЕХНОЛОГИИ
И ТЕЛЕКОММУНИКАЦИИ

INFORMATION
TECHNOLOGIES
AND TELECOMMUNICATIONS

УДК 004.43

DOI: 10.17212/2782-2001-2025-4-69-84

Сравнение производительности для различных способов реализации мультиметодов*

П.В. КОСОВ^а, А.И. ЛЕГАЛОВ^б

РФ, 101000, г. Москва, ул. Мясницкая, 20, Национальный исследовательский университет «Высшая школа экономики»

^а pvkosov@hse.ru ^б alegalov@hse.ru

Мультиметоды, определяемые как функции, полиморфные к нескольким аргументам, регулярно встречаются при разработке программного обеспечения. Они непосредственно реализованы в ряде языков программирования, для которых компиляторы автоматически порождают соответствующий машинный код, используя при этом различные подходы. Отсутствие реализации мультиметодов компенсируется либо непосредственным написанием кода, либо их включением в состав библиотек языков программирования. Разнообразие алгоритмов реализации мультиметодов часто связано с необходимостью достижения требуемых критериев качества, одним из которых является эволюционная расширяемость без изменения ранее написанного кода. Скрывая внутренние решения за высокоуровневыми конструкциями языков программирования, компиляторы формируют окончательное связывание альтернативных фрагментов мультиметодов, расположенных в независимых единицах компиляции, используя различные приемы, что обеспечивает необходимую целостность на выходе. При этом подходы к реализации множественного полиморфизма обладают разной производительностью и сложностью поддержки в добавлении новых комбинаций аргументов при расширении функциональности. Одним из подходов, обеспечивающих гибкую реализацию мультиметодов, является процедурно-параметрическая парадигма программирования. Она предлагает эффективное решение, основанное на параметризации функциональных зависимостей через многомерные массивы. Это обеспечивает эволюционное расширение функциональности и данных без изменения ранее написанного кода, поддерживая при этом безопасность за счет статической типизации. В работе проводится сравнение процедурно-параметрической реализации мультиметодов с существующими подходами. Для различных алгоритмов сформированы временные оценки поиска комбинаций параметров, необходимых для вызова мультиметода. Эксперименты демонстрируют эффективность процедурно-параметрического полиморфизма по сравнению с другими подходами. На основе результатов проведенного исследования даны рекомендации по реализации мультиметодов в зависимости от требований к производительности.

Ключевые слова: языки программирования, компиляция, процедурно-параметрическое программирование, полиморфизм, множественный полиморфизм, мультиметоды, эволюционная разработка программного обеспечения, парадигмы программирования, методы разработки программного обеспечения

* Статья получена 01 июля 2025 г.

ВВЕДЕНИЕ

При разработке программного обеспечения критически важным становится обеспечение гибкости и постоянного расширения функциональности системы во время эксплуатации. Одним из способов является эволюционное расширение программы. Это актуально также и на ранних этапах разработки, когда требования к системе могут меняться, а необходимость ускоренного вывода продукта на рынок требует минимальных изменений в уже написанном коде. Общеизвестно, что внесение изменений в работающие компоненты часто приводит к ошибкам, увеличивая стоимость сопровождения и снижая надежность.

Эффективным средством, помогающим решать эту задачу, выступает динамический полиморфизм. Он предоставляет возможность обработки альтернативных ситуаций без явной проверки типов во время выполнения. Во многих ситуациях это позволяет наращивать функциональность программы без изменения ранее написанного кода. Исторически его реализация тесно связана с объектно ориентированным программированием (ООП). В языках с динамической типизацией, таких как Smalltalk [1] и Python [2], применяется «утиная типизация», которая определяет совместимость объектов по наличию методов с одинаковыми сигнатурами. В статически типизированных объектно ориентированных (ОО) языках (например, в C++ [3], Java [4], C# [5]) полиморфизм реализуется через наследование и виртуализацию. Наследование гарантирует идентичность интерфейсов, а виртуализация – подмену или расширение функциональности в производных классах.

Эволюция парадигм программирования привела к использованию динамического полиморфизма в процедурных и функциональных языках программирования. Например, в Go [6] интерфейсы связывают функции со структурами данных, а в Rust [7] аналогичная функциональность реализована через типаж. В функциональных языках, таких как Haskell [8], указатели на специализированные методы передаются в виде таблиц (словарей) неявно, обеспечивая гибкость при сохранении статической типизации [9].

Вместе с тем перечисленные подходы ориентированы на поддержку только одиночного полиморфизма (монометоды), при котором выбор альтернативной функциональности определяется только одним параметром. Это создает трудности при использовании мультиметодов – функций, полиморфных от нескольких аргументов. Для программной реализации множественного полиморфизма используют различные подходы. В ООП можно использовать диспетчеризацию [10], которая, в частности, применена в ОО-паттерне проектирования «Посетитель» [11]. Часто применяются комбинированные решения, сочетающие ОО и процедурный подходы [12–14]. Аналогичные приемы используются в компиляторах языков программирования, поддерживающих мультиметоды, а также в библиотеках функций и классов, ориентированных на их использование. Однако существующие реализации зачастую имеют невысокую производительность или не обеспечивают гибкое расширение программ без изменения ранее написанного кода.

К языкам, поддерживающим мультиметоды, относится CLOS [15]. Однако их реализация, несмотря на возможность эволюционного добавления альтернативных обработчиков, является громоздкой, что ограничивает применимость подхода в статически типизированных языках программирования. Например, анализируя подобные решения, Б. Страуструп отказался от вклю-

чения мультиметодов в C++ [16]. Для языка программирования Rust разработана библиотека мультиметодов (multimethods.rs), использующая хеш-таблицы. Однако, согласно исследованиям авторов библиотеки, она обладает низкой производительностью [17].

