УДК 004.05+004.416.3+004.422.63

ПОВЫШЕНИЕ ВРЕМЕННОЙ ЭФФЕКТИВНОСТИ СТРУКТУР ДАННЫХ В ОПЕРАТИВНОЙ ПАМЯТИ НА ОСНОВЕ АДАПТАЦИИ

В.И. Шинкаренко, Г.В. Забула

Днепропетровский национальный университет железнодорожного транспорта
им. академика В. Лазаряна 49010, Днепропетровск, ул. академика Лазаряна, 2
e-mail: ccp@diit-70.dp.ua

Сформулировано понятие временной эффективности структур данных. Разработаны методы определения показателей соответствующего свойства структур данных. Предложен метод адаптивного синтеза структур данных в оперативной памяти. Физическая реализация структур данных адаптируется к программно-аппаратной среде эксплуатации и особенностям использования данных. Разработаны инструментальные средства синтеза адаптированных структур данных для виртуальных машин. Они позволяют существенно повысить временную эффективность структур данных при решении задач распределенных вычислений в гетерогенных вычислительных сетях. Основные результаты подтверждаются выполненным компьютерным экспериментом.

Scientific concept of time's efficiency of data structures was proposed. Methods of determining parameters of appropriate properties of data structures were developed. The method of synthesis of adaptive data structures in memory has been proposed. The physical implementations of data structures have been adapted with respect to the software, hardware and operations with the data. Tools tailored for synthesis of data structures for virtual machines were developed. They substantially increase data structures time's efficiency in solving problems of distributed computing in heterogeneous computing networks. The main results are confirmed through computer experiments.

Введение

При решении задач связанных с обработкой больших объемов данных и необходимостью хранения значительной их части в оперативной памяти, с повышенными требованиями к временной эффективности огромное значение имеет насколько эффективным по временным характеристикам является эксплуатируемое ПО. В частности, это касается задачи распределенных вычислений в гетерогенных программно-аппаратных средах.

В таких случаях необходимо особое внимание обратить на разработку эффективных алгоритмов и эффективных структур данных. И хотя существует множество методов и способов адаптации и оптимизации алгоритмов к конкретным программно-аппаратным средам, однако они не оптимизируют, не адаптируют и не изменяют заданные в программе структуры данных [1]. Практика программирования указывает на то, что программисты решают задачу проектирования структур данных в значительной степени интуитивно, на основе опыта либо общих соображений.

Целью работы является разработка методологии структурной адаптации данных, а также создание соответствующего инструментального обеспечения. В ней рассматриваются возможности автоматизированного проектирования структур данных на физическом уровне с улучшенной временной эффективностью.

Учитывая это необходимо решить следующие задачи: разработать метод синтеза адаптированных в процессе эксплуатации структур данных с улучшенными временными характеристиками, совершенствовать технологию их проектирования, реализации и эксплуатации. Решение этих задач подразумевает необходимость разработки показателей временных характеристик структур данных, а также определиться, что считать временем обработки структур данных.

Жизненный цикл структур данных

Жизненный цикл структур данных неразрывно связан с жизненным циклом программных систем [2], составной частью которых они являются.

Рассмотрим традиционные этапы жизненного цикла структур данных и предлагаемые модификации с целью улучшения их эксплуатационных характеристик.

Этап анализа данных можно отождествить с проектированием данных на абстрактном уровне [3, 4]. На этом уровне определяют физическую сущность данных (семантику), связи между элементами на уровне представления пользователя и операции над данными, абстрагируясь от структур данных и их реализаций.

Результатом проектирования являются спецификации данных. Порядок разработки регламентированных спецификаций и анализа абстрактных данных представлен в [3]. Абстрактные спецификации учитывают только внешние аспекты представления данных.

Этап проектирования структур данных может включать проектирование логической структуры данных. Результатом проектирования является разработанная и обоснованная логическая организация данных (концептуальная модель). Проектирование выполняется на основе, выработанных на этом этапе, требований по эффективности [3], безопасности, ограничениям [5] и др. Логическая организация данных должна органично сочетаться с алгоритмами и способствовать упрощению процесса проектирования и разработки программных средств.

