А. В. ПЕТРОСЯН

НЕКОТОРЫЕ СВОЙСТВА АЛГОРИТМОВ, СВЯЗАННЫЕ С ИХ РЕАЛИЗАЦИЕЙ НА ЭЦВМ

Настоящая работа посвящена одному из аспектов вопроса о повышении производительности устройств, перерабатывающих информацию (УПИ) [1]. Производительность УПИ, очевидно, можно увеличивать за счет повышения частотных характеристик элементов, из которых сделаны эти системы, т. е. за счет сокращения такта работы УПИ [2]. Однако нас будет интересовать именно тот случай, когда дальнейшее уменьшение такта УПИ невозможно. Последнее может быть вызвано двумя причинами.

Во-первых, при данном уровне развития техники дальнейшее увеличение быстродействия элементов невозможно. В этом случае нас будет интересовать величина максимально достижимой производительности УПИ при реализации данного алгоритма.

Во-вторых, при развитии вычислительной техники наступит такой момент, когда скорость работы УПИ будет ограничиваться скоростью света. Уже для того, чтобы получить в УПИ скорости, при которых возможно перерабатывать одну „единицу информации" за 1 *нсек*, средний радиус УПИ не должен превышать 30 *см*, так как время передачи „единицы информации" со скоростью распространения света на расстояние в 30 *см* составляет 1 *нсек*. Однако, как показано в работе [4], уменьшением габаритов тоже невозможно бесконечно увеличивать скорость работы УПИ, так что имеется некоторый естественный предел скорости работы УПИ. Время обращения к ячейкам запоминающего устройства должно удовлетворять неравенству

$$t \geqslant \sqrt[4]{N/d} \cdot 10^{-20} \text{ сек},$$

где N — количество разрядов запоминающего устройства, а d—средняя плотность ($г/см^3$) материала, использованного при изготовлении запоминающего устройства.

Отсюда, например, следует, что если $d = 20\ г/см^3$ и $N = 10^9$, то

вышеуказанное устройство не может иметь время обращения менее 10^{-18} *сек*.

В случае, когда дальнейшее увеличение быстродействия вообще не будет возможным, можно ставить вопрос о нахождении максимально доступных величин производительности УПИ при реализации того или иного алгоритма. Для того чтобы можно было оценить величину максимальной производительности УПИ как при данном уровне развития вычислительной техники, так и максимально достижимую, вообще необходимо изучить некоторые свойства алгоритмов.

Существует много способов уточнения интуитивного понятия алгоритма. Большинство из них, имея большое теоретическое значение, почти непригодно для практического описания алгоритмов, и поэтому анализ свойств алгоритмов, связанных с их реализацией на УПИ, вызывает известные трудности.

Реальная машина всегда допускает только конечное число разграниченных „состояний", и в соответствии с этим число различных начальных условий, которые реальная машина может перерабатывать в выходной результат, неизбежным образом тоже конечно. Кроме того, алгоритмы расчленяются на элементарные, выполнимые „механически" отдельные шаги, и так как обычно эти шаги применимы к бесконечному множеству начальных данных, то может потребоваться неограниченное число выполнений элементарных шагов, что также неосуществимо в реальных машинах. Поэтому приходится предполагать, что машины потенциально обладают бесконечными возможностями как по объему, так и по времени их работы.

Чтобы приблизить наши рассуждения к ситуациям, встречающимся на практике, мы будем предполагать, что в каждом конкретном случае заданный алгоритм должен быть применен лишь к конечной совокупности начальных данных, к которым этот алгоритм применим. При таком предположении количество элементарных тактов, которое потребуется для решения всей задачи, всегда будет конечным, хотя может оказаться настолько большим, что данная задача будет практически неразрешимой при данном уровне развития вычислительной техники или даже вообще неразрешимой.