Решение многих указанных проблем возможно при использовании процедурно-параметрической (ПП) парадигмы программирования [18]. Предлагаемый в ней механизм динамического полиморфизма основан на формировании параметрических отношений между альтернативами, определяемыми в качестве подтипов обобщающего типа. Альтернативные специализации имеют локальную идентификацию внутри обобщения, что позволяет создавать функции, полиморфные от нескольких аргументов за счет их косвенного связывания в единую обобщающую функцию через многомерные массивы указателей. Для обработки различных комбинаций альтернатив осуществляется доступ с использованием индексов этой таблицы, что обеспечивает высокую скорость вызова нужного обработчика. Необходимо, используя индексы специализаций, вычислить только местоположения указателя на нужную функцию. При этом имеется возможность эволюционного расширения программы при добавлении новых альтернатив.

Целью работы является сравнительный анализ производительности процедурно-параметрического подхода по сравнению с традиционными решениями при выполнении мультиметодов. Для достижения цели решаются следующие задачи:

- анализ существующих подходов к реализации мультиметодов и их ограничений;
- описание методики экспериментального сравнения, используемой метрики и механизма проведения сравнения производительности диспетчеризации;
- анализ результатов экспериментального сравнения ПП подхода с альтернативными методами.

1. ПОДХОДЫ К РЕАЛИЗАЦИИ МУЛЬТИМЕТОДОВ

Множественный полиморфизм используется при решении различных задач, что объясняет стремление к его непосредственной поддержке на уровне языков программирования. Однако реализация мультиметодов на уровне трансформации в машинный код сталкивается с рядом вызовов, связанных как с поддержкой производительности вычислений, так и с обеспечением последующего гибкого расширения программы. Традиционные подходы, такие как деревья решений, хеш-таблицы или динамическая диспетчеризация на основе признаков типов, часто требуют компромиссов между скоростью поиска специализаций и гибкостью поддержки добавляемых типов данных или классов, что ограничивает их применимость в сложных высоконагруженных системах.

Одной из задач, требующих нетривиального решения, является обеспечение симметрии в скорости выбора специализаций при асимметричных алгоритмах поиска комбинаций аргументов в мультиметодах. Различие между симметричными (symmetric) и асимметричными (asymmetric) алгоритмами заключается в последовательности обхода аргументов.

В симметричных алгоритмах все аргументы, участвующие в выборе реализации, обрабатываются равноправно. Система ищет единственный наиболее специфичный метод, в сигнатуре которого типы параметров являются подтипами фактических типов всех аргументов одновременно, не выделяя среди них доминирующий. Это обеспечивает максимальную гибкость для определения поведения, зависящего от произвольных комбинаций типов, но требует сложных механизмов для гарантии однозначности.

В асимметричных алгоритмах явно или неявно выделяется доминирующий аргумент (часто первый) и диспетчеризация осуществляется последовательно. В начале выбирается группа методов на основе типа доминирующего аргумента, а затем среди них отбирается наиболее подходящая реализация на основе типов остальных (второстепенных) аргументов. Хотя это ограничивает выразительность, так как привязывает поведение к иерархии доминирующего типа. Асимметричный подход значительно упрощает интеграцию в традиционные ОО-системы. Он, используя ОО-полиморфизм, является более производительным, чем симметричный подход, облегчая проверку однозначности и полноты покрытия в рамках выбранного доминирующего типа.

В работе [19] отмечено, что чистое симметричное решение для мультиметодов усложняет систему и снижает производительность, поэтому предлагаются гибридные подходы, сочетающие асимметричные стратегии с обратными проходами для уточнения выбора. В то же время в функциональных языках, таких как Kea [20] и NOAL [21], реализация мультиметодов сопровождается строгими требованиями к симметрии и полноте, что ограничивает их адаптивность в динамических сценариях.

Используемая в CLOS симметричная диспетчеризация [22] не обеспечивает желаемую поддержку производительности и однозначности (особенно при множественном наследовании), что привело к доминированию асимметричных моделей с неявным главенством первого аргумента. Примером языка программирования с асимметричной диспетчеризацией можно назвать Cecil [23].

Такой подход, как диспетчеризация по кортежам [24], предлагает пути синтаксической интеграции мультиметодов в существующие ОО-языки, но не решает ограничений множественного полиморфизма в иерархических системах типов.

Поиск эффективных подходов к реализации мультиметодов симметричным способом без применения параметрических таблиц оказался неэффективным по производительности, а использование таких таблиц ведет к повышенному потреблению памяти. Поэтому для ОО-подхода были предложены решения, связанные со сжатием таблиц [25]. При наличии в вызове мультиметода множества одинаковых специализаций проблема также решалась раскрашиванием таблицы [26]. Однако этот подход невозможен в случае, когда число совпадающих специализаций относительно мало, что является, согласно [10], самым типичным случаем применения мультиметодов (плоская гетерогенная иерархия). Также в работе [25] указано, что параметрические таблицы в ООП-языках без сжатия неприменимы на практике из-за слишком глубоких иерархий наследования и из-за множественного наследования (в тех языках, которые это поддерживают).

Большая иерархия наследования в целом ведет к снижению производительности поиска обработчика специализаций. Для увеличения производительности также предлагается использование различных оптимизаций [27]: оптимизации всей программы (whole-program optimizations), оптимизации с использованием профиля выполнения программы (profile-guided optimization), сопоставления образцов (pattern matching) и деревьев решений (decision tree). В работе [27] дерево решений выбрано как наиболее подходящее для выбора специализации мультиметода по сравнению с остальными вариантами (например, хеш-таблицами). На примере языка Smalltalk [28] предложено решение улучшения производительности поиска специализации за счет выполнения оптимизации всей программы и поддержки со стороны системы сборки программы.

2. РЕАЛИЗАЦИЯ МНОЖЕСТВЕННОГО ПОЛИМОРФИЗМА В ПРОЦЕДУРНО-ПАРАМЕТРИЧЕСКОЙ ПАРАДИГМЕ

Для пояснения механизма, поддерживающего ПП-полиморфизм, сопоставим его с поддержкой ОО-полиморфизма. Схематично эти подходы представлены на рис. 1.

