

МЕТОДЫ И СРЕДСТВА ПОВЫШЕНИЯ ВРЕМЕННОЙ ЭФФЕКТИВНОСТИ СТРУКТУР ДАННЫХ

Анотация. Выполнен обзор и анализ известных методов проектирования и реализации структур данных, используемых в информационных системах с повышенными требованиями к временным характеристикам. Рассмотрены методы и средства проектирования на высоком уровне абстракции, позволяющие формировать эффективные структуры данных на физическом уровне в оперативной памяти. Выделены три направления объективных показателей оценки временных характеристик структур данных: амортизационный анализ, вычислительная сложность, средство на основе вычислительной сложности и временной эффективности.

Ключевые слова: структуры данных, логическая реализация данных, физическая реализация данных, эффективность, показатель эффективности.

Введение

Стремительный рост количества обрабатываемых данных обостряет проблему эффективной обработки данных в оперативной памяти (ОП). Это в свою очередь актуализирует проблему автоматизированного подбора структур данных для размещения в ОП. Правильный выбор организации данных для заданного алгоритма позволяет существенно улучшить его показатели качества [1, 2, 3].

Цель работы: выделить основные направления исследования в области проектирования, разработки и эксплуатации программного обеспечения, требующего значительных временных ресурсов, путем повышения временной эффективности за счет выбора способа организации структур данных.

Проектирование эффективных структур данных на абстрактном уровне

Для решения задач оптимизации структур данных нужно формализовать взаимодействие структур данных с остальной програм-

мой. Для этого нужно разделить представление структур данных на несколько уровней. Одно из таких разделений представлено в [4]. В работе рассматривается структура данных как конструктивная система с двумя уровнями организации: логическом и физическом и системно-изоморфным соответствием между ними.

Структура данных (СД) определяется как совокупность элементов данных и отношений между ними. Это определение СД отражает только логическую организацию данных (ЛОД), которая должна быть зафиксирована любым подходящим способом в физической организации данных при разработке программного обеспечения.

Физическая организация данных (ФОД) также представляется некоторой СД, которая носит конкретный характер и определяет размещение данных в памяти, методы доступа к ним и способы реализации операций их обработки.

СД рассматривается как конструктивная система (КСД). Внешние объекты взаимодействуют с КСД через интерфейс, предоставляющий доступ к логическому уровню организации данных, который является виртуальным. Реальная организация данных и их обработка осуществляются на физическом уровне, который полностью поддерживает логическое представление данных. Соответственно, нет необходимости организовывать и размещать данные в ОП на логическом уровне, а достаточно формировать их на основе отображения ЛОД-ФОД. Интерфейс доступа к данным в КСД представляется в виде предметно-ориентированного языка (Domain Specific Language, DSL), специфицирующего логическую СД и методы ее обработки.

Поскольку реализация DSL имеет доступ к информации высокого уровня, его оптимизации могут применяться в больших масштабах (например, в глобальных преобразованиях программ) и часто производят более эффективный код чем при работе среднего программиста.

Подобное разделение также используется в [5], где рассматриваются предметно-ориентированные языки, которые позволяют лучше выражать конструкции, присущие заданной предметной области. Вводится понятие генераторов-компиляторов для предметно-ориентированных языков. В статье используется генерирование реализаций на основе компонентной композиции. Основная часть генератора – это распознавание простых блоков для программного обеспече-

ния заданной предметной области. Генератор посредством преобразований осуществляет конвертацию конструкций DSL в компоненты целевого языка. Преимуществом такого подхода является расширяемость. Из небольшого количества компонентов, которые могут быть собраны различными способами, возможно создание большого количества уникальных реализаций конструкций DSL.

Абстрагирование от деталей реализации приводит к более простым и более детальным спецификациям. Основной целью проектирования DSL заключается в повышении уровня абстракции, который используется пользователем.