Рассмотрим некоторое множество X, которое будем называть п а м я т ь ю, а его элементы назовем п е р е м е н н ы м и или я ч е й к а м и п а м я т и. Каждая ячейка памяти в каждый момент времени может принимать допустимые для нее значения из какого-то конкретного множества конструктивных объектов. Значение, которое принимает ячейка памяти в данный момент времени, будем называть содержимым этой ячейки. Совокупность значений всех ячеек памяти в данный момент времени назовем с о с т о я н и е м п а м я т и.

Рассмотрим k-местные операции $A(x_1, x_2, \cdots, x_k)$, где x_1, $x_2, \cdots, x_k$ являются ячейками памяти X. Операция для некоторых наборов содержимых ячеек $x_1, x_2, \cdots, x_k$ задает конструктивный способ построения нового конструктивного объекта β, называемого ре-

зультатом применения операции A к содержимому ячеек $x_1, x_2, \cdots, x_k$. Это обстоятельство мы будем записывать в виде

$$A(\alpha_1, \alpha_2, \cdots, \alpha_k) = \beta,$$

где $\alpha_1, \alpha_2, \cdots, \alpha_k$ являются содержимым ячеек $x_1, x_2, \cdots, x_k$ соответственно.

Конструктивный объект в свою очередь может стать содержимым некоторой ячейки y. Операцию A над заданными ячейками $x_1, x_2, \cdots, x_k$ (над содержимым которых надо применять операцию) и с заданной ячейкой y (содержимым которой становится результат применения операции) назовем формулой и запишем в виде

$$A(x_1, x_2, \cdots, x_k) \to y. \qquad (1)$$

Ячейки $x_1, x_2, \cdots, x_k$ назовем входными, а ячейку y — выходной ячейкой формулы. Впредь под словами „выполнить формулу (1)" мы будем понимать следующее: применить операцию A к содержимому ячеек $x_1, x_2, \cdots, x_k$ и заменить содержимое ячейки y полученным результатом.

Рассмотрим логические функции $\varphi(x_1, x_2, \cdots, x_s)$ над ячейками памяти $x_1, x_2, \cdots, x_s$, которые в зависимости от содержимого ячеек $x_1, x_2, \cdots, x_s$ принимают одно из двух значений: нуль или единицу.

Зададимся некоторым конечным множеством операций $A = (A_1, A_2, \cdots, A_l)$, конечным множеством логических функций $\Phi = (\varphi_1, \varphi_2, \cdots, \varphi_s)$ и конечным множеством ячеек $x = (x_1, x_2, \cdots, x_m)$.

Нам будет удобно описывать алгоритм с помощью его реализации в виде конечного дерева. Все узлы дерева делим на три типа. К первому принадлежат узлы, из которых выходит одна стрелка, ко второму типу — узлы, из которых выходят две стрелки, и наконец, к третьему типу принадлежат узлы, называемые выходными, из которых не выходит ни одной стрелки. Кроме того, дерево содержит один узел (любого из этих типов), в котором не оканчивается ни одна стрелка. Этот узел называется корнем дерева. В каждом узле первого типа помещаем некоторую формулу типа (1) из списка операции A. В каждом узле второго типа помещаем логическую функцию из множества Φ. Кроме того, одна из стрелок, выходящая из узлов второго типа, отмечена нулем, другая — единицей.

Описанный алгоритм $B(A, \Phi, X)$ работает следующим образом. Выделяется некоторое множество ячеек $x_{i_1}, x_{i_2}, \cdots, x_{i_{k_0}}$ из множества X, называемых входными ячейками алгоритма B. В эти ячейки записываются некоторые значения $\alpha_1, \alpha_2, \cdots, \alpha_{k_0}$, называемые начальными данными алгоритма B. Работа алгоритма начинается от корня дерева. Если очередной узел является узлом первого типа, то выполняется формула, записанная в данном узле, и нужно перейти в другой узел по направлению стрелки, выходящей из данного узла, а если стрелки нет, т. е. узел является выходным, то работа алгоритма этим завершается. Если рассматриваемый узел является узлом второго типа, то

вычисляется значение функции, записанной в данном узле, и производится ход к смежному узлу по той из двух выходящих стрелок, отметка которой совпадает с вычисленным значением функции.