Объектно ориентированный полиморфизм формируется сочетанием наследования и виртуализации. Виртуализация при этом обычно реализуется за счет таблиц виртуальных методов (VT, сокращение от англ. *virtual table*). Таблица базового класса Base, выступающего в роли обобщения, содержит указатели на один или несколько методов (F), которые обычно переопределяются в виртуальных таблицах производных классов, являющихся специализациями обобщения ($Child_1 - Child_n$). При этом каждый производный класс, расширяя базовый класс, имеет собственный тип, что выражается в уникальном имени класса ($Child_i$). Производные классы формируются независимо друг от друга. Это позволяет эволюционно расширять альтернативные программные объекты, каждый из которых при наличии одинаковых интерфейсов может иметь иную функциональность, обеспечивая тем самым реализацию динамического ОО-полиморфизма. Однако добавление нового виртуального метода требует модификации всей иерархии классов. Также этот механизм напрямую не поддерживает мультиметодов, для реализации которых ОО-решением является использование диспетчеризации, не способствующей безболезненному расширению классов.

При ПП-подходе реализация отношений между обобщением (Generalization) и его специализациями осуществляется с использованием иного механизма [18]. Формирование новых специализаций ($specialization_1 - specialization_n$) обычно происходит на основе существующих типов, определяемых как основы специализаций ($foundation_1 - foundation_n$). Добавление альтернативных специализаций к обобщающему типу данных может осуществляться в произвольные моменты времени и в различных единицах компиляции. Формируемые при этом специализации обобщения принадлежат к тому же типу, что и обобщающий тип (Generalization), являясь его подтипами. Само обобщение также трактуется как одна из специализаций ($specialization_0$), которая содержит свои внутренние данные.

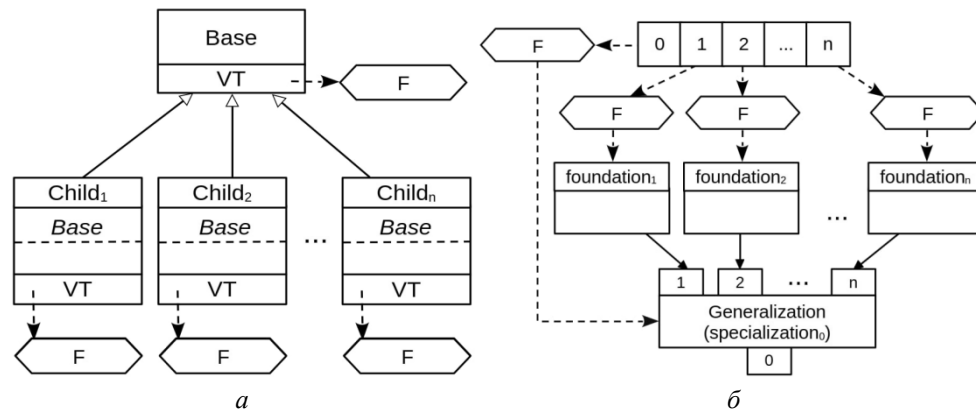


Рис. 1. Механизмы динамического полиморфизма:

a – объектно ориентированный подход; *б* – процедурно-параметрический подход

Fig. 1. Mechanisms of dynamic polymorphism:

a is object oriented approach; *b* is procedural-parametric approach

Обработка сформированных альтернатив осуществляется с использованием внешних независимых специальных функций – обработчиков специализаций, расширяющих обобщающую их функцию. Они также могут добавляться независимо друг от друга. Механизм идентификации специализаций локализован внутри каждого обобщения с использованием внутренней индексации (параметрических индексов). Этот же механизм используется для автоматического выбора обработчиков специализаций через параметрические таблицы, каждая из которых связана со своим набором обобщающих функций. Добавление новых обобщающих функций и их обработчиков специализаций может осуществляться в любой момент времени независимо от уже существующих данных и функций, что характерно для добавления любых новых функций в процедурном подходе. Эти функции могут содержать произвольное число обобщающих аргументов, обеспечивая прямую реализацию эволюционно расширяемых мультиметодов. Разработанный подход был использован при реализации языков программирования O2M [29] и PPC [30].

3. СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Для оценки эффективности механизма поддержки мультиметодов в процедурно-параметрической парадигме программирования необходимо сравнить его с существующими подходами, используемыми для реализации мультиметодов как на уровне отдельных языков, так и с применением программных решений. Это позволяет определить место ПП-подхода среди альтернативных методов. В ходе сравнения рассматриваются следующие вопросы:

- выбор вариантов для сравнения;
- методика измерений. Описываются условия тестирования, включая аппаратные характеристики, стратегию минимизации шума в данных и методику вычисления времени выполнения диспетчеризации. Особое внимание уделено повторяемости результатов и исключению влияния сторонних факторов;

- результаты и анализ. Представлены количественные данные по времени поиска специализаций для каждого подхода, включая ППП с разным количеством параметров. Анализируется масштабируемость решений и их чувствительность к увеличению числа специализаций.

3.1. ОПИСАНИЕ СРАВНИВАЕМЫХ ВАРИАНТОВ

Для сравнения в тестах на производительность (далее – бенчмарки) были выбраны языки программирования с характерными подходами к реализации мультиметодов:

- в языке программирования Julia [31] используются оптимизированные хеш-таблицы, применяемые для поиска обработчиков специализаций;
- язык программирования C++ является типичным представителем ОО-языков со статической типизацией, использующий диспетчеризацию через таблицы виртуальных методов и обладающий при этом высокой производительностью;
- ручная реализация поиска альтернативных обработчиков с использованием оператора `switch` является типичной для статически типизированного и высокопроизводительного языка программирования C;
- рассматриваемый в работе процедурно-параметрический подход реализуется в языке программирования RPS [34] в качестве расширения языка C [30].

Такой набор решений покрывает распространенные подходы к реализации мультиметодов и позволяет оценить производительность процедурно-параметрического подхода по сравнению с ними. Полный код бенчмарков приведен в репозитории [35].

3.1.1. РЕАЛИЗАЦИЯ ДИСПЕТЧЕРИЗАЦИИ В JULIA

Для повышения производительности диспетчеризации мультиметодов в компиляторе языка Julia реализовано два уровня кеширования для поиска комбинаций специализаций: быстрый и основной. Исследуемые решения были взяты из открытого репозитория компилятора Julia [32] и адаптированы для тестирования различных комбинаций аргументов мультиметодов.