Таким образом, можно радикально изменить реализацию спецификации без необходимости модифицировать исходный код приложения. Выделение того что должно быть сделано, а не то, как это должно быть сделано, является первым шагом в процессе абстракции. Ни один инструментарий на основе DSL не пригоден для использования, если он не предоставляет приемлемую производительность. В противном случае абстрактная спецификация может использоваться лишь в качестве примера дизайна, а фактическая реализация будет результатом мануального уточнения спецификации. Задача разработчика DSL заключается в выявлении уровня абстракции, который поддается автоматизированной аргументации для контроля ошибок и оптимизации. Таким образом, могут быть получены преимущества как абстрактности, так и эффективности.

Адаптация структур данных в процессе эксплуатации ПО

Иной подход оптимизации структур данных основывается на построении модели системы и ее анализе в разрезе использования структур данных. Например, в [6] описывается построение модели системы реального времени на основе трассировки ее работы. Построенная модель позволяет осуществлять планирование добавления новой функциональности, а также анализ воздействия изменений на модель. Абстрактная модель позволяет отлаживать систему без прямого взаимодействия с системой. Также она позволяет автоматизировано составлять документацию и предоставляет возможность новым сотрудникам быстрее знакомится с системой.

Модель строится на основе языка ART-ML. Её построение выполняется с помощью семплирования работы системы. На основе семплов строится дерево выполнения задач в системе (рис. 1).