Методология проектирования данных на различных носителях и для различных способов использования существенно различается. Проектирование данных в оперативной памяти (ОП) основывается на выборе из

базовых структур данных, либо конструировании других композиционных структур на основе базовых. Базовыми являются хорошо известные массивы, списки, деревья и т. п., множества на основе хеш-функций и др., которые широко представлены и исследованы в [4, 6–9]. Конструирование выполняется путем следования или включения базовых структур друг в друга.

Этап реализации структур данных [10] предусматривает представление концептуальной модели средствами и в терминах среды разработки. Для данных в ОП это среда программирования [10], структуры данных и операции над ними определяют средствами языка программирования с учетом его возможностей и ограничений.

На *этапе эксплуатации или сопровождения* программных систем проявляются эксплуатационные характеристики структур данных. Последние определяются представлением структур данных на физическом уровне [5, 10], т. е. порядком размещение данных на носителях информации и организацией связей между элементами структур. Представление структур данных на физическом уровне в ОП для динамических данных определяется порядком выделения и утилизации памяти, для массивов – порядком размещения элементов (по строкам или столбцам).

Реализация структур данных в ОП, в большинстве случаев, предопределена средствами разработки программных систем, в частности, трансляторами. И лишь иногда предъявленные повышенные требования к эффективности вынуждают на этапе проектирования разрабатывать проект структур данных на физических носителях.

Результатом проектирования является схема размещения данных на носителях, включая вспомогательные данные, такие как адреса, индексы и т. п., а также порядок доступа к элементам данных.

Каждый этап жизненного цикла подразумевает проектирование соответствующих операций обработки данных. На абстрактом уровне определены все операции над данными, которые представлены в виде спецификаций соответствующих процедур и функций. Например, для абстрактных данных "множество вагонов на станции" такими операциями могут быть поступление вагона на станцию, поступление группы вагонов, определение наличия вагона на станции и т. п.

На этапе проектирования – это операции над структурированными данными. Например, если для "множества вагонов на станции" принята структура в виде массива записей, то операциями будут – добавление элемента в массив записей, добавления списка элементов в массив записей, поиск элемента и т. п.

На уровне реализации операции определяются соответствующими методами (процедурами), разработанными средствами среды разработки (языка программирования) на основе их базовых операций, таких как присваивание, сравнение, сложение и т. п. На физическом уровне операции представлены машинными командами процессора, контроллера и т. п.

Предлагаемая в работе методика изменяет жизненный цикл структур данных (табл. 1).

Таблица 1. Особенности предлагаемого подхода в сравнении с традиционным

Этап жизненного цикла	Модели и реализации структур данных	
	Традиционная технология	Предлагаемая технология с адаптацией структур данных на физическом уровне
Анализ	Спецификации структур данных и операций обработки данных	Спецификации структур данных и операций обработки данных
Проектирование	Логические модели на основе выбора базовых структур или конструирования на их основе	Концептуальная модель: элементы данных, логические связи, ограничения
Реализация	Структуры данных и операции над ними терминами средств разработки (языков программирования)	Отсутствует (при наличии специализированных инструментальных средств)
Эксплуатация или сопровождение	Данные на физических носителях	Данные на физических носителях с повышенной временной эффективностью совокупности операций их обработки

Временная эффективность структур данных

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 работах [1, 11] показано, что время операции доступа 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.

Уточним, что понимается под операциями обработки данных. В соответствии с современными принципами программирования обработка данных выполняется на основе атомарных или примитивных операций обработки. Все примитивные операции обработки структур данных можно классифицировать как:

– операции не связанные с изменением структуры и значений ее элементов. Такие операции должны быть ограничены исключительно выборкой или поиском значений и исключать другую функциональность, такую как, например, определение средних значений по выборке и т. п.;

– операции связанные с преобразованием структуры данных в том числе:
– изменение логической структуры (например, замена массива списком, уравновешивание дерева);
– добавление/удаление элемента (ов);
– операции связанные с изменением значений ее элементов без изменения структуры. Такие операции должны быть ограничены исключительно поиском (при необходимости) позиции и заменой значения и исключать преобразование данных;
– операции связанные с изменением значений ее элементов и связанной с этим изменением структуры. Например, добавление элемента в упорядоченный список.

Использование структур данных некоторым конкретным приложением можно представить в виде алгоритма (нотация [12]):

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

где B_i – алгоритмы обработки исследуемой структуры данных (согласно вышеприведенной классификацией), C_i – алгоритмы преобразования данных и обработки других данных и структур, X и Y – их области определения и значения, соответственно. Часть алгоритмов C_i могут быть пустыми.