Реализацией данного алгоритма мы назовем последовательность формул, расположенных на любой ветви дерева, начало которой совпадает с корнем дерева, а конец — с каким-нибудь из его выходов. Конкретную реализацию алгоритма обозначим последовательностью формул

$$B = B_1(x_{11}, x_{12}, \cdots, x_{1r_1}) \to y_1 * B_2(x_{21}, x_{22}, \cdots, x_{2r_2}) \to y_2 * \cdots * B_n(x_{n1}, x_{n2}, \cdots, x_{nr_n}) \to y_n, \quad (2)$$

где B — операции из A, а x_{ij} и y_k — ячейки памяти из X. Ясно, что все алгоритмы, описываемые конечными деревьями, будут иметь конечное число различных реализаций, соответствующих различным ветвям дерева.

Обозначим множество выходных ячеек k-ой формулы в реализации (2) через a_k.

Определение 1. Множество ячеек памяти

$$x_{вход} = a_1 + a_2/a_2y_1 + a_3/a_3(y_1 + y_2) + \cdots + a_n/a_n(y_1 + y_2 + \cdots + y_n)$$

назовем выходными ячейками алгоритма при данной реализации, а теоретико-множественную сумму входных ячеек по всем реализациям — входными ячейками алгоритма.

Обозначим через $x_{рез} = y_1 + y_2 + \cdots + y_n$ множество выходных ячеек реализации (2), а через $Ux_{рез}$ — сумму множеств выходных ячеек по всем реализациям данного алгоритма $B(A, \Phi, X)$.

Определение 2. Некоторое подмножество $x_{вых}$ множества $Ux_{рез}$.

$$x_{вых} \subset Ux_{рез},$$

назовем выходными ячейками алгоритма, а пересечение

$$x_{вых}(y_1 + y_2 + \cdots + y_n)$$

выходными ячейками данной реализации алгоритма (обычно это множество совпадает с $x_{вых}$).

В этих двух определениях знаки $+, /, .$ обозначают теоретико-множественные суммы, дополнение и умножение соответственно.

Данную реализацию алгоритма назовем выполнимой, если существуют такие начальные данные (т. е. содержимое входных ячеек алгоритма), для которых работа алгоритма проходит по ветви дерева, соответствующей данной реализации. В дальнейшем мы будем рассматривать только такие реализации алгоритма.

Определение 3. Последовательность из n-формул

$$\overline{B} = \overline{B}_1(\overline{x}_{11}, \overline{x}_{12}, \cdots, \overline{x}_{1r_1}) \to \overline{y} * \overline{B}_2(\overline{x}_{21}, \overline{x}_{22}, \cdots, \overline{x}_{2r_2}) \to \overline{y}_2 * \cdots * \overline{B}_n(\overline{x}_{n1}, \overline{x}_{n2}, \cdots, \overline{x}_{nr_n}) \to \overline{y}_n$$

назовем эквивалентной реализации (2), если:

1) эти последовательности отличаются друг от друга только порядком записи формул и их входными и выходными ячейками;

2) рассмотрим совокупность v всевозможных систем входных данных, для которых при работе алгоритма B получается всегда одна и та же реализация (2). Имеются такие взаимно-однозначные соответствия отдельно между входными и отдельно — выходными ячейками этих двух последовательностей, при которых каковы бы ни были входные значения, принадлежащие множеству v (равные для соответствующих ячеек), значения соответствующих выходных ячеек, после выполнения обеих последовательностей, совпадают.

Определение 4. Перестановку местами двух формул в реализации (2) назовем допустимой, если полученная после перестановки последовательность эквивалентна (2).

