УДК 510.5

ВРЕМЕННАЯ ОЦЕНКА ОПЕРАЦИЙ ОБРАБОТКИ СТРУКТУРИРОВАННЫХ ДАННЫХ С УЧЕТОМ КОНВЕЙЕРИЗАЦИИ И КЭШИРОВАНИЯ

В.И. Шинкаренко

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

Рассмотрены вопросы поэтапного проектирования данных. Исследованы операции работы со структурированными данными. Предложена методика оценки времени выполнения для операций языков программирования и машинных команд. Предложена также методика оценки операций доступа к позиции и к элементам структурированных данных. Указанные методики учитывают конвейеризацию команд и кэширование данных. Все методики являются статистическими и учитывают особенности конфигурации ПЭВМ. Проведение исследований операций на основе предложенных методик позволят более обоснованно подходить к проектированию данных в условиях повышенных требований к временной эффективности программ.

This report presents multistage designing of data. The operations with the structured data are investigated. The technique of an estimation time's performance for operations of the programming languages and machine commands is offered. A technique of measurement of operations such as access to a position and to elements for the structured data also is offered. The specified techniques take into account of piping command and caching of the data. All techniques are statistical and use a configuration of computer. Realization of researches of operations on the basis of the offered techniques will allow more soundly to solve the tasks of designing.

Введение

Перефразируя известную формулу Н. Вирта, можно сказать "эффективные программы = эффективные алгоритмы + эффективные структуры данных". Под эффективностью алгоритмов, будем понимать временную эффективность, связанную со временем выполнения алгоритмов определенными исполнительными механизмами. Под временной эффективностью структур данных – временную эффективность операций обработки данных с учетом частоты их выполнения.

Вопросам разработки эффективных алгоритмов и их анализу [1-5], а также проектированию данных на внешних носителях (баз данных [6] и т.п.) уделяется достаточно большое внимание. Вопросы проектирования данных в оперативной памяти (ОП) как теоретически, так и практически исследованы значительно меньше. И хотя временная эффективность программ существенно зависит от удачного выбора структур данных, практика программирования указывает на то, что программисты решают эту задачу в значительной степени интуитивно, на основе опыта либо общих соображений.

В рамках модели СММ управление качеством подразумевает достижение в программном продукте требуемых показателей качества [7]. Среди них временные показатели программных средств представляются достаточно важными.

Разный уровень требований к эффективности программ предопределяет различные подходы к выбору структур данных. Если разрабатываемые части программ не критичны по времени либо требования к временной эффективности программных средств не выдвигаются, то существующие подходы вполне оправданы. Повышенные требования вынуждают к выполнению длительных и дорогостоящих исследований при разработке программ.

Одним из инструментов улучшения временных характеристик программ является обоснованное, а в лучшем случае оптимальное проектирование структур данных.

1. Проектирование данных

Можно выделить 4 уровня (этапа) проектирования данных.

Абстрактный уровень [8,9], где определяют физическую сущность данных, связи между элементами на уровне представления пользователя и операции над данными, абстрагируясь от структур данных и их реализации.

Результатом проектирования являются спецификации данных. Порядок разработки регламентированных спецификаций и анализа абстрактных данных приведен в [8]. Его можно соотнести с внешним уровнем проектирования баз данных (БД) [6], так как абстрактный уровень учитывает только внешние аспекты представления данных.

Проектный уровень [10] (концептуальный уровень [6]). Проектирование выполняется на основе выработанных на этом этапе требований по эффективности [8], безопасности, ограничениям [6] и др. Логическая организация данных должна органично сочетаться с алгоритмами и способствовать упрощению процесса проектирования и разработки программных средств.

Результатом проектирования является разработанная и обоснованная логическая организация данных (концептуальная модель).

Методология проектирования данных на различных носителях и для различных способов использования существенно различается.

Так, для реляционных БД применяют известные методы нормализации [6].

Проектирование данных в ОП основывается на выборе из базовых структур данных, таких, как массивы, различные графовые структуры динамической организации (списки, деревья и т.п.), множества на основе хеш-функций и др., которые широко представлены и исследованы в [1-4,9], либо конструировании других композиционных структур на основе базовых, в основном путем следования или включения базовых друг в друга.

Уровень реализации [10]. Логическую организацию данных реализуют 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. В случае БД средой разработки являются средства проектирования (в основном, визуальные) СУБД.

Физический уровень [6,10]. Проектируется размещение данных на носителях информации.

В ОП это в первую очередь касается динамических данных, их размещения, утилизации и т.п. Для массивов может быть существенным порядок размещения элементов (по строкам или столбцам). В БД – это вопросы размещения записей на магнитном диске (МД), порядок размещения таблиц индексов и т.п.

Результатом проектирования является схема размещения данных на носителях, включая вспомогательные данные, такие, как адреса, индексы и т.п.

Естественно, что при решении конкретных задач проектирования программных средств (ПС) не все этапы и не для всех данных обязательны. Так, при проектировании БД физический уровень проектирования, как правило, уже выполнен и реализован средствами СУБД.