Быстрый кеш представляет собой массив из 4096 записей, реализующий политику LRU (least-recently used) для хранения адресов последних вызовов специализаций. Каждая запись содержит информацию о типах аргументов и указатель на соответствующую специализацию.

Основной кеш специализаций мультиметодов основан на двухуровневой хеш-таблице:

- словаре, хранящем точные совпадения кортежей типов аргументов с уже скомпилированными специализациями. Здесь происходит поиск по точному совпадению типов;
- обобщенной таблице всех возможных специализаций. Здесь происходит выбор наиболее подходящей специализации, для которой нет точного совпадения типов. Для случаев с двумя параметрами применяются специализированные алгоритмы, ускоряющие поиск. С более чем тремя параметрами используется более медленный обобщенный алгоритм.

Для тестирования разработаны следующие варианты бенчмарков:

- быстрый кеш – мультиметод с двумя параметрами;
- основной кеш – мультиметод с двумя параметрами (используется оптимизированный путь поиска в хеш-таблице);
- основной кеш – мультиметод с тремя параметрами (используется оптимизированный путь поиска в хеш-таблице);
- основной кеш – мультиметод с пятью параметрами (используется обобщенный путь поиска в хеш-таблице).

Общий алгоритм поиска специализации реализован следующим образом. При вызове мультиметода сначала осуществляется проверка в массиве закешированных специализаций по адресу места вызова [33]). При совпадении адреса и типов аргументов возвращается готовая специализация. Если совпадение не найдено, поиск продолжается в общем кеше. При отсутствии совпадений в общем кеше выполняется проверка среди всех скомпилированных методов. Если же вызываемый код отсутствует среди скомпилированных, то после компиляции он добавляется в оба кеша для ускорения последующих вызовов.

Комбинация кеша двух уровней обеспечивает минимальные накладные расходы на поиск специализаций, минимизируя необходимость прохода по иерархии типов. Это позволяет сохранить гибкость динамической типизации Julia при сохранении высокой производительности поиска специализаций мультиметодов.

3.1.2. РЕАЛИЗАЦИЯ ДИСПЕТЧЕРИЗАЦИИ В ООП

В ООП используется двойная диспетчеризация с одинаковыми типами (базовыми классами) первого и второго аргументов. При реализации мультиметода для трех аргументов добавляется еще один производный класс, а диспетчеризация расширяется дополнительными виртуальными методами под все комбинации типов, что требует изменения числа методов в наследниках. Каждая специализация мультиметода фиксирует вызов через инкремент глобального счетчика (полный код доступен в репозитории [35]). В результате для ОО-подхода реализованы следующие бенчмарки:

- диспетчеризация с двумя параметрами, использующая наследование из двух дочерних классов;
- диспетчеризации с тремя параметрами при использовании наследования из трех дочерних классов.

3.1.3. РЕАЛИЗАЦИЯ МУЛЬТИМЕТОДА В ПРОЦЕДУРНОМ ПОДХОДЕ

В процедурном программировании мультиметод может быть реализован через механизм вложенных переключателей или условных операторов, проверяющих тип специализации или ее признак, если она реализуется в статически типизированном языке. Например, в языке С признак может задаваться перечислимый типом. Для мультиметода, содержащего N аргументов, используется N вложенных операторов переключения (switch). В результате процесс анализа аргументов и выполнения мультиметода выглядит следующим образом.

- Первоначально во внешнем переключателе осуществляется определение типа первого аргумента. Для каждого возможного типа этого аргумента

существует вложенный переключатель, анализирующий тип второго аргумента. Этот процесс повторяется для всех N аргументов мультиметода.

- После проведенного анализа типов всех аргументов выполняется команда-обработчик специализации для полученной комбинации аргументов.

Варианты бенчмарков, используемых для проверки производительности:

- мультиметод содержит два аргумента при анализе двух альтернатив, определяющих специализации;
- мультиметод имеет два аргумента, для каждого из которых анализируется 10 альтернатив;
- мультиметод содержит три аргумента, для каждого из которых имеется по три альтернативы.

3.1.4. РЕАЛИЗАЦИЯ ДИСПЕТЧЕРИЗАЦИИ В ПРОЦЕДУРНО-ПАРАМЕТРИЧЕСКОМ ПОДХОДЕ

Диспетчеризация в процедурном-параметрическом подходе не требует дополнительных конструкций, так как основана на встроенных механизмах поддержки множественного полиморфизма. Ниже приведен пример кода на языке программирования RPS с использованием ПП-расширений для мультиметода с двумя параметрами, типы которых относятся к одному обобщению, содержащему две специализации.

Определение типов данных:

```
struct Specialization1{};
struct Specialization2{};
struct Object{}< Specialization1; Specialization2; >;
```

Определение функции, возвращающей объект со случайно выбранным типом специализации:

```
Object* get_object_with_randomized_type()
```

мультиметод с двумя параметрами:

```
void MultiMethodCollide<Object* f1, Object* f2>();
```

мультиметод с тремя параметрами:

```
void MultiMethodCollide3D<Object* f1, Object* f2, Object* f3>();
```

мультиметод с пятью параметрами:

```
void MultiMethodCollide5D<
    Object* f1, Object* f2, Object* f3, Object* f4, Object* f5>();
```

мультиметод с десятью параметрами:

```
void MultiMethodCollide10D<Object* f1, Object* f2, Object* f3,
    Object* f4, Object* f5, Object* f6, Object* f7,
    Object* f8, Object* f9, Object* f10>();
```

Количество специализаций в обобщении не увеличивалось, так как экспериментально это не оказывает влияния на производительность из-за особенностей реализации: диспетчеризирующая функция, в которой реализуется алгоритм поиска специализации мультиметода, обладает сложностью $O_{\text{алг}}$, которая зависит не от количества специализаций типов, а от числа параметров:

$$O_{\text{алг}} = (1 \text{ сложение} + 1 \text{ умножение}) \times (N_{\text{params}} - 1),$$

где N_{params} – количество полиморфных параметров мультиметода.