Тогда временную эффективность структуры данных можно определить на основе временной эффективности алгоритмов вида:

$$A\,|_X^Y = \prod_i B_i\,|_{X_i}^{Y_i}. \qquad (1)$$

Для сравнения временной эффективности двух алгоритмов можно воспользоваться показателем степени превосходства одного (*i*-го) алгоритма над другим (*j*-тым) на ограниченных подмножествах $\overline{\Omega} = ((\overline{V} \subseteq V^*) \times (\overline{U} \subseteq U^*) \times (\overline{X} \subseteq X^*) \times (\overline{\Psi} \subseteq \Psi^*) \times (\overline{\Re} \subseteq \Re^*))$ [13]:

$$SUP_{ij}\left|\overline{V},\overline{U},\overline{X},\overline{\Psi},\overline{\Re} = \frac{1}{N}\sum_{v\in\overline{V}}\sum_{u\in\overline{U}}\sum_{x\in\overline{X}}\sum_{\psi\in\overline{\Psi}}\sum_{r\in\overline{\Re}}\frac{t_j(v,u,x,\psi,r) - t_i(v,u,x,\psi,r)}{\max(t_i(v,u,x,\psi,r), t_j(v,u,x,\psi,r))},\right.$$

где V^* – множество возможных значений объема данных, U^* – множество возможных типов данных, X^* – множество возможных значений данных, Ψ^* – множество возможных программных сред, $\Re^*$ – множество архитектур ЭВМ, на которых возможна реализация алгоритмов.

С учетом размещения и эксплуатации структуры данных в конкретной программно-аппаратной среде и фиксированных типов данных ее элементов:

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

Воспользуемся предложенным там же [13] методом оценки этого показателя, самой оценкой и оценкой ее доверительного интервала. S-оценка степени превосходства *i*-го алгоритма над *j*-м на основании $\overline{N}$ выполнений алгоритмов в области $\overline{\Omega}$:

$$S_{ij}\left|\overline{V},\overline{X} = \frac{1}{\overline{N}}\sum_{k=1}^{\overline{N}}\frac{\bar{t}_{jk}(v,x) - \bar{t}_{ik}(v,x,)}{\max(\bar{t}_{ik}(v,x), \bar{t}_{jk}(v,x))}\cdot 100\ \%,\right. \qquad (2)$$

где $\bar{t}_{ik}$ – время выполнения *i*-го алгоритма при *k*-й реализации в области $\overline{\Omega}$.

Аналогично, показатель области превосходства [13] *i*-го алгоритма над *j*-м:

$$R_{ij}\left|\overline{V},\overline{X} = \frac{1}{\overline{N}}\sum_{k=1}^{\overline{N}} sign(\bar{t}_{jk}(v,x) - \bar{t}_{ik}(v,x,))\cdot 100\%,\ \ sign(a) = \begin{cases}1, \text{ если } a > 0\\ 0, \text{ если } a \le 0\end{cases}.\right. \qquad (3)$$

Оценка степени превосходства и области превосходства *i*-й структуры данных над *j*-й по критерию временной эффективности выполняется в области $\Omega^* == ((\overline{V} \subseteq V^*) \times (\overline{X} \subseteq X^*) \times (\overline{A} \subseteq A^*))$, в которой различие объема и значений данных определяется различным состоянием структуры данных перед и в процессе выполнения различных алгоритмов $A\,|_X^Y$.

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 $A\,|_X^Y$. И здесь возможны варианты.

Первый подход заключается в следующем. Пусть заранее известно количественное соотношение операций обработки структур данных, заданное вектором:

$$\mathbf{N}^T = [n_1, n_2, \ldots, n_m], \tag{4}$$

где n_i – ориентировочное количество выполнений i -ой базовой операции обработки структур данных в конкретной среде эксплуатации, m – количество разнотипных операций обработки данных. Тогда $A|_X^Y$ можно представить как

$$A|_X^Y = \prod_{p=1}^{m} n_p \cdot B_p|_{X_p}^{Y_p} \tag{5}$$

и, исходя из (2) и (4), $\bar{t}_{ik}$ можно определить как:

$$\bar{t}_{ik} = \frac{1}{m}\sum_{p=1}^{m} n_p \tilde{t}_{ikp}, \tag{6}$$

где $\tilde{t}_{ikp}$ – время выполнения p -ой базовой операции обработки структур данных операции при k-й реализации при эксплуатации i-й структуры.