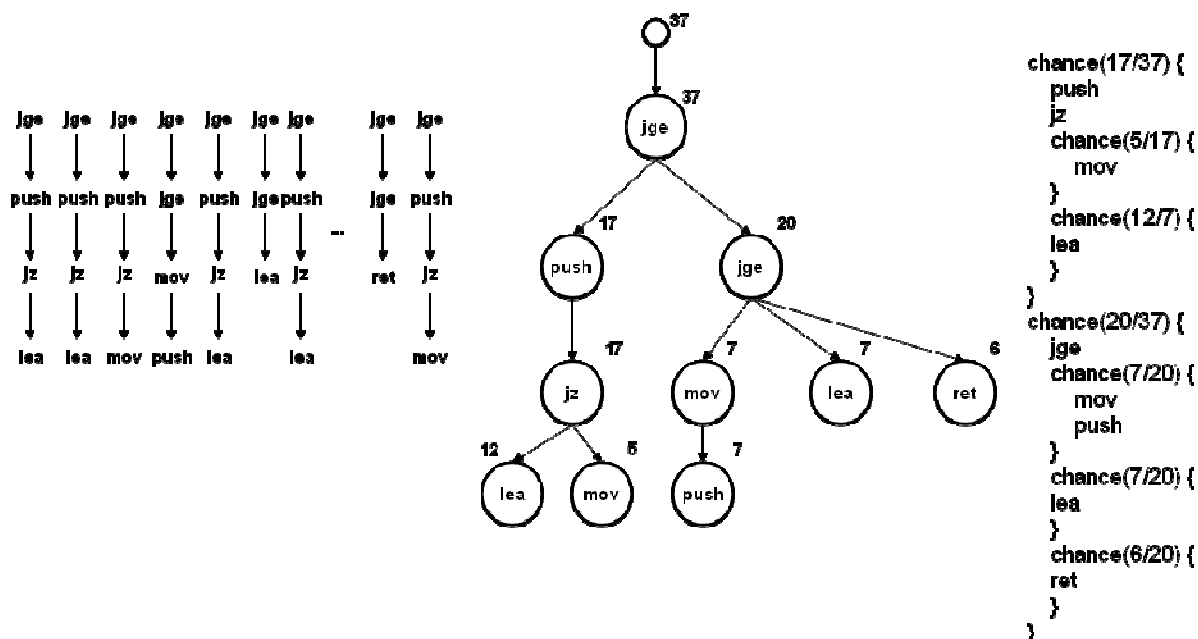


Рисунок 1 – Пример модели

Еще одна работа [7] сосредотачивает внимание на построении модели использования коллекций (контейнеров) исполнительской среды. Рассматривается программный комплекс, работающий на основе профилей выполнения приложения. Комплекс анализирует сценарии выделения, использования и освобождения памяти объектами коллекциями. На основе заданных разработчиками правил программный комплекс указывает какие коллекции стоит заменить, и в каких конкретных случаях следует изменить сценарий работы с ними. Например, правило `ArrayList: #contains>X & maxSize>Y → HashSet` указывает, что если у заданного объекта тип коллекции `ArrayList`, и среднее время вызовов методов `contains` для этого объекта превышает заданный порог X , и среднее максимальное количество элементов в коллекции выше заданного порога Y , в таком случае тип коллекции стоит заменить на `HashSet`. По данным авторов, такой подход позволяет уменьшить время выполнения алгоритмов приложения на 40-90%.

Другое использование такого подхода представлено в [8], где описывается проблема приложений, работающих с данными в виде графов. В работе описаны случаи, когда значения данных кардинально влияют как на алгоритмы их обработки, так и на память, используемую ими. Работа представляет приложение, выбирающее физическое представление из нескольких специализированных на основе ха-

рактических входных данных. Рассматриваются две структуры данных ADJList – представление графа в виде списка, которое оптимально для разреженных графов, и ADJMat – представление в виде матрицы, которое показывает лучшие характеристики для высокоплотных данных. В работе также представлена оценка времени, затрачиваемая на миграцию между представлениями, и ее влияние на общий выигрыш работы алгоритмов обработки графов.

Проектирование структур данных с учетом особенностей аппаратных средств

Разработка новых технических средств позволяет существенно улучшить временные характеристики за счет развития алгоритмов обработки данных. Например, введение новых пакетных наборов инструкций SSE, SSE2, SSE3 позволяет работать с данными как с единым целым и более малым количеством инструкций [9].

Другим способом улучшения временных характеристик является разработка параллельных алгоритмов [10]. Например, в [11] решается прикладная задача по дифракции. Однопоточный код, изначально написанный без оптимизации, работает в 40 раз хуже, чем этот же код, переписанный с использованием библиотеки параллельных программ OpenMP [12] под данную задачу.

Методы и средства оценки временных характеристик алгоритмов и структур данных

Построение автоматизированных систем оптимизации структур данных требует выделение объективных оценок временной эффективности структур данных. Рассмотрим оценки эффективности СД, традиционно использующихся в прикладном программировании.

Теория сложности вычислений возникла из потребности сравнивать быстродействие алгоритмов [13], четко описывать их поведение (время исполнения и объем необходимой памяти) в зависимости от размера входных данных.

Количество элементарных операций, затраченных алгоритмом для решения конкретного экземпляра задачи, зависит не только от размера входных данных, но и от самих данных. Например, количество операций алгоритма сортировки вставками значительно меньше в случае, если входные данные уже отсортированы. Чтобы избежать подобных трудностей, рассматривают понятие временной сложности алгоритма в худшем случае.

Временная сложность алгоритма (в худшем случае) — это функция от размера входных данных, равная максимальному количеству элементарных операций, выполняемых алгоритмом для решения экземпляра задачи указанного размера.

Довольно часто для определения превосходства одного алгоритма относительно другого не нужно знать точную оценку их сложности [14, 15, 16]. Достаточно оценить асимптотику роста времени выполнения предоставленных алгоритмов. Если скорость роста времени выполнения одного алгоритма при увеличении количества входных данных ниже скорости роста времени другого алгоритма, первый алгоритм более эффективен.