Каждый уровень проектирования подразумевает и соответствующий уровень 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 проектном уровне – 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 базовыми операциями среды разработки (языка программирования) такими, как присваивание, сравнение, сложение и т.п. На физическом уровне операции представлены машинными командами процессора, контроллера и т.п.

Так как эффективность структур данных определяется эффективностью операций, существенным является вопрос методологии оценки эффективности операций на всех уровнях проектирования.

2. Оценка эффективности структур данных на проектном уровне

Есть несколько подходов для оценки временных характеристик структур данных.

Вероятностный подход заключается в том, что используются учетные стоимости операций [1,9] с данными, которые аналогичны вычислительной сложности алгоритмов и определяются теми же методами, что и последние. Учетная стоимость определяет порядок зависимости времени выполнения операции от объема данных в среднем. Учетную стоимость обозначают с использованием символ *O*, который ввел Поль Бахман в 1894г. [3]. Запись $f(n) = O(g(n))$ означает, что с ростом *n* отношение $f(n)/g(n)$ остается ограниченным некоторой константой *C* [3].

Однако при решении конкретных задач подобные оценки не всегда применимы. Особенно если объем данных относительно небольшой, но они часто используются.

При статистическом подходе операции можно рассматривать как алгоритмы обработки данных и применять к ним те же статистические S-R-L-оценки [11], что и к алгоритмам, но в более узкой области.

Третий подход к оценке временной эффективности структур данных на проектном уровне базируется на необходимом количестве операций уровня реализации:

проектируются конкурентноспособные структуры данных;

устанавливаются веса для всех операций проектного уровня (либо на основе вероятностных или статистических оценок, либо экспертным методом);

подсчитывается количество требуемых для выполнения операций уровня реализации:

$$N_{ik} = \sum_j v_j n_{jik} \ , \qquad (1)$$

где N_{ik} - оценочное количество *i*-й операции уровня реализации (присваивания, сравнения и т.п.) для *k*-й рассматриваемой структуры данных;

v_j - вес *j*-й операции проектного уровня;

n_{jik} - среднее количество *i*-х операций уровня реализации, необходимых для выполнения *j*-й операции проектного уровня для *k*-й рассматриваемой структуры данных.

На основании построенных количественных оценок и принимается решение о выборе структуры данных.

Для перехода к временным оценкам в (1) желательно учесть время выполнения операций уровня реализации:

$$T_k = \sum_i \sum_j v_j n_{jik} t_i , \tag{2}$$

где T_k - оценка времени работы с *k*-й рассматриваемой структурой данных;

t_i - время выполнения *i*-й операции уровня реализации.

3. Оценка времени выполнения операций уровня реализации

Операции уровня реализации являются базовыми операциями алгоритмов. Наиболее часто применяемые из них - операции присваивания и сравнения.

Рассмотрим единичную операцию сравнения двух элементов структур данных

$$x(i) > x(j). \tag{3}$$

Команда сравнения процессоров Intel Pentium и его аналогов имеет табличное время выполнения один такт. В условиях конвейерной обработки с латентностью конвейера до 30 команд фактическое время выполнения может быть значительно сокращено. Если значения элементов структуры определяются предыдущими командами конвейера и не готовы для выполнения текущей команды, возможны задержки конвейера. При этом время выполнения операций сравнения может измениться.

Таким образом, оценки количества операций в виде (3) не могут дать оценки операций со структурами ввиду неоднозначности времени выполнения самых примитивных операций на конвейере.

Если алгоритм обработки предусматривает одну операцию сравнения типа (3), то соответствующая программа – 8. Как известно [12], наличие терма $x(i)$ в программе подразумевает выполнение четырех операций: обработки ссылки, доступа к позиции, доступа к значению и приведения типов.

Операция обработки ссылки заключается в идентификации объекта. В конкретном случае следует установить, что это элемент массива. Чаще всего операция обработки ссылки выполняется на этапе трансляции и на время выполнения программы влияния не оказывает. Однако многие современные языки программирования имеют данные, тип которых определяется на стадии выполнения. Так, Borland Delphi имеет тип Variant, данные которого могут быть любого типа. При этом идентификация объекта производится во время выполнения программы. Операция обработки ссылки и приведения типа при этом может состоять из многих сотен машинных команд, поэтому время выполнения операции обработки ссылки и приведения типов на несколько порядков превосходит время самой операции сравнения.

Учитывая, что при проектировании и разработке данных, в случаях критичных по времени выполнения, операции приведения типов и обработки ссылки на этапе выполнения практически исключаются, остановимся на следующих.

Операция доступа к позиции [12,13] заключается в определении (вычислении) адреса данных. Для одномерного массива она определяется как

$$A = A_0 + l_э(i - i_0) = A_0 - l_э i_0 + l_э i = \overline{A} + l_э i , \tag{4}$$

где $A_0, \overline{A}$ - начальный и приведенный адрес массива; $l_э$ - размер элементов массива; i, i_0 - текущий и начальный индексы.

В выражении (4) присутствуют операции сложения и умножения (при умножении на число кратное степени двойки, умножение заменяется сдвигом либо выполняется в одной команде процессора с использованием индексного регистра).