При таком подходе необходимо выполнить множество замеров времени выполнения базовых алгоритмов обработки структур данных.

Однако следует учитываь, что время $\tilde{t}_{ikp}$ зависит не только от V и X, а и от последовательности выполнения операций $B_p|_{X_p}^{Y_p}$. Например, порядок выполнения операций добавления и удаления элементов в/из стека может быть достаточно существенным. Результат (и время) добавления трех элементов в пустой стек и затем удаление из него двух элементов будет отличным, чем при удалении двух элементов из пустого стека, а затем добавления трех.

Второй и последующий подходы основываются на моделировании алгоритма по (1), что в отмеченном смысле предпочтительнее, чем по (5). Моделируемую последовательность операции обработки данных назовем сценарием работы со структурой данных (или просто сценарием).

При втором подходе предлагается такой способ формирования сценария. Заданы ориентировочные границы количества операций обработки структур данных ($\underline{n}_i$ – нижняя, $\overline{n}_i$ – верхняя граница ожидаемого количества выполнений i -ой базовой операции обработки структур данных):

$$\overline{\mathbf{N}}^T = [(\underline{n}_1, \overline{n}_1), (\underline{n}_2, \overline{n}_2), \ldots, (\underline{n}_m, \overline{n}_m)].$$

Для конкретного сценария случайным образом (применяя равномерный закон распределения) определяется количество повторений каждой базовой операции в пределах заданных границ. Последовательность операций строится случайным образом в соответствии с частотой их использования $n_i / \sum_{k=1}^{m} n_k$. При этом

$$\bar{t}_{ik} = t_{ik}^e - t_{ik}^b, \tag{7}$$

где t_{ik}^b, t_{ik}^e – начальное и конечное время выполнения k-го сценария при обработке i-й структуры.

Другие подходы формирования сценария:

– произвольные сценарии формируются пользователем, с использованием соответствующего интерфейса. В этом случае можно учитывать особенности работы со структурами данных;

– сценарий строиться автоматически при наличии средств отслеживания операций обработки структур данных в процессе их эксплуатации (фиксация истории работы со структурой данных).

Применение этих подходов (как и второго) предполагает оценку временной эффективности структур данных по (2) и (7).

Особенности проектирования логической структуры данных

Концептуальная модель логической структуры данных должна отображать составляющие, их свойства и связи между составляющими.

На рисунке показан пример такой модели. Данное представление основано на известных средствах ER-диаграмм [14] и представляют собой модифицированные ER- диаграммы нагруженные атрибутами – MERA-диаграммы.

В качестве составляющих (обозначены в виде прямоугольников) могут быть элементарные, неделимые неструктурированные данные, а также части структур (подструктуры).

Составляющие структуры определяется одним из отношений MERA- диаграмм (обозначены ромбом):

– упорядоченная последовательность однородных, однотипных элементов или подструктур. Примеры таких структур: массивы, хеш-таблицы;

– неупорядоченная последовательность разнотипных элементов или подструктур, например, записей;

– варьируемый состав элементов или подструктур (записи с вариативной частью).

Имеется возможность задания свойств элементов, подструктур и связей в виде их атрибутов (на рисунке – овалы). Атрибуты могут накладывать ограничения на значения элементов, их количество, связи между значениями различных элементов, вычислимые значения элементов и т. п.