В качестве следствия этого свойства можно отметить, что время поиска также не зависит от того, сколько специализаций мультиметода описано поль-

зователем. То есть если пользователем описаны отдельные обработчики для каждой комбинации параметров, то время поиска соответствующего обработчика не будет отличаться от случая, когда не описан ни один обработчик (то есть используется обработчик по умолчанию).

3.2. МЕТОДИКА ИЗМЕРЕНИЙ

Исследования проводились на ноутбуке Huawei MateBook X Pro MRGF-X (Intel Core i7-1260P, 16 ГБ LPDDR5, SSD YMTC YM00D216, Ubuntu 24.04). Для минимизации шума и обеспечения воспроизводимости было сделано следующее:

- тесты выполнялись на фиксированном высокопроизводительном ядре (CPU 1) с помощью taskset [36];
- тактовая частота ядра зафиксирована на уровне 4,7 ГГц, кеш: L1 – 80 кБ, L2 – 1,25 МБ, L3 – 18 МБ (общий);
- случайный выбор типов аргументов мультиметодов реализован через rand() с инициализацией srand() для детерминированности;
- каждый бенчмарк запускался 10 млн раз, повторяемость проверялась на 100 итерациях.

Проводилось два варианта тестов.

1. Подготовительный – измеряет базовые накладные расходы (создание объектов, вызов функции):

```
void multimethod_empty(Object* obj1, Object* obj2)
{ vol_global_int++; }
```

2. Основной – использует вызов:

```
void multimethod_real(Object* obj1, Object* obj2)
{ /* Вызов специализации */ }
```

Разница между результатами основного и подготовительного тестов являлась оценкой производительности диспетчеризации. Для статистической достоверности рассчитывались среднее арифметическое и среднеквадратичное отклонение.

Все диспетчеризующие функции реализованы в отдельных единицах компиляции. Сборка выполнялась с флагом O3, без LTO для минимизации влияния оптимизаций. Метод уменьшает влияние побочных факторов (создание объектов, вызов функций), обеспечивая точное сравнение алгоритмов диспетчеризации. Статистическая обработка данных подтверждает надежность полученных результатов.

3.3. РЕЗУЛЬТАТЫ ЗАМЕРОВ И ИХ АНАЛИЗ

В результате проведенных замеров по описанной выше методике была получена скорость нахождения нужного обработчика специализации для переданных аргументов (суммарно для одного прохода измерений, т. е. 10 млн поисков). Значения приведены в таблице. В первом столбце в квадратных скобках для каждого варианта указаны сокращения, используемые на графике.

Графическая интерпретация результатов представлена на рис. 2.

Скорость поиска специализации мультиметода**The speed of the multimethod specialization search**

Наименование бенчмарка	Скорость выполнения, с
ПП-мультиметод, 2 параметра [PP2]	0,039
ПП-мультиметод, 3 параметра [PP3]	0,041
ПП-мультиметод, 5 параметров [PP5]	0,114
ПП-мультиметод, 10 параметров [PP10]	0,174
На основе оператора switch, 2 параметра [Switch2]	0,089
На основе оператора switch, 3 параметра [Switch3]	0,102
На основе оператора switch, 2 параметра, 10 специализаций типа [Switch2x10]	0,528
Мультиметоды Julia. Поиск в быстром кеше, 2 параметра [JuliaFast]	0,173
ООП-диспетчеризация, 2 параметра [OOPDispatch2]	0,047
ООП-диспетчеризация, 3 параметра [OOPDispatch3]	0,549
Мультиметоды Julia. Поиск в основном кеше, 2 параметра [Julia2]	9,427
Мультиметоды Julia. Поиск в основном кеше, 3 параметра [Julia3]	16,65
Мультиметоды Julia. Поиск в основном кеше, 5 параметров [Julia5]	35,20

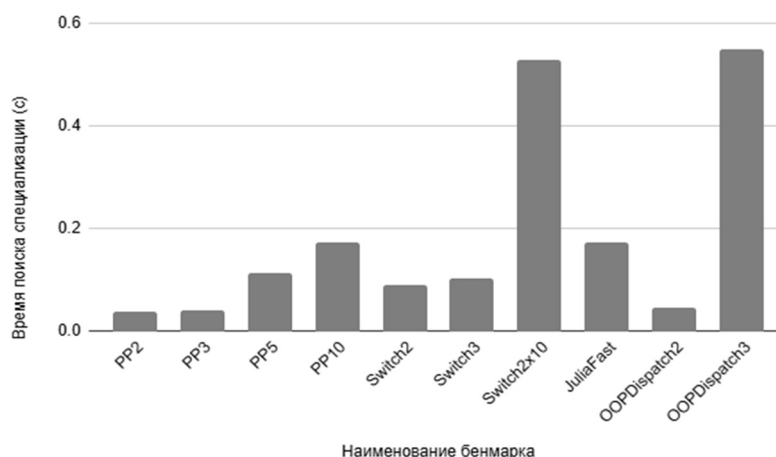


Рис. 2. Время выполнения поиска специализации мультиметода

Fig. 2. Execution time of the multimethod specialization search

На основе полученных результатов можно сделать вывод, что реализация поиска специализации мультиметодов, основанного на ПП-конструкциях, обладает большей производительностью, чем иные рассматриваемые решения. Производительность этого подхода не зависит от количества специализаций, которое может значительно ухудшить производительность других подходов (например, основанных на операторе switch).

При увеличении количества параметров мультиметодов время поиска обработчика увеличивается практически линейно (с учетом особенностей работы процессора, таких как кеш и предсказатель переходов), что экспериментально подтверждает расчеты, выполненные в п. 3.1.4. Время для расчета 10 парамет-

ров – 0,174 с, что примерно в 5 раз больше времени для расчета двух параметров ($0,174/0,039 = 4,46$ с).

Стоит отметить также высокую производительность подходов, использующих ОО-диспетчеризацию, но этот подход немасштабируемый и уже для трех параметров показывает значительное (примерно в 10 раз) ухудшение производительности (количество косвенных вызовов увеличивается с 6 до 9, и в несколько раз увеличивается размер двоичного кода, поэтому код перестает попадать в кеш линии L1).