В Borland Delphi операция доступа к позиции для элементов массива Integer строится следующим образом (ассемблерная нотация):

mov eax,[ebp]

*mov eax,[eax*4+* $\overline{A}$ *].*

Время выполнения операций уровня реализации определяется следующим образом:

$$t_{onII} = t_к + t_{oc} + t_{∂n} + t_{∂∂} , \tag{5}$$

где t_{onII} - время выполнения операции языка программирования; $t_к$ - время выполнения соответствующей (щих) команды процессора; t_{oc} - время выполнения операции обработки ссылки; $t_{∂n}$ - время доступа к позиции операндов; $t_{∂∂}$ - время доступа к данным операндов.

Например, время выполнения операции присваивания элементов массива в Delphi $x[i] := x[j]$ складывается из времени выполнения команды (одной или двух) пересылки mov, нулевого времени обработки ссы-

лок, двух операций доступа к позиции (согласно (4)) и одной операции доступа к данным (операнда справа) и, возможно, операции приведения типа.

4. Оценка времени выполнения команд процессора

С учетом конвейерной обработки команд время выполнения одной команды процессора является величиной нечеткой, что обусловлено особенностями аппаратного исполнительного механизма, в частности конвейерной обработкой команд и кэшированием данных и команд.

На конвейере одновременно выполняется несколько команд в зависимости от его длины (Pentium II – 14, Pentium III – 20 [14]). Говорят о латентности команды – времени от начала выполнения команды на конвейере до ее выхода с конвейера и о производительности конвейера – количестве выполненных команд за единицу времени [14].

Под временем выполнения команды процессором будем понимать величину, обратную производительности конвейера при выполнении потока таких команд с условием, что данные находятся в кэше:

$$t_к = 1/M\ , \qquad (6)$$

где M – количество выполненных конвейером команд за единицу времени.

В [14] выделены аппаратные средства измерения времени: системный таймер (точность до 0,84 мкс), часы реального времени (0,98мс) и регистр-счетчик времени (1 такт процессора) и показаны проблемы, связанные с их использованием в качестве измерительных инструментов для времени выполнения команды.

Казалось бы, что наиболее подходящим для измерения времени выполнения команды является регистр-счетчик времени. Однако команда доступа к нему является неупорядоченной и может выполниться раньше измеряемой команды и при ее упорядочении с помощью других команд позволяет измерить только латентность [14]. Кроме того, следует учитывать, что между двумя измерениями может быть переключение процессов диспетчером операционной системы.

Предлагается следующая процедура измерения.

В качестве измерительных средств используются часы реального времени. Для снижения случайных влияний других процессов количество процессов в системе сводится к минимуму, удаляются процессы со спонтанным запуском. Длительное наблюдение за загрузкой процессора показывает, что простой процессора ~99%, т.е. следует ожидать, что ошибки измерений за счет случайных переключений на другие процессы будет порядка 1%.

Измеряется время выполнения цикла (рис.1). Достаточно большое количество повторений цикла ($n_ц$ =5, …, 100 млн) обеспечивает высокую точность измерений и практически снимает влияние измерительной систем. Измеряется время выполнения нескольких таких циклов с постепенным увеличением количества вложенных команд (*n+1, n+2* …).

```
t_н = time                    замер времени начала цикла
for i:=1 to n_ц do
  begin
  asm
      add ax,1
        …                     очистка конвейера
      add ax,1                (число команд больше латентности
конвейера)

      измеряемая команда
          …                   n_k повторений команды
(операций)
```

Рис.1. Цикл для замера времени выполнения команды процессора

Для снижения случайного влияния измерительной системы, программного и аппаратного непостоянства проводилось несколько параллельных измерений ($n_{пар}$). Из них выбиралось минимальное, так как помехи могут лишь увеличить время выполнения цикла, но никак не уменьшить. Опыт показал, что уже при $n_{пар}$ =9 результаты измерений практически абсолютно совпадали.

Измерения проводились при n_k =30…80.

Казалось бы, разница времени выполнения цикла с n+1 и n вложенных команд и есть временем выполнения команды i раз. Однако это не так. Посмотрим на примерные графики зависимости времени выполнения цикла от количества вложенных команд (рис. 2.).