Рассмотрим условия, при которых перестановка двух соседних формул в некоторой реализации является допустимой. Это зависит от трех следующих множеств:

$$a_i y_{i+1},\quad y_i a_{i+1} \text{ и } y_i y_{i+1}.$$

Для простоты введем три переменные α, β, γ, которые определяются следующим образом:

$$\alpha = \begin{cases} 0, \text{ если} & a_i y_{i+1} \text{ пусто} \\ 1, \text{ если} & a_i y_{i+1} \text{ не пусто} \end{cases}$$

$$\beta = \begin{cases} 0, \text{ если} & y_i a_{i+1} \text{ пусто} \\ 1, \text{ если} & y_i a_{i+1} \text{ не пусто} \end{cases}$$

$$\gamma = \begin{cases} 0, \text{ если} & y_i y_{i+1} \text{ пусто} \\ 1, \text{ если} & y_i y_{i+1} \text{ не пусто} \end{cases}$$

Рассмотрим различные возможные здесь случаи.

1. Если α, β, γ равны соответственно 0, 0, 0, то очевидно, что перестановка i-й и $(i+1)$-й формулы допустима.

2. Если α, β, γ равны соответственно 0, 1, 0; 0, 1, 1; 1, 1, 0; 1, 1, 1, т. е. если результат операции B_i используется операцией $B_{i+1}(\beta=1)$, то перестановка i-й и $(i+1)$-й формулы, вообще говоря, недопустима. Однако, если B_i, B_{i+1} являются фиктивными для данной реализации алгоритма или если операция B_{i+1} фиктивно зависит от выходной переменной i-й формулы, то перестановка этих формул может оказаться допустимой.

3. Случай, когда α, β, γ равны соответственно 0, 0, 1 и 1, 0, 1, т. е. случай, когда выходные переменные i-й и $(i+1)$-й формул совпадают и ни одна из них не использует результата другой, можно считать невозможным, так как в этом случае или i-я формула является фиктивной для данной реализации алгоритма (так как ее результат, не использованный никакой другой операцией, портится операцией B_{i+1}) или формула $B_{i+1} \to y_{i+1}$ является тождественной.

4. Рассмотрим, наконец, случай, когда α, β, γ принимают значения 1, 0, 0, т. е. случай, когда одна или несколько входных ячеек i-й формулы совпадают с выходной ячейкой $(i+1)$-й формулы, а два других множества не пересекаются.

Ясно, что в этом случае непосредственная перестановка этих формул недопустима. Можно, однако, заменить выходную ячейку $(i+1)$-й формулы и все ячейки, совпадающие с ней во всех последующих формулах, новой ячейкой, не принадлежащей множеству $\bigcup_{i=1}^{n}(a_i + y_i)$.

В результате такой замены будет получена последовательность, эквивалентная исходной, но допускающая перестановку i-й и $(i+1)$-й формул.

Важно отметить, что перестановка двух соседних формул может оказаться фактически недопустимой только в тех случаях, когда последующая формула использует результат предыдущей, а в остальных случаях эта перестановка, как правило, бывает допустимой.

В реализации (2) перестановку $(i+k)$-й формулы на i-е место, т. е. перед i-й формулой, будем считать допустимой, если допустимы последовательные перестановки соседних формул: $(i+k)$-й с $(i+k-1)$-й, затем с $(i+k-2)$-й и т. д. вплоть до перестановки с i-й формулой.

Определение 4. Некоторую формулу в данной реализации назовем формулой первого ранга, если допустима ее перестановка на первое место. Формулу назовем $(r+1)$-го ранга, если после зачеркивания всех формул до r-го ранга включительно допустима ее перестановка на первое место.

Определение 5. Максимальный ранг всех формул в данной реализации назовем длиной данной реализации. Максимальную длину всех реализаций данного алгоритма назовем длиной алгоритма.