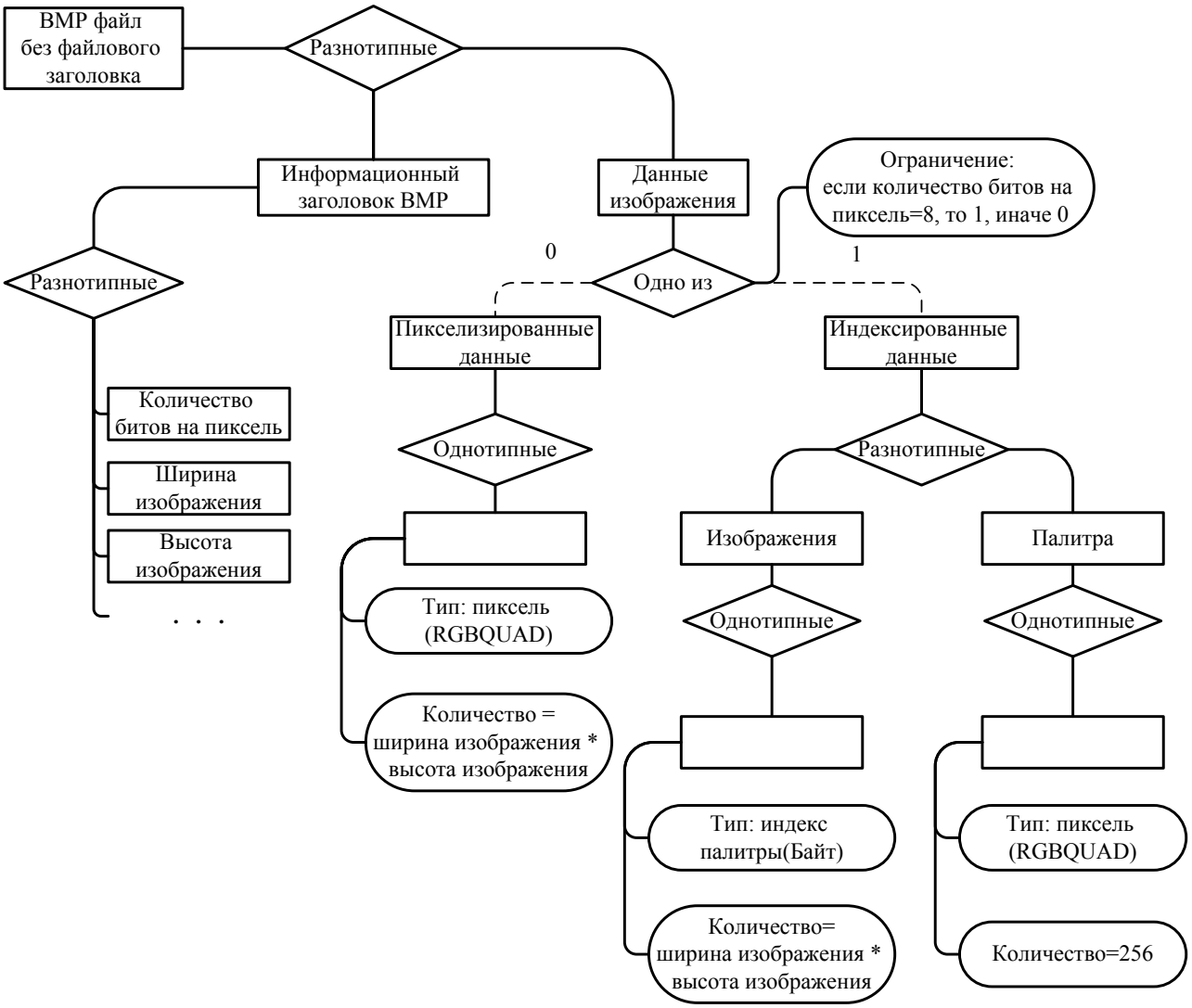

Рисунок. Фрагмент концептуальной модели логической структуры файла формата BMP

Таким образом, концептуальная модель отображает все особенности структуры данных, ее составляющих и связей между ними, не привязываясь в то же время к порядку обработки данных и к традиционным примитивным логическим структурам данных.

Возможности размещения структур данных в оперативной памяти

Любая логическая структура может быть преобразована в разнообразные физические представления (физические структуры). Физическая структура сохраняет семантические связи между элементами логической структуры и добавляет связи связанные с размещением элементов на физическом носителе.

Возможности вариативности физических структур предопределяется следующим: однотипные связи логической структуры могут быть физически реализованы в виде массива (последовательности ячеек). Если массив двух и более размерностей, то вариативным является способ расположения строк и столбцов массива. Кроме того, существуют варианты размещения нескольких массивов с чередованием элементов. Связь между элементами неявно задается последовательной адресацией ячеек. Эта же логическая структура может быть представлена в виде: одно-, двунаправленных списков, деревьев, графов и других динамических структур, в которых элементы связаны явной адресацией. Поиск элементов в первом случае осуществляется вычислением адреса элемента. Во втором случае происходит переход по адресам. Оба эти способа могут комбинироваться. Например, расширяемые массивы без релокации данных. Массив с фиксированным количеством строк и столбцов в случае необходимости увеличения кол-ва элементов может расширяться таким же массивом или

другой размерности в другом месте памяти, при этом строится динамическая последовательность массивов. Второй пример: известные хеш-таблицы с различными методами разрешения коллизий. Одним из атрибутов однотипной связи является количество и размерность, которые могут быть, как фиксированы, так и изменятся во время работы программы.