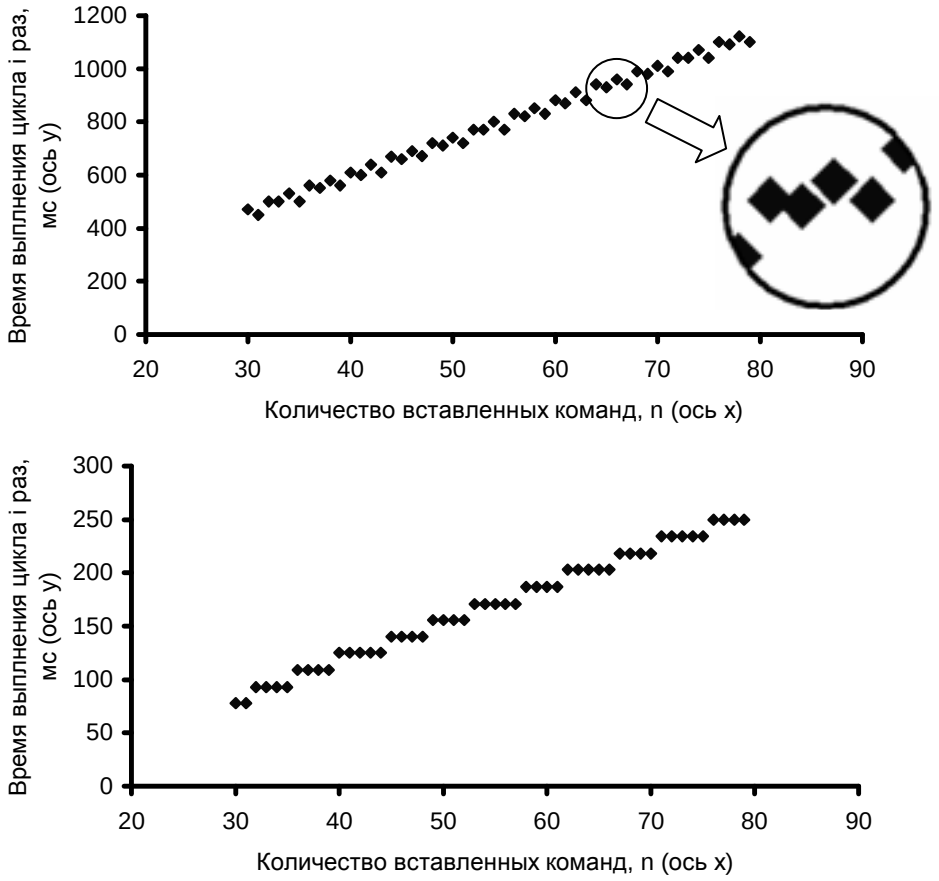

Рис.2. Примеры зависимости времени выполнения цикла от количества вставленных команд

Как видим, добавление команды может даже уменьшить время выполнения цикла. Анализ показывает, что это связано с тем, что команды управления циклом по-разному ложатся на линейку кэша. Если части команды ложатся на участок памяти, попадающий на разные линейки кэша (т.е. команда покрывает адрес кратный 32 байтам), то время выполнения цикла значительно увеличивается. Складывается ситуация, когда увеличение времени выполнения цикла полностью компенсируется и даже уменьшается за счет того, что команды управления циклом смещаются с границы линеек кэша.

Следует отметить, что приведенные зависимости носят устойчивый характер. Повторное выполнение эксперимента дает практически полностью совпадающие результаты измерений.

Угол наклона зависимости (см. рис.2) определяет время выполнения команды $t_к$:

$$t_к = \frac{\Delta t}{\Delta n} \cdot \frac{1}{i}. \tag{7}$$

Но для каких точек взять Δt и Δn?

Заметим, что повернув систему координат на искомый угол, получим почти периодическую функцию (с небольшими помехами измерительной системы). Учитывая этот факт и то, что зависимость является дискретной, предложен следующий порядок вычисления $t_к$.

Для простоты изложения перейдем к обозначениям системы координат – *x, y*.

Считаем, что зависимость периодически сдвинута по *x* и *y* и каждой точке в первом периоде есть соответствующая точка в остальных:

$$\begin{aligned} x_1 &= x + ka \\ y_1 &= y + kb \end{aligned} \quad \text{при} \quad \begin{aligned} x \in [0, T_x], &\quad x_1 \in [kT_x, (k+1)T_x], \\ y \in [0, T_y], &\quad y_1 \in [kT_y, (k+1)T_y], \end{aligned} \tag{8}$$

где T_x, T_y - период функции по оси *x* и *y*.

Неизвестные *a,b,T* ищем методом наименьших квадратов. Необходимо найти минимум следующей функции:

$$F = \sum_{i=1}^{N} (x_i^T - x_i^Э)^2 + (y_i^T - y_i^Э)^2 \to \min \tag{9}$$

или

$$F = \sum_{i=1}^{N} (x_{i+k} - x_i + a)^2 + (y_{i+k} - y_i + b)^2 \to \min . \tag{10}$$

Приравняв нулю частные производные $\frac{\partial F}{\partial a} = 0$ и $\frac{\partial F}{\partial b} = 0$, получим

$$\begin{aligned} \overline{a} &= \frac{1}{k} \sum_{i=1}^{k} (x_{i+k} - x_i), \\ \overline{b} &= \frac{1}{k} \sum_{i=1}^{k} (y_{i+k} - y_i). \end{aligned} \tag{11}$$

Период *к* определяем перебором при минимизации функции *F*:

$$k: \quad F = \min_k \sum_{i=1}^{N} (x_{i+k} - x_i + \overline{a})^2 + (y_{i+k} - y_i + \overline{b})^2 . \tag{12}$$

Время выполнения команды в тактах вычисляем как

$$t_к = \frac{\overline{b}}{\overline{a} \cdot n_ц} \cdot F_n , \tag{13}$$

где F_n - тактовая частота процессора.