Необходимо отметить, что производительность поиска в хеш-таблице значительно ниже по сравнению с другими способами. Но именно такой подход (с различными вариациями) используется во многих языках программирования (Julia, CLOS, R, Clojure) благодаря его гибкости. Он позволяет динамически подключать новые специализации, что невозможно при использовании ОО-диспетчеризации или при использовании решения с применением оператора switch.

ЗАКЛЮЧЕНИЕ

Предлагаемое расширение языка программирования С конструкциями, обеспечивающими инструментальную поддержку процедурно-параметрической парадигмы программирования, позволяет разрабатывать гибкие эволюционно расширяемые программы. При этом поддерживается добавление как новых альтернативных данных, так и функций без изменения ранее написанного кода. Добавленные конструкции сохраняют обратную совместимость с языком С. Это позволяет использовать ранее написанные программы и библиотеки, а также обеспечивать их рефакторинг для создания более надежных и безболезненно расширяемых программ.

Инструментальная поддержка множественного полиморфизма позволяет увеличить производительность программ и повысить эффективность их разработки за счет непосредственного описания мультиметодов от произвольного числа аргументов. Это снижает стоимость поддержки программных продуктов во многих сферах: от математических библиотек и программных средств для машинного обучения до компиляторов и игровых приложений.

Дальнейшие исследования ПП-подхода проводятся по различным направлениям:

- повышение эффективности методов компиляции, что позволит создавать более эффективный код;
- создание инструментальных средств и интегрированных сред разработки, повышающих эффективность процесса создания программ;
- разработка новых языков императивного и функционального программирования, непосредственно поддерживающих разнообразные возможности, предлагаемые ПП-парадигмой;
- исследование методологических подходов к использованию процедурно параметрической парадигмы на этапах проектирования.

Помимо этого, при формировании альтернативных данных и функций, обеспечивающих их обработку, применяются оригинальные решения и подходы, что позволяет проводить исследования, связанные с расширением теории типов языков программирования.

СПИСОК ЛИТЕРАТУРЫ

1. *Shafer D., Ritz D.A.* Practical Smalltalk: using Smalltalk/V. – Springer, 1991. – 233 p.
2. *Hunt J.* Advanced guide to Python 3 programming. – Springer, 2019. – 497 p.
3. *Gregoire M.* Professional C++. – John Wiley & Sons, 2018. – 1122 p.
4. *Sciore E.*, Java program design. – Apress Media, 2019. – 473 p.
5. *Albahari J., Albahari B.* C# 6.0 Pocket Reference. – O'Reilly Media, 2016. – 224 p.
6. *Freeman A.* Pro Go: the complete guide to programming reliable and efficient software using Golang. – Apress Media, 2022. – 1076 p.
7. *Blandy J., Orendorff J., Tindall L.F.S.* Programming Rust. – O'Reilly Media, 2021. – 1470 p.
8. Report on the programming language Haskell: a non-strict, purely functional language version 1.2 / P. Hudak, S. Peyton Jones, P. Wadler, B. Boutel, J. Fairbairn, J. Fasel, M.M. Guzmán, K. Hammond, J. Hughes, T. Johnsson, D. Kieburtz, R. Nikhil, W. Partain, J. Peterson // ACM SIGPLAN Notices. – 1992. – Vol. 27 (5). – P. 1–164.
9. *Peyton Jones S., Jones M., Meijer E.* Type classes: an exploration of the design space // Proceedings of the Haskell Workshop. – Amsterdam, Netherlands, 1997. – P. 1–16.
10. *Ingalls D.H.H.* A simple technique for handling multiple polymorphism // ACM SIGPLAN Notices. – 1986. – Vol. 21 (11). – P. 347–349.
11. Design patterns: Elements of reusable object-oriented software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Addison-Wesley Professional, 1994. – 416 p.
12. *Alexandrescu A.* Modern C++ design: Generic programming and design patterns applied. – Addison-Wesley Professional, 2001. – 360 p.
13. *Meyers S.* More effective C++: 35 new ways to improve your programs and designs // Addison-Wesley Professional, 1996. – 318 p.
14. *Легалов А.И.* ООП, мультиметоды и пирамидальная эволюция // Открытые системы. СУБД. – 2002. – № 3. – С. 41–45.
15. *Demichiel L.G.* Overview: The common Lisp Object System // Lisp and Symbolic Computation. – 1988. – Vol. 1 (2). – P. 227–244.
16. *Stroustrup B.* The design and evolution of C++. – Addison-Wesley Publishing, 1994. – 427 p.
17. Rust library for Multimethods support – URL: <https://github.com/fcard/multimethods.rs> (accessed: 28.11.2025).
18. *Legalov A.I., Kosov P.V.* C language extension to support procedural-parametric polymorphism // Automatic Control and Computer Sciences. – 2024. – Vol. 58 (7). – P. 762–778. – DOI: 10.3103/S014641162470024X.
19. Polymorphic symmetric multiple dispatch with variance / G. Park, J. Hong, G.L. Steele Jr., S. Ryu // Proceedings of the ACM on Programming Languages. – 2019. – Vol. 3. – P. 1–28.
20. *Mugridge W.B., Hamer J., Hosking J.G.* Multi-methods in a statically-typed programming language // ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming. – Springer, 1991. – P. 307–324.
21. *Leavens G.T.* Verifying object-oriented programs that use subtypes. – Massachusetts Institute of Technology, 1989. – 211 p.
22. The Cecil Language. Specification and Rationale. Version 3.2. – URL: <https://projectswest.cs.washington.edu/research/projects/cecil/www/Release/doc-cecil-lang/cecil-spec.pdf> (accessed: 28.11.2025).
23. *Leavens G.T., Millstein T.D.* Multiple dispatch as dispatch on Tuples // ACM SIGPLAN Notices. – 1998. – Vol. 33 (10). – P. 374–387.
24. *Amiel E., Gruber O., Simon E.* Optimizing multi-method dispatch using compressed dispatch tables // ACM SIGPLAN Notices. – 1994. – Vol. 29 (10). – P. 244–258.
25. A fast method dispatcher for compiled languages with multiple inheritance / R. Dixon, T. McKee, M. Vaughan, P. Schweizer // ACM SIGPLAN Notices. – 1989. – Vol. 24 (10). – P. 211–214.
26. *Chambers C., Chen W.* Efficient multiple and predicated dispatching // ACM SIGPLAN Notices. – 1999. – Vol. 34 (10). – P. 238–255.
27. *Foote B., Johnson R.E., Noble J.* Efficient multimethods in a single dispatch language // ECOOP'05: Proceedings of the 19th European Conference on Object-Oriented Programming. – Springer, 2005. – P. 337–361.
29. *Легалов А.И., Швец Д.А.* Процедурный язык с поддержкой эволюционного проектирования // Научный вестник НГТУ. – 2003. – № 2 (15). – С. 25–38.