Рассмотрим общий принцип асимптотических оценок [16]. Пусть есть некоторые функции $g(n)$ и $f(n)$. Если существуют такие значения c_1 , c_2 и n , и начиная с некоторого n_0 выполняется условие $c_1g(n) \leq f(n) \leq c_2g(n)$, при $c_1 > 0$, $c_2 > 0$ и $n > n_0$, принято говорить, что функция $f(n)$ ограничена функцией $g(n)$. В виде асимптотических обозначений ситуация выглядит следующим образом:

$$f(n) = \Theta(g(n)).$$

Если начиная с n_0 выполняется условие $f(n) \leq cg(n)$, при $c > 0$ и $n > n_0$, принято говорить, что функция $f(n)$ ограничена функцией $g(n)$ сверху. А если выполняется условие $cg(n) \leq f(n)$ — ограничена снизу. Первую оценку принято определять как O , вторую — Ω , и записывать соответственно:

$$f(n) = O(g(n)), \quad f(n) = \Omega(g(n)).$$

Задача определения асимптотической вычислительной сложности может быть решена различными методами. Рассмотрим те, что базируются на рекурсии. Оценка количества вычислений алгоритмом в таких методах может быть представлена как рекуррентное соотношение. Соотношение такого типа связывает оценку для одного количества входных данных и оценки для меньшего количества.

Известны методы, которые позволяют определить соответствующую асимптотической оценке, зная рекуррентное соотношение [15]:

- метод подстановки;
- метод итераций;
- общий метод согласно доказанной теоремы.

Метод подстановки или индуктивный метод заключается в следующем. Сначала нужно допустить возможную асимптотической оценку. Затем допущение доказывается с помощью математической индукции. Допущение делается на основе опыта, по аналогии с подобными методами. Иначе – методом подбора с постепенным уточнением оценки. Этот метод может использоваться как для нижних так и для верхних оценок.

Если допустить оценку не удастся, используется метод итераций. Смысл метода следующий: соотношение подставляется само в себя определенное количество раз, достаточное для того, чтобы представить соотношение в виде ряда или функции.

Амортизационный анализ используется для оценки времени выполнения последовательности взаимосвязанных операций над определенной структурой данных [15-17]. Для определения верхней границы оценки достаточно умножить максимальную продолжительность одной операции на общее количество операций в последовательности. Если операции с большим и малым временем выполнения чередуются, оценка может быть уточнена – она соответствует среднему времени выполнения. Оценки такого рода называются амортизационными.

Для выполнения амортизационного анализа нужно определить некоторую учетную стоимость каждой операции. Учетная стоимость может быть больше или меньше реального время выполнения операции, но такой, что: для любой последовательности операций фактическая суммарная продолжительность их выполнения не должна превышать сумму их учетных стоимостей. Если условие выполняется – учетная стоимость корректна.

Основные методы определения учетной стоимости [15]:

- метод группировки;
- метод предоплаты;
- метод потенциалов.

Метод группировки заключается в следующем: для каждого выполняется оценка времени $T(n)$ в худшем случае, где n – количество операций. Время выполнения одной операции оценивается как $T(n)/n$ и объявляется учетной стоимостью. Учетная стоимость всех операций будет считаться равной.

Метод предоплаты имеет следующее содержание: каждая операция получает свою учетную стоимость, причем полученные стоимости могут быть как больше, так и меньше реальных. Если учетная стоимость превышает реальную, разница рассматривается как резерв. Резерв связывается с другой операцией над структурой данных и считается предоплатой за следующее ее выполнение. Таким образом, за счет резерва компенсируется разница между учетной и реальной стоимостями для операций, чья учетная стоимость ниже реальной.

Метод потенциалов можно охарактеризовать как частный случай метода предоплаты. Разница заключается в том, что резерв не распределен между отдельными операциями, а накапливается в целом для структуры данных. Такой резерв называется «потенциалом» или «потенциальной энергией» структуры.

Структуры данных не обладают функциональностью, поэтому говорить об их временных характеристиках в этом смысле некорректно. Однако физическая реализация структур данных может в значительной степени влиять на временную эффективность программ и программных систем, использующих структурированные данные. В [18, 19] показано, что время операции доступа [20] к позиции в зависимости от физического размещения данных в ОП может отличаться на два порядка.

Под временной эффективностью структур данных будем понимать временную эффективность совокупности операций (алгоритмов) обработки данных.

