# Оптимальное локальное планирование инструкций для нескольких процессоров при отсутствии структурных конфликтов

С. С. Сурин
sergey.surin@pobox.spbu.ru

В статье рассматривается задача построения оптимального по критерию времени расписания вычисления набора выражений многопроцессорной Вычислительной Системой. Набор выражений не должен содержать общих подвыражений. Все операции, соответствующие листовым вершинам графа зависимостей по данным, могут стартовать в любом такте интервала вычисления. Вычислительная Система может содержать любое фиксированное число идентичных процессоров. Каждый процессор работает в конвейерном режиме. Все операции имеют одну и ту же длительность выполнения любым процессором. Выясняется структура оптимальных расписаний и предлагается алгоритм решения этой задачи, имеющий полиномиальную сложность.

## Введение

Вычислительная Система в нашей задаче состоит из $Q > 1$ процессоров и общей памяти двух видов: основной и регистровой. Здесь $Q$ — фиксированное число.

Каждый процессор работает в конвейерном режиме и выполняет арифметические и логические операции с одним и двумя ар-

гументами и операции обмена данными между основной и регистровой памятью. Результат каждой арифметической и логической операции помещается в регистр.

Каждая операция имеет несколько модификаций. В одной модификации все ее аргументы перед началом выполнения должны находиться в регистрах. В других модификациях все или некоторые аргументы могут быть литералами, которые указываются в записи операции.

Объем памяти обоих видов считается неограниченным.

Все операции (включая операции обмена данными между основной и регистровой памятью) имеют одну и ту же длительность выполнения любым процессором, измеряемую тактами. Эту величину мы обозначим через $t$.

Аппаратных ограничений на последовательность выполнения операций не имеется.

Мы будем изучать организацию работы Вычислительной Системы при вычислении значений набора выражений.

Набор выражений не содержит общих подвыражений и задается графом зависимостей по данным, который, следовательно, является лесом. Все операции, соответствующие листовым вершинам этого графа, могут стартовать в любом порядке, начиная с первого такта интервала вычисления.

Требуется составить расписание выполнения Вычислительной Системой операций набора, по которому он вычислялся бы за минимальное время.

Задачи планирования работы Вычислительных Систем, как правило, являются $NP$-полными задачами. Это было хорошо известно еще до появления процессоров с конвейерным способом выполнения операций [1, 4].

В 1967-ом году Т. С. Ху [5] доказал полиномиальную сложность задачи, которая отличается от нашей лишь тем, что в ней процессоры работают не в конвейерном режиме, как у нас, а в режиме последовательного выполнения операций, причем их число заранее не фиксировано. В [4] приведено следующее доказательство этого результата. Оно состоит из двух частей. Сначала строится полиномиальный алгоритм составления кратчайшего, то есть оптимального по времени, расписания для случая, когда каждая операция на каждом процессоре выполняется в течение одного такта. Далее из того, что процессоры работают в последовательном режиме, делается вывод о том, что этот алгоритм решает задачу при любом

равном для всех операций на всех процессорах времени выполнения.

Такой способ доказательства для задачи с конвейерными процессорами не приемлем потому, что каждый из них может в каждом такте начинать выполнение одной операции и одновременно с этим продолжать выполнять несколько (в нашем случае не более $t$) других операций. Последовательные процессоры такой возможности не имеют.

Позднее была доказана полиномиальная сложность и найдены эффективные алгоритмы решения еще нескольких задач, близких по содержанию к нашей, в том числе и для конвейерных процессоров (см. например [6, 7, 9]), но нам не удалось найти среди них ни одной, которая бы совпадала с нашей задачей или включала ее в себя в качестве частного случая.

Коффман [4] и Грэхэм [7] обобщили алгоритм Ху на случай, когда набор выражений (далее набор $C$) задается произвольным орграфом (далее графом $G(C)$), а длительности выполнения всех операций $i$ на всех процессорах $j$ (далее величины $\tau_{ij}$), как и в [5], равны между собой. Но при этом пришлось пожертвовать произвольным числом процессоров. Теперь их в Системе — только два. Оценка сложности предложенного ими алгоритма — $O(n^2)$ [4], где $n$ — число операций в наборе. Алгоритм гарантирует оптимальность расписания лишь в том случае, когда он получает на вход граф $G(C)$, не содержащий транзитных дуг, или транзитивное замыкание этого графа. Наилучший алгоритм, приводящий исходный граф $G(C)$ к такому виду имеет сложность, как утверждается в [8], $O(min(en, n^{2.61}))$, где $e$ — число дуг в графе $G(C)$.