Предположим, что в данной реализации все формулы первого ранга переведены в начало (их расположение не существенно), затем расположены формулы второго, затем третьего и так далее до формул l-го ранга, которые расположены в конце последовательности формул. Такое расположение формул показывает, что для осуществления данной реализации можно сначала выполнить все формулы первого ранга, затем второго, третьего и т. д. до l-го ранга включительно. При этом, так как порядок выполнения формулы одного и того же ранга безразличен, то данную реализацию можно записать в виде

$$\left(\begin{array}{c} B_{11} \to y_{11} \\ B_{12} \to y_{12} \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ B_{1s_1} \to y_{1s_1} \end{array}\right)\left(\begin{array}{c} B_{21} \to y_{21} \\ B_{22} \to y_{22} \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ B_{2s_2} \to y_{2s_2} \end{array}\right) \cdots \left(\begin{array}{c} B_{l1} \to y_{l1} \\ B_{l2} \to y_{l2} \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ B_{lk_l} \to B_{lkl} \end{array}\right) \qquad (3)$$

где в каждой скобке собраны формулы одинакового ранга.

Рассмотрим устройства, которые умеют выполнять любые формулы над списком операций A и памятью x. Предположим, что эти устройства выполняют каждую отдельную формулу за одинаковое количество тактов [2]. Для простоты будем считать, что на выполнение одной формулы расходуется один такт. Если подключить несколько таких устройств к одной памяти, то можно будет выполнять за один такт уже не одну, а несколько формул одновременно. Такое объединение s-устройств с одной общей памятью назовем s-устройством.

Из представления (3) реализации алгоритма следует, что s-устройство может выполнить данную реализацию за l тактов, если только $s \geqslant \max\limits_{1<j<l}(k_j)$. Чтобы определить минимальное значение, для которого можно выполнить данную реализацию за l тактов, введем понятие ширины реализации алгоритма.

В представлении (3) реализации алгоритма количество столбцов оставим постоянным и рассмотрим всевозможные представления, которые получаются допустимыми перестановками формул из младших групп в старшие. Количество формул в этих группах обозначим через $k_1', k_2', \cdots, k_l'$ соответственно.

Определение 6. Число

$$h = \min \max_{1<j<l}(k_j'),$$

где минимум берется по всем представлениям вида (3), назовем шириной данной реализации алгоритма, а $H = \max h$, где максимум берется по всем реализациям, назовем шириной алгоритма.

Ясно, что h есть то минимальное s, для которого s-устройства могут выполнить за l-тактов реализацию алгоритма, длина которого равна l и ширина h.

В реализации алгоритма (2) последовательность формул $B_{i_1} \to y_{i_1}$; $B_{i_2} \to y_{i_2}$; $\cdots$; $B_{i_k} \to y_{i_k}$ назовем цепочкой, а k—ее длиной, если каждая следующая формула использует результат непосредственно предыдущей.

Лемма 1. $l = \max k$, где максимум берется по всевозможным цепочкам, то есть длина максимально длинной цепочки в данной реализации совпадает с длиной самой реализации.

Очевидно, что максимальная длина цепочки не может превосходить длины реализации алгоритма. Покажем, что существует хотя бы одна цепочка длины l. Для этого возьмем одну какую-нибудь формулу ранга l. Ясно, что существует хоть одна формула ранга $(l-1)$, результат которой использует данная формула ранга l, так как в противном случае ранг последней был бы меньше l. Перед данной формулой ранга l запишем эту формулу ранга $(l-1)$. Поступая так дальше, мы можем последовательно построить цепочку длины l. Этим завершается доказательство леммы.

Из всех вышеприведенных утверждений непосредственно следует следующая теорема.

Теорема 1.

1) Для того, чтобы на s-устройстве можно было выполнять всевозможные реализации данного алгоритма длины L и ширины H не более чем за L тактов, необходимо и достаточно, чтобы $s \geqslant H$.