Временную эффективность структуры данных можно определить на основе временной эффективности алгоритмов вида [21]:

$$A|_X^Y = \prod_i B_i |_{X_i}^{Y_i},$$

где B_i – алгоритмы обработки исследуемой структуры данных; X_i и Y_i – их области определения и значения, соответственно.

Для сравнения временной эффективности двух алгоритмов с учетом размещения и эксплуатации структуры данных в конкретной программно-аппаратной среде и фиксированных типов данных ее элементов воспользуемся следующим показателем степени превосходства одного (i -го) алгоритма над другим (j -тым):

$$SUP_{ij}|\bar{V}, \bar{X} = \frac{1}{N} \sum_{v \in \bar{V}} \sum_{x \in \bar{X}} \frac{t_j(v, x) - t_i(v, x)}{\max(t_i(v, x), t_j(v, x))}$$

где N – количество выполнений алгоритмов, \bar{V} – множество возможных значений объема данных, \bar{X} – множество возможных значений данных

Воспользуемся предложенным там же [22] методом оценки этого показателя, самой оценкой и оценкой ее доверительного интервала. S-оценка степени превосходства i -го алгоритма над j -м на основании N выполнений алгоритмов в области $\bar{\Omega}$:

$$S_{ij}|\bar{V}, \bar{X} = \frac{1}{\bar{N}} \sum_{k=1}^{\bar{N}} \frac{\bar{t}_{jk}(v, x) - \bar{t}_{ik}(v, x)}{\max(\bar{t}_{ik}(v, x), \bar{t}_{jk}(v, x))} \cdot 100\%,$$

где \bar{t}_{ik} – время выполнения i -го алгоритма при k -й реализации в конкретной программно-аппаратной среде.

Аналогично показатель области превосходства [22] i -го алгоритма над j -м:

$$R_{ij}|\bar{V}, \bar{X} = \frac{1}{\bar{N}} \sum_{k=1}^{\bar{N}} \text{sign}(\bar{t}_{jk}(v, x) - \bar{t}_{ik}(v, x)) \cdot 100\%, \quad \text{sign}(a) = \begin{cases} 1, & \text{если } a > 0 \\ 0, & \text{если } a \leq 0 \end{cases}$$

Оценка степени превосходства и области превосходства i -й структуры данных над j -й по критерию временной эффективности выполняется в области использования, в которой различие объема и значений данных определяется различным состоянием структуры данных перед и в процессе выполнения различных алгоритмов вида $A|_X^Y$.

Для измерения \bar{t}_{ik} и \bar{t}_{jk} необходимо моделирование среды эксплуатации структуры данных. Естественно, лучшим вариантом является выполнение замеров в той же программно-аппаратной среде, где структура эксплуатируется (ЭВМ, ОС, совместно выполняемые прикладные программы). Остается моделировать лишь процесс работы со структурой, то есть алгоритм $A|_X^Y$.

Выводы

Рассмотренные методы оценки эффективности структур данных и алгоритмов средствами амортизационного анализа, оценки вычислительной сложности требуют участия специалистов в области программирования и прикладной математики, могут применяться

только на этапе проектирования и не подразумевают использование эволюционных адаптивных структур данных.

Рассмотрены направления повышения временной эффективности структур данных, которые позволяют автоматически (адаптивно либо превентивно) формировать эффективные структуры данных в оперативной памяти, основываются не на объективных оценках, а на эмпирических показателях.

Представляется перспективным направление разработки средств адаптации структур данных в оперативной памяти на основе высокоуровневого их проектирования (на абстрактном и логическом уровне) с применением показателей временной эффективности.