В случае разнотипных связей данные могут размещаться в совершенно произвольных местах памяти.

Удачное проектирование физической структуры данных в оперативной памяти может в значительной мере повысить временную эффективность в первую очередь по причине эффективного использования кэша памяти. Размещение данных, использующихся почти одновременно, на одной или соседних линейках кэша позволит снизить количество промахов кэша и ускорить обработку структур данных теоретически на один два порядка. Так как операция доступа к данным в оперативной памяти может занимать порядка 100-300 тактов процессора, а в кэш памяти – ноль тактов.

Процесс формирования физической структуры данных на основе логического представления заключается в следующем. На основе простых элементов: целое, вещественное, символ и т.п., применяя различные примитивные структуры: массивы, списки, хеш-таблицы и т. п., формируется множество составных физических структур данных. Все физические структуры данных имеют одинаковый интерфейс (одинаковые методы обработки данных), но различное физическое представления.

Инструментальные средства адаптации

В разработанных инструментальных средствах адаптации структур данных к программно-аппаратной среде эксплуатации предусмотрена следующая функциональность:

- формирование концептуальной модели логической структуры данных;
- построение интерфейсов физических реализаций структур;
- обеспечение формирования и выполнения различных алгоритмов обработки структур данных (1);
- задание и использование данных и/или их особенностей используемых в качестве обучающей выборки;
- выполнение адаптации, т. е. поиск структуры данных, обладающей повышенной временной эффективностью в данной программно-аппаратной среде эксплуатации.

Для формирования концептуальной модели логической структуры данных была использована freeware программное обеспечение Dia [15]. Для подготовки модели использованы стандартные функциональные возможности редактора диаграмм со следующими ограничениями:

- MERA диаграмма должна состоять только из объектов библиотеки ER пакета Dia;
- MERA диаграмма должна содержать только один главный элемент, все остальные элементы должны быть подчиненными ему;
- подчинение объектов обозначается с помощью отношений;
- отношения должны идентифицироваться как «разнотипные» или «однотипные»;
- количество подчиненных элементов для отношения «разнотипные» может быть не ограничено;
- количество подчиненных элементов для отношения «однотипные» должно быть равным одному;
- главный элемент отношения задается с помощью смены типа участия на «общий»;
- для создания поля в отношении «разнотипные», нужно создать сущность, имя которой будет отвечать имени поля. К этой сущности нужно добавить атрибут типа поля. Атрибут должен иметь имя «Тип: <тип>», где тип – один из следующих: dword, byte, string, float.

Для использования адаптированных структур данных из произвольной программы пользователя необходимо создание соответствующего интерфейса. Для доступа к структурам данных и их подструктурам применяются автоматически формируемые классы. Для каждой сущности из концептуальной модели строится свой класс, имя класса совпадает с именем сущности. Имена и параметры методов классов для однотипных данных совпадают с соответствующими методами класса ICollection из .NET. А для разнотипных данных одинаковый фиксированный интерфейс из двух методов для каждого поля: get_<имя_поля> и set_<имя_поля>.

Для построения интерфейсов, формирования физических структур данных и выполнения методов физической реализации используется рефлексия [16] и программные шаблоны. Рефлексия – способ метапрограммирования, при котором программа знает свою структуру и имеет возможность изменить ее. Языки, основанные на промежуточном коде, поддерживают работу с рефлексией. Программный шаблон – часть метапрограммы, которая создает программный код, для реализации той или иной примитивной структуры данных.

Реализация шаблонов осуществляется от обратного в несколько этапов:

- разрабатывается класс, который обеспечивает результат работы программного шаблона с одной из физических реализаций структур данных;
- компилируется библиотека, которая содержит этот класс. Затем библиотека дизассемблируется, чтобы получить IL-код;
- полученный код затем используется в реализации программного шаблона с помощью рефлексии.