2) Существуют такие реализации данного алгоритма, которые для любых s не могут быть выполнены s-устройствами меньше, чем за L-тактов.

Две введенные характеристики длины и ширины алгоритмов должны сыграть очень важную роль при проектировании УПИ с максимальной логической производительностью.

Ширина алгоритмов показывает, какими максимальными свойствами параллелизма обладает последний, а длина характеризует величину максимальной логической производительности, которую можно получить при реализации данного алгоритма.

Вычисление длины алгоритма сравнительно несложно, но определение ширины алгоритма, даже для простых алгоритмов, представляет очень сложную задачу. Для иллюстрации вышеприведенных понятий рассмотрим пример на вычисление суммы n-чисел

$$z = \sum_{i=1}^{n} z_i, \quad n \geqslant 2.$$

Пусть множество A состоит из одной операции $A(a, b) = a + b$ (сложение двух чисел). Алгоритм решения этой задачи возьмем следующим. Сложим попарно все z_i (если n нечетно, то все, кроме одного). В результате получим $\left[\frac{n+1}{2}\right]$ чисел, с которыми будем поступать таким же образом и так до тех пор, пока не получим одно число, которое и будет результатом. Ясно, что этот алгоритм имеет всего одну реализацию. Если поместить числа $z_1, z_2, \cdots, z_n$ в ячейки $x_1, x_2, \cdots, x_n$ соответственно, то, например, при $n = 9$ получим реализацию:

$$A_1(x_1, x_2) \to x_{10} * A_2(x_3, x_4) \to x_{11} * A_3(x_5, x_6) \to x_{12} * A_4(x_7, x_8) \to x_{13} *,$$
$$* A_5(x_{10}, x_{11}) \to x_{14} * A_6(x_{12}, x_9) \to x_{15} * A_7(x_{14}, x_{15}) \to x_{16} *$$
$$* A_8(x_{16}, x_{13}) \to z.$$

Ясно, что формулы 1, 2, 3, 4 являются формулами первого ранга, формулы 5, 6—второго ранга, 7-я формула—третьего; а 8-я—четвертого ранга, так что длина этого алгоритма равна 4. Представление этой реализации в виде (3) будет:

$$\begin{pmatrix} A_1(x_1, x_2) \to x_{10} \\ A_2(x_3, x_4) \to x_{11} \\ A_3(x_5, x_6) \to x_{12} \\ A_4(x_7, x_8) \to x_{13} \end{pmatrix} \begin{pmatrix} A_5(x_{10}, x_{11}) \to x_{14} \\ A_6(x_{12}, x_9) \to x_{15} \end{pmatrix} \Big(A_7(x_{14}, x_{15}) \to x_{16} \Big) \times$$
$$\times \Big(A_8(x_{16}, x_{13}) \to z \Big).$$

Формулу 4 здесь можно перенести либо во вторую группу, либо в третью и представить эту реализацию в виде

$$\begin{pmatrix} A_1(x_1, x_2) \to x_{10} \\ A_2(x_3, x_4) \to x_{11} \\ A_3(x_5, x_6) \to x_{12} \end{pmatrix} \begin{pmatrix} A_5(x_{10}, x_{11}) \to x_{14} \\ A_6(x_{12}, x_9) \to x_{15} \end{pmatrix} \begin{pmatrix} A_4(x_7, x_8) \to x_{13} \\ A_7(x_{14}, x_{15}) \to x_{16} \end{pmatrix} \times$$

$$\times (A_8(x_{16}, x_{13}) \to z).$$

Отсюда видно, что ширина этого алгоритма равна 3. Для случая, когда $n = 2^p$ после первого шага количество полученных чисел будет равно $\left[\frac{n+1}{2}\right] = \left[\frac{2^p+1}{2}\right] = 2^{p-1}$, откуда следует, что длина такого алгоритма равна p, а ширина—количеству формул первого ранга, т. е. $\frac{n}{2}$. Если теперь n удовлетворяет условию $2^p < n \leqslant 2^{p+1}$, то ясно, что в таком случае длина алгоритма будет