ЛИТЕРАТУРА

1. Michanan J. Predicting data structures for energy efficient computing [Текст] / J. Michanan, R. Dewri, M. J. Rutherford // Green Computing Conference and Sustainable Computing Conference (IGSC). – 2015.– С. 1-8
2. ISO/IEC 9126-1:2001. Software Engineering – Product quality – Part 1: Quality model.
3. ДСТУ 2850-94. Програмні засоби ЕОМ. Показники і методи оцінювання якості.
4. Володин А. М. Конструктивные структуры данных [Текст] / А. М. Володин // Известия Пензенского государственного педагогического университета им. ВГ Белинского. – 2010. – № 22. – С. 118-122.
5. Smaragdakis Y. Distil: A Transformation Library For Data Structures. [Текст] / Y. Smaragdakis, D. S. Batory // Proceedings of the Conference on Domain-Specific Languages, October 15-17. – 1997.
6. Huselius J. Model synthesis for real-time systems [Текст] / J. Huselius, J. Andersson // Software Maintenance and Reengineering, Ninth European Conference – 2005. – С. 52-60
7. Shacham O. Chameleon: adaptive selection of collections [Текст] / O. Shacham, M. Vechev, E. Yahav // ACM Sigplan Notices. – 2009. – № 44.6. – С. 408-418.
8. Kusum A. Adapting Graph Application Performance via Alternate Data Structure Representation [Текст] / A. Kusum, I. Neamtiu, R. Gupta // 5th International Workshop on Adaptive Self-tuning Computing Systems. – 2015.
9. Streaming SIMD Extensions 3 (SSE3) [Электронный ресурс]. – Режим доступа:
URL: <http://softpixel.com/~cwright/programming/simd/sse3.php>.
10. Analysis of parallel algorithms [Электронный ресурс]. – Режим доступа:

URL: https://en.wikipedia.org/wiki/Analysis_of_parallel_algorithms

11. Mishchenko V. Accelerating the computation of the discrete currents method by modification takes into account the architectural features of a modern PCs. [Текст] / V. O. Mishchenko, B. V. ПАТОСНКИН // Вісник Харківського національного університету імені В. Н. Каразіна. Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління – 2015. – № 26. – С. 129-139.
12. The OpenMP® API specification for parallel programming [Електронний ресурс]. – Режим доступу: URL: <http://openmp.org/wp/>.
13. Sipser M. Introduction to the Theory of Computation [Текст] // Boston: Thomson Course Technology. – 2006.
14. Макконнелл Дж. Анализ алгоритмов. Вводный курс / Дж. Макконнелл. – М.: Техносфера, 2002. – 304 с.
15. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001. – 960 с.
16. Шинкаренко В. І. Оцінка часових характеристик структур даних на проектному рівні / В.І. Шинкаренко, Д.О. Петін, Г.В. Забула // Східно-Європейський журнал передових технологій. – 2014. – № 1. – С. 39-46.
17. Amortized Analysis Explained [Електронний ресурс]. – Режим доступу: URL: http://www.cs.princeton.edu/~fiebrink/423/AmortizedAnalysisExplained_Fiebrink.pdf
18. Шинкаренко В. И. Экспериментальные исследования алгоритмов в программно-аппаратных средах : монография / В. И. Шинкаренко. – Д.: Изд-во Днепропетр. нац. ун-та ж.-д. трансп. им. акад. В. Лазаряна, 2009. – 279 с.
19. Шинкаренко В.И. Временная оценка операций обработки структурированных данных с учетом конвейеризации и кэширования / В.И. Шинкаренко // Проблемы програмування. – 2006 – № 2-3. – С. 43-52.
20. Шинкаренко В. И. Повышение временной эффективности структур данных в оперативной памяти на основе адаптации / В. И. Шинкаренко, Г. В. Забула // Проблемы програмування. – 2012. – № 2-3. – С. 211-218.
21. Шинкаренко В.И. Структурные модели алгоритмов в задачах прикладного программирования Часть I. Формальные алгоритмические структуры / В.И. Шинкаренко, В.М. Ильман, В.В. Скалосуб // Кибернетика и системный анализ. – 2009 – №3. – С. 3-14.
22. Шинкаренко В.И. Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов / В.И. Шинкаренко // Проблемы программирования. – 2001. – № 3-4. – С. 31-39.