Для оценки достоверности полученных результатов определяются доверительные интервалы.

Для периодической функции с периодом *k* значения *y* в точках на расстоянии $\Delta x = k$ должны совпадать. Разница в их значении есть случайная величина, вызванная помехами измерения, т.е.

$$\xi_i = y_{i+k} - (y_i + \overline{b}) . \tag{14}$$

Таких величин будет $N_\xi = \max(n_k) - \min(n_k) - k$.

Сгруппируем их по три и определим средние:

$$\zeta_j = \frac{1}{3} \sum_{1+(j-1)*3}^{3+(j-1)*3} \xi_j . \tag{15}$$

Так как ζ_j - независимые одинаково нормально распределенные случайные величины, то случайная величина, то

$$t = \sqrt{N_\xi / 3} \, \frac{\overline{\zeta} - M[\zeta]}{s}, \tag{16}$$

(где $\overline{\zeta} = \frac{3}{N_\xi} \sum_{k=1}^{N_\xi/3} \zeta_k$ и $s^2 = \frac{3}{N_\xi} \sum_{k=1}^{N_\xi/3} (\zeta_k - \overline{\zeta})^2$ $M[\zeta]$ - математическое ожидание ζ) распределена по закону Стьюдента [15].

Так как математическое ожидание и средние ζ и ξ совпадают, из (16) получаем доверительные интервалы для ξ:

$$M[\xi] = \overline{\zeta} \pm t_{N_\xi/3-1,\beta} \sqrt{s^2 (N_\xi / 3 - 1)} , \tag{17}$$

где $t_{\alpha,\beta}$ - табличное значение распределения Стьюдента с α степенями свободы и коэффициентом доверия β.

Тогда

$$M[\bar{b}] = \bar{b} \pm t_{N_\xi/3-1,\beta}\sqrt{s^2(N_\xi/3-1)}\,. \qquad (18)$$

Согласно (13) определяем доверительный интервал для $M[t_k]$.

В таблице приведены результаты измерения команды сложения *add ax,1* на некоторых ПЭВМ. Здесь же указаны наиболее существенные условия проведения измерений. Технические характеристики определялись с помощью программы AIDA32 [16].

Несмотря на работу 15…20 спящих системных процессов, результаты имеют достаточную для проектирования структур данных точность.

Предложенная методика позволяет также производить измерения времени выполнения операций уровня языка программирования. В таблице даны результаты измерений операций присваивания (без и с задержкой по данным) и операции сравнения Borland Delphi (здесь и ниже примеры даны в терминах языка Паскаль и Ассемблера). Соответствующие операции для измерения приведены на рис. 3. Поскольку языки программирования не позволяют повторять подряд операции сравнения, эта операция моделируется ассемблерными командами так же, как и Delphi. При этом условные переходы не выполнялись.

```
a    a1:=b1          b    a2:=a1          c    mj:  mov  eax[a1]
     a2:=b2               a3:=a2                    cmp  eax[b1]
     a3:=b3               a4:=a3                    jne  mj
      …                    …                         …
```

Рис.3. Последовательности измеряемых операций: a - присваивания без задержки по данным; b - присваивание с задержкой по данным; c - сравнения без задержки по данным

5. Оценка времени выполнения операций доступа к позиции и значению

Приведенная выше методика позволяет оценить и время доступа к позиции для массивов. Измерения последовательностей операций, приведенных на рис. 4, a и b, отличаются лишь тем, что в первом случае доступ осуществляется к простым неструктурированным данным, а во втором – к элементам массива. Разница во времени и определяет время доступа к позиции массива с элементами длиной 4 байта. Отметим, что для ПЭВМ на базе процессора Intel Pentium это время практически нулевое, а на базе процессора AMD оно составляет 2…3 такта.

```
a    a1:=b1          b    a2:=x[j+1]
     a2:=b2               a3:=x[j+2]
     a3:=b3               a4:=x[j+3]
      …                    …
```

Рис.4. Последовательности измеряемых операций: a - присваивания простых неструктурированных данных; b - присваивание элементов массива

Следует ожидать, что время доступа к данным будет сильно отличаться в зависимости от количества промахов в кэше. Поэтому для оценки времени доступа к данным выполнялся несколько другой эксперимент.

В цикле (см. рис. 1) находилась лишь одна измеряемая операция ($n_k = 1$), а именно i:=x[i]. При этом по-разному формировался массив x. В первом случае $x_i = i$. При этом в цикле постоянно шло обращение к одному и тому же элементу массива, который после первых проходов точно попадал в кэш. Можно считать, что первый случай соответствует ситуации, когда структурированные данные постоянно находятся в кэше. Измеренное время выполнения цикла обозначим τ_1.

Во втором случае $x_i = i + 1$. При этом в цикле выполнялась последовательная выборка элементов массива. Если время этого цикла обозначить τ_2, то $\tau_2 - \tau_1$ - минимальное среднее время доступа по всем элементам массива. Заметим, что это время близко к нулю, что объясняется упреждающей дешифровкой команд и упреждающим чтением данных.