$$L(n) = p + 1 = [\log_2(n-1)] + 1. \tag{4}$$

Определение ширины этого алгоритма представляет довольно сложную задачу, решенную в недавно вышедшей работе ([3]), где для вычисления ширины выведена явная формула

$$H(n) = \begin{cases} \left[\frac{n - 2^{[\log_2 n]-1} - 1}{2}\right] + 1, & \text{если } 2^{[\log_2 n]} < n < 2^{[\log_2 n]} + 2^{[\log_2 n]-1} \\ n - 2^{[\log_2(n-1)]}, & \text{если } 2^{[\log_2(n-1)]} + 2^{[\log_2(n-1)]-1} \leqslant n \leqslant 2^{[\log_2(n-1)]+1} \end{cases}$$

Следует отметить, что эти характеристики вычислены лишь для некоторых достаточно простых задач.

Если построение s-устройства, для которого $s \geqslant H$, невозможно, то для выполнения алгоритма потребуется больше чем L тактов. Обозначим минимальное количество тактов, которое необходимо для реализации алгоритма для данного s, через $T(s)$. Ясно, что $T(s)$—невозрастающая функция, равная N, когда $s = 1$, и равная L ($L \ll N$), когда $s \geqslant H$ (H—ширина алгоритма, N—число формул в реализации алгоритма с максимальным числом формул).

Если для некоторого алгоритма известна лишь величина N, то для величины $T(s)$ можно дать лишь следующие оценки:

$$\left[\frac{N-1}{s}\right] + 1 \leqslant T(s) \leqslant N. \tag{5}$$

Если же заданы и другие характеристики алгоритма, такие, как ширина H и длина L, то оценку (5) для $T(s)$ можно уточнить. В частности, верна следующая теорема.

Теорема 2. $T(s)$ удовлетворяет следующим неравенствам:

$$\left[\frac{N-1}{s}\right] + 1 \leqslant T(s) \leqslant \begin{cases} L, & \text{если } s \geqslant H \\ \left[\frac{N-L}{s}\right] + L, & \text{если } s < H \end{cases} \tag{6}$$

Оценка слева очевидна. Очевидна также оценка справа для случая, когда $s > H$. Рассмотрим оценку справа в случае, когда $s < H$. Разобьем формулы каждого ранга на группы по s формул в каждой, кроме разве лишь одной группы, в которой может оказаться меньше, чем s формул.

Из каждого множества формул одного ранга выделим неполную группу, если таковая есть, а если неполной группы нет, то выделим одну полную группу. Количество оставшихся групп по всем рангам будет не более, чем $\left[\frac{N-L}{s}\right]$, так как всего операций было N, а из них мы выделили, по крайней мере, L операций (L неполных групп). Таким образом, всего полных и неполных групп будет не более, чем $\left[\frac{N-L}{s}\right]+L$, что и утверждает правая часть неравенства.

Рассмотрим приведенный выше пример сложения n чисел.

Если, например, $n=21$, то $N=20$, а из формул (3) и (4) получим $L=5$, $H=7$. Из неравенств (6) следует

$$7 \leqslant T(3) \leqslant 10 \text{ (в действительности } T(3)=8),$$
$$4 \leqslant T(6) \leqslant 7 \text{ (в действительности } T(6)=6).$$