29. Легалов А.И., Косов П. Процедурно-параметрическое расширение языка программирования C. Синтаксис и семантика. – URL: <http://www.softcraft.ru/ppp/ppc/> (accessed: 28.11.2025).
30. Julia Documentation. – URL: <https://docs.julialang.org/en/v1/> (accessed: 28.11.2025).
31. Julia Compiler Source Code. Generic Functions. – URL: <https://github.com/JuliaLang/julia/blob/v1.11.3/src/gf.c> (accessed: 28.11.2025).
32. GNU Manual. Getting the Return or Frame Address of a Function. – URL: <https://gcc.gnu.org/onlinedocs/gcc/Return-Address.html> (accessed: 28.11.2025).
33. Язык программирования PPC. – URL: <https://gitverse.ru/kpdev/llvm-project/content/pp-extension-v2> (accessed: 15.06.2025).
34. Benchmarks for multiple dispatch mechanisms. – URL: <https://gitverse.ru/kpdev/multi-method-benchmarks> (accessed: 15.06.2025).
35. taskset – Linux manual page. – URL: <https://man7.org/linux/man-pages/man1/taskset.1.html> (accessed: 28.11.2025).

Косов Павел Владимирович, аспирант факультета компьютерных наук НИУ «Высшая школа экономики». Основное направление научных исследований – разработка компиляторов и языков программирования, оптимизация в LLVM, эволюционная разработка программного обеспечения. Имеет 10 печатных работ. E-mail: pvkosov@hse.ru

Легалов Александр Иванович, доктор технических наук, профессор факультета компьютерных наук НИУ «Высшая школа экономики». Научные интересы: технологии программирования, эволюционная разработка программного обеспечения, архитектурно-независимое параллельное программирование, языки программирования. Имеет более 200 печатных работ. E-mail: alegalov@hse.ru

Kosov Pavel V., is a graduate student at the Faculty of Computer Science at the Higher School of Economics. His research interests are related to the development of programming languages and compilers, LLVM optimization, and evolutionary software development. He has more than 10 publications. E-mail: pvkosov@hse.ru

Legalov Alexander I., D.Sc. (Eng.), professor at the Faculty of Computer Science at the Higher School of Economics. His research interests include software engineering, evolutionary software development, architecture-independent parallel programming, and programming languages. He has more than 200 publications. E-mail: alegalov@hse.ru

DOI: 10.17212/2782-2001-2025-4-69-84

Performance comparison for different approaches to implementing multimethods*

P.V. KOSOV^a, A.I. LEGALOV^b

Higher School of Economics, National Research University, 20 Myasnitskaya Street, Moscow, 101000, Russian Federation

^a pvkosov@hse.ru ^b alegalov@hse.ru

Abstract

Multimethods, defined as functions polymorphic to multiple arguments, are regularly encountered in software development. They are directly implemented in a number of programming languages, for which compilers automatically generate the corresponding machine code using various approaches. The lack of such an implementation is compensated either by writing codes directly or by including them in programming language libraries. The variety of multimethod implementation algorithms is often related to the need to achieve the required quality criteria, one of which is evolutionary extensibility without changing the previously written code. By hiding

* Received 01 July 2025.

internal solutions behind high-level constructs of programming languages, compilers form the final binding of alternative fragments of multimethods located in independent compilation units using various techniques, which ensures the necessary integrity at the output. At the same time, approaches to implementing multiple polymorphism have different performance and complexity of support in adding new combinations of arguments while extending functionality. One of the approaches that provides flexible implementation of multimethods is the procedural-parametric programming paradigm. It offers an effective solution based on parameterization of functional dependencies via multidimensional arrays. This approach provides an evolutionary extension of functionality and data without changing the previously written code, while maintaining security through static typing. The paper compares the procedural-parametric implementation of multimethods with existing approaches. For various algorithms, time estimates of the search for combinations of parameters necessary to call a multimethod are formed. Experiments demonstrate the effectiveness of procedural-parametric polymorphism compared to other approaches.

The results of the study make it possible to formulate practical recommendations on the choice of methods for implementing multimethods, depending on performance requirements. The proposed approach can be integrated into languages that support the procedural-parametric programming paradigm.

Keywords: programming languages, compilation, procedural-parametric programming, polymorphism, multiple polymorphism, multimethods, evolutionary software development, programming paradigms, software development methods