Созданный с помощью программных шаблонов код впоследствии может быть сохранен в виде .NET библиотеки и использоваться в дальнейшей разработке сторонних приложений.

Адаптация предполагает поиск предпочтительной по временной эффективности структуры данных. Применяется метод случайного поиска на основе генетического алгоритма. [17]. Пользовательский интерфейс программы позволяет изменять основные параметры генетического алгоритма, что при наличии опыта у поль-

зователя может способствовать сокращению временных затрат на адаптацию. Установленные на предварительных экспериментах параметры генетического алгоритма показывают достаточно высокую его временную эффективность.

Реализация инструментальных средств выполнена на C#, в виду его возможности поддержки рефлексии.

Экспериментальные исследования

Целью эксперимента являлась апробация разработанных инструментальных средств и предварительная оценка возможностей адаптации. Выполнялась адаптация физической структуры данных в ОП из BMP-файлов.

Экспериментальная база состояла с технической стороны из трех компьютеров, основные характеристики которых приведены в табл. 2. Информационная база состояла из 3037 файлов в формате BMP общим объемом 2,1 Гб размером от 104 байт до 5,7 Мб.

Таблица 2. Технические характеристики компьютеров

ЭВМ	Процессор	Кэш L1 кода / L1 данных / L2, Кб	Тактовая частота/частота системной шины / частота памяти, МГц	Время доступа к ОП: чтение / запись, Мб/с	Операционная система
C_1	Intel Core 2 Duo E4600	32/32/2048	2 400 (12 * 200) / 800 / 400	5350 / 1962	Windows 7 Ultimate Prof
C_2	Intel Celeron G530	64/64/512	2 400 (24 * 100) / 100 / 532	6670 / 3220	Windows XP Ultimate Prof
C_3	AMD Athlon 64 × 2 3800 Dual Core	128/128/1024	2 000 (10 * 200) / 200 / 400	5400 / 1860	Windows XP Prof

Согласно концептуальной модели было сгенерировано шесть классов с 5, 10, 4, 6, 6 и 4 методами.

Различные физические реализации структур данных формировались по хромосомам генетического алгоритма. Сценарий обработки структур данных формировался случайным образом согласно заданным ограничениям количества операций обработки. Были установлены следующие ограничения: для методов добавить и удалить пиксель, определить текущее количество пикселей, найти пиксель в изображении класса «данные изображения» $(\underline{n}_i, \overline{n}_i) = (1000, 1000)$, для всех остальных методов $(\underline{n}_i, \overline{n}_i) = (0, 0)$. Сценарии выполнялись для случайно выбранного файла из информационной базы. Последовательность выполнения эксперимента такова:

- формирование сценария случайным образом;
- выбор файла из информационной базы;
- адаптация физической реализации логической структуры данных;
- формирование контрольных структур данных;
- измерение времени выполнения сценария на адаптированной и контрольных структурах данных;
- на основе полученных измерений вычислялись SR- оценки показателей временной эффективности.

В качестве контрольных использовались такие структуры: $STRUCT_1$ – размещение данных в ОП как в файле, $STRUCT_2$ – размещение самого изображения как в файле, $STRUCT_3$ – структура на основе хеш-таблицы (функции хеширования) и $STRUCT_4$ – случайным образом сформированная структура.

Выполнено 64 реализации эксперимента (адаптаций).

Оценка эффективности адаптированной структуры данных выполнялась по SR- показателям (2) и (3). Найденные показатели и доверительные интервалы (с коэффициентом доверия 0,95) приведены в табл. 3.

Таблица 3. Сравнительные оценки временной эффективности адаптированной структуры данных

ЭВМ	$STRUCT_1$		$STRUCT_2$		$STRUCT_3$		$STRUCT_4$	
	S- оценка	R- оценка	S- оценка	R- оценка	S- оценка	R- оценка	S- оценка	R- оценка
C_1	83^{+16}_{-44}	100	83^{+16}_{-44}	100	5^{+19}_{-0}	100	67^{+21}_{-21}	100
C_2	89^{+11}_{-16}	100	89^{+11}_{-16}	100	1^{+1}_{-0}	100	67^{+33}_{-43}	100
C_3	93^{+6}_{-11}	100	93^{+6}_{-11}	100	3^{+9}_{-0}	100	80^{+20}_{-64}	100
Все	88^{+11}_{-12}	100	88^{+11}_{-12}	100	3^{+6}_{-0}	100	71^{+14}_{-14}	100

Как и следовало ожидать, адаптированная структура данных по временной эффективности превосходит все контрольные и во всей области исследования. Показательной является степень превосходства на уровне 90 %, что соответствует разнице во времени выполнения сценария примерно в 10 раз.