Нетрудно убедиться, что оценки (6) нельзя улучшить. Для этого рассмотрим алгоритм, реализация которого записывается в следующем виде:

$$\left(\begin{array}{l} A_1(x_{11}, x_{12}, \cdots, x_{1n}) \to x_{21} \\ A_2(x_{11}, x_{12}, \cdots, x_{1n}) \to x_{22} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ A_n(x_{11}, x_{12}, \cdots, x_{1n}) \to x_{2n} \end{array}\right)\left(\begin{array}{l} A_1(x_{21}, x_{22}, \cdots, x_{2n}) \to x_{31} \\ A_2(x_{21}, x_{22}, \cdots, x_{2n}) \to x_{32} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ A_n(x_{21}, x_{22}, \cdots, x_{2n}) \to x_{3n} \end{array}\right)\cdots$$

$$\cdots\left(\begin{array}{l} A_1(x_{k1}, x_{k2}, \cdots, x_{kn}) \to z_1 \\ A_2(x_{k1}, x_{k2}, \cdots, x_{kn}) \to z_2 \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ A_n(x_{k1}, x_{k2}, \cdots, x_{kn}) \to z_n \end{array}\right),$$

где $A_1, A_2, \cdots, A_n$ — различные операции, существенно зависящие от своих аргументов.

Из этого представления реализации алгоритма следует, что ширина этого алгоритма $H=n$, а длина $L=k$.

Если теперь взять $n=2s+1$, то для выполнения формул каждого ранга необходимо $(r+1)$ тактов, а для реализации всего алгоритма потребуется $k(r+1)$ тактов, т. е. $T(s)=k(r+1)$. Для этой реализации

$$\left[\frac{N-L}{s}\right]+L=\left[\frac{nk-k}{s}\right]+k=\left[\frac{(rs+1)k-k}{s}\right]+k=$$
$$=\left[\frac{rsk}{s}\right]+k=k(r+1),$$

что совпадает с $T(s)$. Если же $n = rs$, то для выполнения формул каждого ранга необходимо r тактов, а для выполнения всего алгоритма необходимо $T(s) = rk$ тактов. В этом случае

$$\left[\frac{N-L}{s}\right] + 1 = \left[\frac{nk-1}{s}\right] + 1 = \left[\frac{srk-1}{s}\right] + 1 =$$

$$= \left[rk - \frac{1}{s}\right] + 1 = rk,$$

что совпадает с $T(s)$.

Ա. Վ. ՊԵՏՐՈՍՅԱՆ

ԱԼԳՈՐԻԹՄՆԵՐԻ ՈՐՈՇ ՀԱՏԿՈՒԹՅՈՒՆՆԵՐ՝ ԿԱՊՎԱԾ ԷԼԵԿՏՐՈՆԱՅԻՆ ՀԱՇՎԻՉ ՄԵՔԵՆԱՆԵՐԻ ՎՐԱ ՆՐԱՆՑ ԻՐԱՑՄԱՆ ՀԵՏ

Ա մ փ ո փ ո ւ մ

Հոդվածում ուսումնասիրվում է ալգորիթմների այնպիսի հատկություններ, որոնք կապված են էլեկտրոնային հաշվիչ մեքենաների պարամետրերի հետ։ Մտցված են 2 նոր գաղափարներ, ալգորիթմի «լայնություն» և «երկարություն», որոնք թույլ են տալիս ստանալ գնահատականներ, ինչպես վերից, այնպես էլ վարից, հաշվիչ մեքենաների մաքսիմալ արտադրողականության համար։

ЛИТЕРАТУРА

1. *Петросян А. В.* Определение некоторых характеристик алгоритмов, влияющих на конструкцию вычислительной машины. XLII научная сессия Ереванского государственного университета (тезисы докладов), Ереван, 1962.
2. *Петросян А.* В. Логическая производительность и эффективность УЦВМ. Труды Вычислительного центра АН АрмССР и ЕГУ, Ереван, 1963, 59—65.
3. *Бекишев Г. А.* Об алгоритмах, эффективно реализуемых на вычислительных системах. Сб. „Вычислительные системы". вып. 7, Новосибирск, 1963.
4. A. Basic dimitation on the Speed of Digital Computers. JRF Trans. on Electron. Comput". 1961, 10, № 3, 530—531.