REFERENCES

1. Shafer D., Ritz D.A. *Practical Smalltalk: using Smalltalk/V*. Springer, 1991. 233 p.
2. Hunt J. *Advanced guide to Python 3 programming*. Springer, 2019. 497 p.
3. Gregoire M. *Professional C++*. John Wiley & Sons, 2018. 1122 p.
4. Sciore E. *Java program design*. Apress Media, 2019. 473 p.
5. Albahari J., Albahari B. *C# 6.0 Pocket Reference*. O'Reilly Media, 2016. 224 p.
6. Freeman A. *Pro Go: the complete guide to programming reliable and efficient software using Golang*. Apress Media, 2022. 1076 p.
7. Blandy J., Orendorff J., Tindall L.F.S. *Programming Rust*. O'Reilly Media, 2021. 1470 p.
8. Hudak P., Peyton Jones S., Wadler P., Boutel B., Fairbairn J., Fasel J., Guzmán M.M., Hammond K., Hughes J., Johnsson T., Kieburtz D., Nikhil R., Partain W., Peterson J. Report on the programming language Haskell: a non-strict, purely functional language version 1.2. *ACM SIGPLAN Notices*, 1992, vol. 27 (5), pp. 1–164.
9. Peyton Jones S., Jones M., Meijer E. Type classes: an exploration of the design space. *Proceedings of the Haskell Workshop*, Amsterdam, Netherlands, 1997, pp. 1–16.
10. Ingalls D.H.H. A simple technique for handling multiple polymorphism. *ACM SIGPLAN Notices*, 1986, vol. 21 (11), pp. 347–349.
11. Gamma E., Helm R., Johnson R., Vlissides J. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional, 1994. 416 p.
12. Alexandrescu A. *Modern C++ design: Generic programming and design patterns applied*. Addison-Wesley Professional, 2001. 360 p.
13. Meyers S. *More effective C++: 35 new ways to improve your programs and designs*. Addison-Wesley Professional, 1996. 318 p.
14. Legalov A.I. OOP, mul'timetody i piramidal'naya evolyutsiya [OOP, multimethods and pyramidal evolution]. *Otkrytye sistemy. SUBD = Open Systems. DBMS*, 2002, no. 3, pp. 41–45.
15. Demichiel L.G. Overview: The common Lisp Object System. *Lisp and Symbolic Computation*, 1988, vol. 1 (2), pp. 227–244.
16. Stroustrup B. *The design and evolution of C++*. Addison-Wesley Publishing, 1994. 427 p.
17. *Rust library for Multimethods support*. Available at: <https://github.com/fcard/multimethods.rs> (accessed 28.11.2025).
18. Legalov A.I., Kosov P.V. C language extension to support procedural-parametric polymorphism. *Automatic Control and Computer Sciences*, 2024, vol. 58 (7), pp. 762–778. DOI: 10.3103/S014641162470024X.

19. Park G., Hong J., Steele Jr. G.L., Ryu S. Polymorphic symmetric multiple dispatch with variance. *Proceedings of the ACM on Programming Languages*, 2019, vol. 3, pp. 1–28.
20. Mugridge W.B., Hamer J., Hosking J.G. Multi-methods in a statically-typed programming language. *ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming*. Springer, 1991, pp. 307–324.
21. Leavens G.T. *Verifying object-oriented programs that use subtypes*. Massachusetts Institute of Technology, 1989. 211 p.
22. *The Cecil Language. Specification and Rationale*. Version 3.2. Available at: <https://projectsweb.cs.washington.edu/research/projects/cecil/www/Release/doc-cecil-lang/cecil-spec.pdf> (accessed 28.11.2025).
23. Leavens G.T., Millstein T.D. Multiple dispatch as dispatch on Tuples. *ACM SIGPLAN Notices*, 1998, vol. 33 (10), pp. 374–387.
24. Amiel E., Gruber O., Simon E. Optimizing multi-method dispatch using compressed dispatch tables. *ACM SIGPLAN Notices*, 1994, vol. 29 (10), pp. 244–258.
25. R. Dixon, T. McKee, M. Vaughan, P. Schweizer, A fast method dispatcher for compiled languages with multiple inheritance // *ACM SIGPLAN Notices*, Issue:10, Volume:24, 1989 – 211–214 p.
26. Chambers C., Chen W. Efficient multiple and predicated dispatching. *ACM SIGPLAN Notices*, 1999, vol. 34 (10), pp. 238–255.
27. Foote B., Johnson R.E., Noble J. Efficient multimethods in a single dispatch language. *ECOOP'05: Proceedings of the 19th European Conference on Object-Oriented Programming*. Springer, 2005, pp. 337–361.
28. Legalov A.I., Shvets D.A. Protsedurnyi yazyk s podderzhkoi evolyutsionnogo proektirovaniya [Procedural language with support for evolutionary design]. *Nauchnyi vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta = Science bulletin of the Novosibirsk state technical university*, 2003, no. 2 (15), pp. 25–38.
29. Legalov A.I., Kosov P. Protsedurno-parametricheskoe rasshirenie yazyka programmirovaniya C. *Sintaksis i semantika* [Procedural-parametric extension of C programming language. Syntax and semantics]. Available at: <http://www.softcraft.ru/ppp/ppc/> (accessed 28.11.2025).
30. *Julia Documentation*. Available at: <https://docs.julialang.org/en/v1/> (accessed 28.11.2025).
31. *Julia Compiler Source Code. Generic Functions*. Available at: <https://github.com/Julia-Lang/julia/blob/v1.11.3/src/gf.c> (accessed 28.11.2025).
32. *GNU Manual. Getting the Return or Frame Address of a Function*. Available at: <https://gcc.gnu.org/onlinedocs/gcc/Return-Address.html> (accessed 28.11.2025).
33. *Yazyk programmirovaniya PPC* [PPC programming language]. Available at: <https://gitverse.ru/kpdev/llvm-project/content/pp-extension-v2> (accessed 15.06.2025).
34. *Benchmarks for multiple dispatch mechanisms*. Available at: <https://gitverse.ru/kpdev/multimethod-benchmarks> (accessed 15.06.2025).
35. *taskset – Linux manual page*. Available at: <https://man7.org/linux/man-pages/man1/taskset.1.html> (accessed 15.06.2025).

Для цитирования:

Косов П.В., Легалов А.И. Сравнение производительности различных подходов реализации диспетчеризации для мультиметодов // Системы анализа и обработки данных. – 2025. – № 4 (100). – С. 69–84. – DOI: 10.17212/2782-2001-2025-4-69-84.

For citation:

Kosov P.V., Legalov A.I. Sravnenie proizvoditel'nosti razlichnykh podkhodov realizatsii dispetcherizatsii dlya mul'timetodov [Performance comparison for different approaches to implementing multi-methods]. *Sistemy analiza i obrabotki dannykh = Analysis and data processing systems*, 2025, no. 4 (100), pp. 69–84. DOI: 10.17212/2782-2001-2025-4-69-84.