В третьем случае $x_{(i-1)\cdot Dn+1} = i \cdot Dn + 1$. При этом просматривались элементы с некоторым шагом (Dn). Изменяя этот шаг в пределах $Dn = 30\ldots80$, определяли максимальное среднее время доступа ко всем элементам массива. Этот случай соответствует ситуации, когда практически всегда требуемый элемент массива отсутствует в кэше.

Результаты выполненных измерений

Таблица

			ПЭВМ							
			1	2	3	4	5	6	7	8
Характеристики вычислительной среды										
Процессор / чипсет системной платы			Intel Pentium IIIE / Intel Solano i815E	Intel Celeron-S / Intel Solano i815E	Intel Pentium 4 /Intel Brookdale i845D	Intel Pentium 4HT/ Intel Springdale-G i865G	Intel Pentium 4A / Intel Brookdale i845E	Intel Pentium 4E / Intel Springdale-G i865G	AMD Duron XP / VIA VT8375 ProSavage DDR KM266	AMD Duron XP / nVIDIA nForce2 Ultra 400
Кэш L1 кода / L1 данных / L2, Кб			16/16/256	16/16/256	12/8/256	12/8/512	12/8/512	12/16/1024	64/64/64	64/64/256
Тактовая частота/частота системной шины / частота памяти, МГц			650 (6.5 x 100) / 100 / 133	994 MHz (10 x 99) / 99 /133	1600 MHz (4 x 400)/133/133	2400 MHz (3 x 800) / 200/200	2400 MHz (4.5 x 533)/133/166	2400 MHz (4.5 x 533)/166/200	1400 (5.25 x 267) / 133 / 200	1666MHz (5 x 333)/ 166/200
Время доступа к ОП:.чтение/запись, Мб/с			515/148	507/120	1449/583	3794/1205	1911/667	3298/1186	1215/438	2205/815
Операционная система			Windows 2000 Prof	Windows XP Prof	Windows XP Prof	Windows XP Prof	Windows XP Prof	Windows XP Profs	Windows XP Prof	Windows XP Prof
Среднее время выполнения операций ± доверительный интервал, такты										
Сложение с задержкой конвейера			1,76±0,02	1,75±0,52	0,51±0,02	0,48±0,15	0,49±0,15	1,02±0,01	1,00±0,02	0,99±0,25
Присваивание простых данных без задержки конвейера			1,38±0,38	1,59±0,51	2,21±0,82	2,21±0,79	2,28±1,04	1,97±0,18	1,05±0,03	1,03±0,04
Присваивание структурированных данных без задержки конвейера			1,33±0,39	2,12±0,04	2,05±0,13	2,20±0,14	2,20±0,10	2,55±0,04	4,00±1,12	4,00±0,03
Присваивание простых данных с задержкой конвейера			3,7±1,15	3,65±1,14	8,71±0,98	8,47±0,83	8,31±0,79	4,87±1,34	3,87±0,61	3,76±0,33
Сравнение простых данных без задержки конвейера			2,01±0,6	2,00±0,59	2,08±0,03	2,08±0,61	2,04±0,62	2,09±0,08	2,01±0,55	2,02±0,61
Среднее время доступа к элементам структур данных ± доверительный интервал, такты										
Массив 4-х байтовых элементов	Время доступа к позиции		-0,05±0,77	0,63±0,55	-0,16±0,95	-0,01±0,93	-0,08±1,14	0,58±0,22	2,95±1,15	2,97±0,07
	Время доступа к данным	min	6,95±1,05	8,94±0,72	1,02±0,56	1,19±1,04	0,95±1,02	1,17±0,95	1,82±0,78	0,84±68
		max	134,5±5,5	198,8±2,4	233,8±5,7	264,0±8,8	273,8±6,3	340,1±22,7	260,2±3,9	258,4±2,5
Динамические структуры с адресом в 4 байта	Время доступа к позиции		$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$	$t_{д\,эл} \cdot (n_{пр} - 1)$
	Время доступа к данным $t_{д\,эл} / n_{пр}$	min	1,00±1,03	18,53±0,48	0,00±0,42	-2,05±2,01	-1,58±2,57	-1,07±1,14	3,9±4,41	-1,53±1,45
		max	134,8±8,1	215,1±2,2	222,2±6,1	240,1±9,4	257,0±6,7	316,1±8,5	266,2±25,4	272,3±5,2

$n_{пр}$ - среднее количество просмотренных элементов динамических данных для доступа к данному.

В четвертом случае $x_i = random(i)$. Массив заполнялся индексами случайным образом, но так, чтобы каждый элемент массива был выбран один раз. В этом случае определяется среднее время доступа к элементам массива при случайном порядке обработки элементов. Пример зависимости среднего времени доступа от размерности массива (для ПЭВМ №8 по таблице) приведен на рис.5.