Выводы

Выполненная работа позволяет повышать временную эффективность программных систем, эксплуатируемых в гетерогенных компьютерных сетях, если предполагается обработка больших объемов данных и выдвигаются повышенные требования к временным характеристикам. Однако разработанные инструментальные средства пока ограничены в функциональности и ориентированы на исследования, а не производство ПО.

Проект, связанный с автоматическим повышением эффективности структур данных, только открывает определенные перспективы. Можно отметить направления дальнейших исследований:

- оценка возможностей повышения временной эффективности структур данных при адаптации в различных программно-аппаратных средах;
- разграничение возможностей адаптивного синтеза структур данных с необходимостью «ручного» проектирования структур данных на физических носителях;
- выявление предпосылок проведения адаптации, когда временные затраты на адаптацию будут меньше разницы во времени выполнения адаптированного и неадаптированного алгоритмов за определенный срок эксплуатации;
- изучение возможностей повышения эффективности адаптации путем совершенствования (а возможно и адаптации) генетического алгоритма.

1. *Шинкаренко В. И.* Экспериментальные исследования алгоритмов в программно-аппаратных средах : монография. – Днепропетровск: Изд-во Днепропетр. нац. ун-та ж.-д. трансп. им. акад. В. Лазаряна, 2009. – 279 с.
2. *Бабенко Л.П., Лавріщева К.М.* Основи програмної інженерії: Навч. посіб. – К.: Т-во "Знання", КОО, 2001. – 269 с.
3. *Лисков Б., Гатэг Дж.* Использование абстракций и спецификаций при разработке программ. – М.: Мир, 1989. – 424 с.
4. *Ахо А.В., Хопкрофт Дж., Ульман Дж.* Структуры данных и алгоритмы. – М.: Изд. дом «Вильямс», 2001. – 384 с.
5. *Конноли Т. Бегг К., Страчан А.* Базы данных: проектирование, реализация и сопровождение. Теория и практика. – М.: Издательский дом "Вильямс", 2000. – 1120 с.
6. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. – М.: МЦНМО, 2001. – 960 с.
7. *Седжвик Р.* Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск/Алгоритмы на графах. – СПб.: ООО "ДиасофтЮП", 2003. – 1136 с.
8. *Кнут Д.* Искусство программирования, том 1. Основные алгоритмы. – [3-е изд.]. – М.: Издательский дом "Вильямс", 2000. – 720 с.
9. *Вирт Н.* Алгоритмы + структуры данных = программа. – М.: Мир, 1985. – 406 с.
10. *Зиглер К.* Методы проектирования программных систем. – М.: Мир, 1985. – 328 с.
11. *Шинкаренко В.И.* Временная оценка операций обработки структурированных данных с учетом конвейеризации и кэширования // Проблеми програмування. –2006 – № 2-3. – С. 43–52.
12. *Шинкаренко В.И., Ильман В.М., Скалозуб В.В.* Структурные модели алгоритмов в задачах прикладного программирования Часть I. Формальные алгоритмические структуры // Кибернетика и системный анализ. – 2009 – № 3. – С. 3–14.
13. *Шинкаренко В.И.* Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов // Проблемы программирования. – 2001. – № 3-4. – С. 31–39.
14. *Чен П.* The Entity-Relationship Model – Toward a Unified View of Data (англ.) // ACM Transactions on Database Systems (TODS) . –Нью-Йорк: ACM, 1976. –V. 1. – P. 9–36.
15. *Dia – GNOME Live!* – http://live.gnome.org/Dia.
16. *Шилдт Г.* C# 4.0: полное руководство.– М.: Вильямс, 2010. – 1056 с.
17. *Гладков Л.А., Курейчик В.В., Курейчик В.М.* Генетические алгоритмы: Учебное пособие. – 2-е изд. – М: Физматлит, 2006. – 320 с.