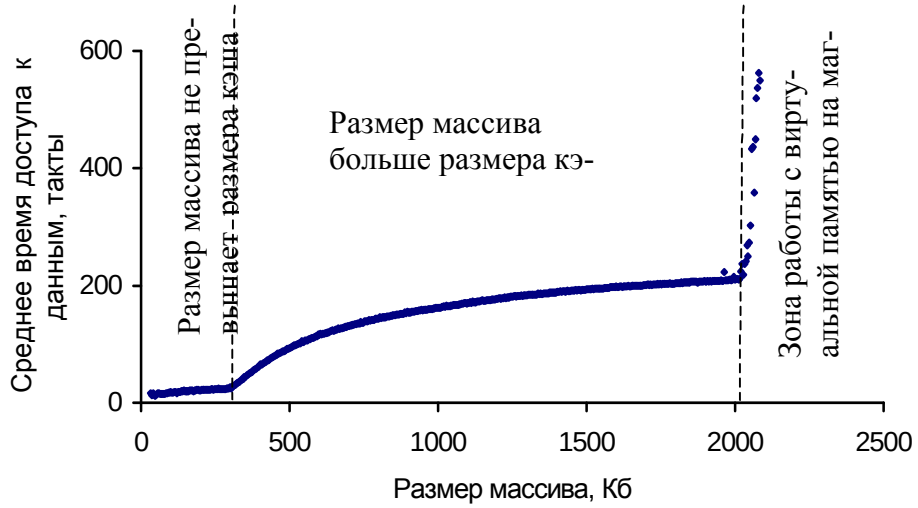

Рис.5. Зависимость среднего времени доступа к элементам массива от его размерности при случайном порядке выбора элементов

Перед каждым измерением кэш очищался интенсивной работой с большим массивом данных.

```
type Tel=integer;
     Prec=^rec;
     rec=record
          x:integer;
          adr:prec;
     end;
   Tarr=array[1..n_ц] of  rec;
   Sarr=^Tarr;
var el:prec; arr:Sarr;
```

Рис.5. Описание структур данных

Как и для массивов, исследование времени доступа к элементам динамических данных проводилось для худшего, среднего и лучшего случаев. Во избежание влияния фрагментации памяти, выделяемой под динамические данные, динамические записи были вложены внутрь динамического массива (рис.5.).

В измеряемый цикл (см. рис.1.) вложена операция el:=el.adr (до цикла адрес в записи ссылается на массив el.adr:=addr(arr)).

Для измерения в лучшем случае, если все элементы в кэше, массив формировался как arr^[i].adr:=Addr(arr^[i]), для измерения минимального среднего времени при просмотре всего списка каждая запись в массиве ссылалась на следующую – arr^[i].adr:=Addr(arr^[i+1]), максимального среднего времени – с шагом Dn - arr^[(i-1)*Dn+1].adr:=Addr(arr^[i*Dn+1]).

В среднем случае запись ссылалась на случайно выбранную другую запись (закольцованный список). Так моделировалась случайным образом фрагментированная память.

Заметим, что для доступа к позиции динамической записи необходимо последовательно просмотреть предыдущие записи списка. Время доступа к позиции динамической записи равно времени доступа к данным предварительно просмотренных записей.

При разработке структур данных следует в какой-то мере ориентироваться на среднее время доступа к элементам массива со случайной выборкой. Однако лучше моделировать последовательность выборки, близкую к работе с проектируемыми данными.

Заключение

Предложены методы измерения времени выполнения операций обработки данных для логического и физического уровней их проектирования. Учитываются особенности архитектуры ЭВМ, такие как конвейеризация команд и кэширование данных.

Современные вычислительные системы основываются на многоуровневой организации памяти:

регистровая память процессора
↓
кэш-память
↓
оперативная память
↓
внешняя память на различных носителях, в том числе и виртуальная на магнитных дисках
↓
память другого компьютера (сервер данных)
↓
распределенная память компьютерной сети.

В связи с этим и многообразием проектных решений по архитектурам и модификациям ЭВМ и сетей, техническим характеристикам носителей данных на всех уровнях задача проектирования структур данных на логическом и физическом уровнях в значительной мере осложняется.

В статье рассмотрен один из этапов проектирования данных, основанный на предложенной методике измерений базовых операций со структурами данных с учетом двухуровневой модели памяти: кэш ↔ ОП.

Как показывают проведенные исследования, существенный вклад во время выполнения базовых операций алгоритмов могут вносить операции доступа к данным. Среднее время операции доступа к данным существенно зависит от порядка обработки данных, насколько близко они выбираются и последовательно обрабатываются. Последнее определяется алгоритмом обработки данных. Таким образом, проектирование структур данных предопределяется алгоритмом (или алгоритмами) их обработки. Одним из путей повышения эффективности программно-аппаратных систем, предназначенных для реализации алгоритмов обработки больших объемов структурированных данных, является комплексный подход к разработке алгоритмов и структур данных с учетом особенностей архитектуры ЭВМ (рис.6).

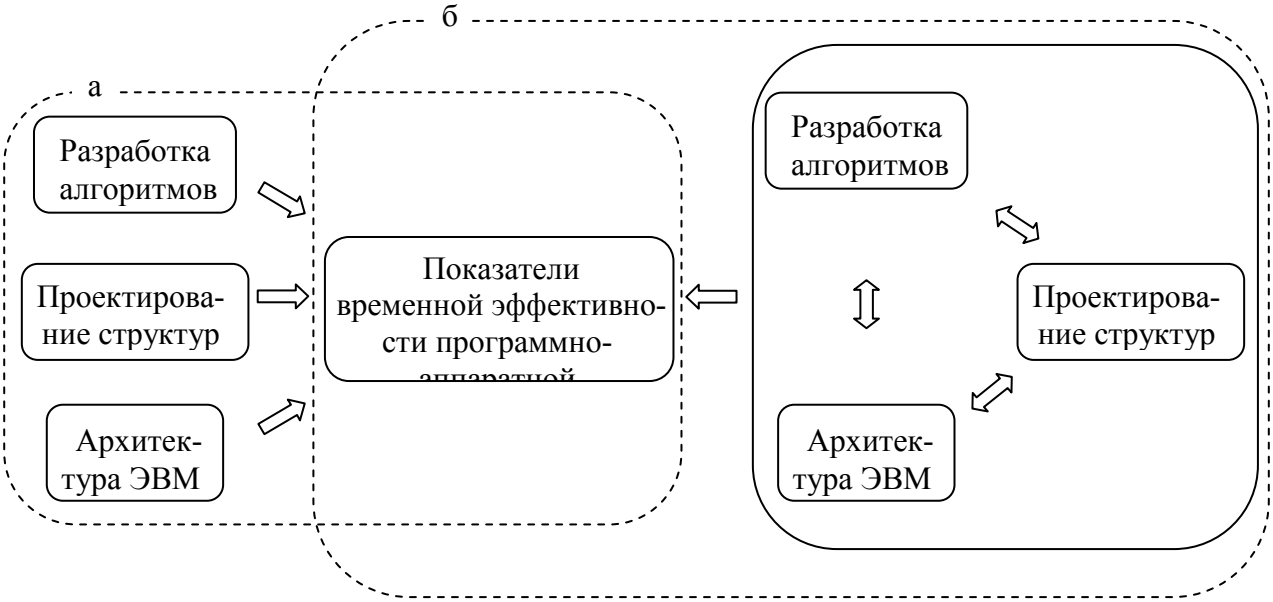

Рис.6. Схема по организации а - раздельной и б - совместной разработки алгоритмов и проектирования структур данных на основе информации о временных характеристиках ЭВМ

Несмотря на несовершенство измерительной системы и многозадачность операционной системы, предлагаемые статистические методы оценки временных характеристик дают устойчивые результаты с достоверностью достаточной для решения задач выбора и проектирования структур данных.

На вполне резонный вопрос:

а нужно ли подстраивать алгоритмы и структуры данных под каждый компьютер,

можно ответить столь же резонно:

а всегда ли можно игнорировать их индивидуальные особенности.

Предложенные методики измерения времени выполнения операций могут быть использованы также для исследования системы "кэш – оперативная память" с целью как ее совершенствования, так и модификации под конкретные решаемые задачи конкретными программными средствами.

1. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы построение и анализ. – М.: МЦНМО, 2001. – 960 с.
2. *Седжвик Р.* Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск/Алгоритмы на графах. – СПб.:ООО "ДиасофтЮП", 2003. – 1136 с.
3. *Кнут Д. Э.* Искусство программирования. Т 1. Основные алгоритмы: 3-е изд. – М.: Издательский дом "Вильямс", 2000. – 720 с.
4. *Вирт Н.* Алгоритмы + структуры данных = программа. — М.: Мир, 1985., - 406 с.
5. *Макконнелл Дж.* Анализ алгоритмов. Вводный курс. – М.: Техносфера, 2002. – 304 с.
6. *Коннола Т., Бегг К., Страчан А.* Базы данных: проектирование, реализация и сопровождение. Теория и практика. – М.: Изд. дом "Вильямс", 2000. – 1120 с.
7. *Основы* инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль, Т.М. Коротун, В.Ю. Суслов – К.: Академпериодика, 2002. – 506 с.
8. *Лисков Б., Гатэг Дж.* Использование абстракций и спецификаций при разработке программ. – М.: Мир, 1989. – 424 с.
9. *Ахо А., Хопкрофт Д., Ульман Д.* Структуры данных и алгоритмы. – М.: Издательский дом "Вильямс", 2000. - 384 с.
10. *Зиглер К.* Методы проектирования программных систем. – М.: Мир, 1985. – 328 с.
11. *Шинкаренко В.И.* Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов // Проблемы программирования. – 2001. - №3-4 – С. 31-39.
12. *Пратт Т.* Языки программирования: разработка и реализация. - М.: Мир, 1979. - 574 с.
13. *Пратт Т., Зелкович М.* Языки программирования: разработка и реализация. – СПб.: Питер, 2002. - 688 с.
14. *Касперски К.* Техника оптимизации программ. Эффективное использование памяти. – СПб.: БХВ-Петербург, 2003. – 464 с.
15. *Соболь И.М.* Численные методы Монте-Карло. М. Наука. Гл. ред. физ.-мат. лит., 1973. – 311 с.
16. http://www.lavalys.com (с http://www.aida32.hu)