Габов [8] для решения этой же задачи предложил более эффективный алгоритм. Оценка его сложности $O(e + n\alpha(n))$, где $\alpha(x)$ — очень медленно растущая функция аргумента $x$ (инверсия функции Аккермана [15]). Кроме того, алгоритм Габова может работать с любым графом $G(C)$.

Если $G(C)$ — произвольный орграф, все $\tau_{ij}$ равны, а число $Q$ процессоров заранее не фиксировано, то задача является $NP$-полной [16]. Она становится $NP$-полной и тогда, когда $Q \geq 2$, а величины $\tau_{ij}$ принимают значения 1 и 2 (там же). Если $G(C)$ — произвольный орграф, $Q \geq 3$ и фиксировано, а величины $\tau_{ij}$ равны, то ее сложность не определена [1].

Применение алгоритма описанного в [7] к решению задачи, в которой $G(C)$ — произвольный орграф, все $\tau_{ij}$ равны, а $Q > 2$ и

заранее фиксировано, приводит к расписанию, отношение длины которого к длине соответствующего кратчайшего расписания стремится снизу к величине $2 - 2/Q$ с ростом $Q$, [4, 11].

Алгоритмы, описанные в [5,7,8] относятся к классу так называемых списочных алгоритмов. Такой алгоритм состоит из двух частей. Первая часть упорядочивает операции набора $C$ в виде списка приоритетов, просматривая который, вторая часть строит искомое расписание.

Алгоритмы, описанные в [5] и [7] отличаются лишь процедурами упорядочивания, то есть первыми частями алгоритма, хотя эти процедуры основаны на одной и той же идее: упорядочивать операции с помощью присвоения им числовых меток. Операции в списке располагаются в порядке убывания значений меток. В [13] показано, что способ создания списка, описанный в [7], может быть реализован со сложностью $O(e + n\alpha(n))$.

В [8] операции разбиваются на группы по удаленности их от операций, вырабатывающих значения выражений (заключительных операций набора). Заключительные операции образуют свою собственную группу. В списке группы располагаются по убыванию их удаленности от группы заключительных операций, которая становится последней группой списка. Операции внутри каждой группы списка располагаются в произвольном порядке. Такой порядок операций в группах несколько усложняет вторую часть алгоритма, при выполнении которой приходится решать, какую операцию из следующих групп списка следует переместить в текущую группу, если оказывается, что она на текущем шаге алгоритма содержит нечетное число операций. Если текущая группа будет содержать нечетное число операций, то в последнем из тактов, в которых выполняются ее операции один из двух процессоров будет простаивать. Перемещаемые операции следует выбирать так, чтобы минимизировать число ситуаций, когда текущая группа содержит нечетное число операций. Для построения списка Габов предлагает воспользоваться алгоритмом топологической сортировки, описанным в [2], изменив его так, чтобы получаемый этим алгоритмом линейный список распадался на группы. Оценка сложности этого алгоритма составляет $O(e + n)$.

Фактически списки, которые строят алгоритмы в [5] и [7] можно разбить на такие же группы. Жесткий порядок расположения операций в этих группах, задаваемый метками, позволяет легко находить перемещаемую операцию: ею становится первая же в списке

готовая к выполнению в текущем такте операция из другой группы.

Изучению различных вариантов задачи с задержками при передаче данных от операции к операции посвящены работы [6,9,10,12].

Бернштейн и Гартнер [6] модифицировали алгоритм из [7] для решения задачи, в которой набор выражений, описывается орграфом, а Вычислительная Система имеет только один процессор, выполняющий одно-тактные операции в конвейерном режиме, но при передаче данных между некоторыми операциями происходит задержка длительностью в один такт. Этот алгоритм наследует все свойства своего прототипа: оценки сложности и необходимость иметь на входе граф $G(C)$, не содержащий транзитных дуг.

В работе [9] задача с задержками при передаче данных ставится таким образом, чтобы определить, существует ли вообще для данного набора выражений расписание (кратчайшее расписание), по которому все вычисления можно было бы выполнить в заданный срок.

Набор выражений $C$ задается произвольным орграфом $G(C)$. Вычислительная Система имеет неограниченное число параллельно работающих процессоров, выполняющих одно-тактные операции в последовательном режиме. Если операция $x$ выполняется на процессоре $P_i$ в такте $t$, и в графе $G(C)$ есть дуга $(y, x)$, то операция $y$ может выполняться на процессоре $P_i$ не позднее такта $t-1$ или на процессоре $P_j \neq P_i$ не позднее такта $t-1-\delta(x,y)$. Величина $\delta(x,y)$ определяет длительность задержки и является целочисленной функцией, заданной на дугах графа $G(C)$. Требуется определить существует ли расписание, длина которого не превосходит заданную величину.

Эту задачу мы для краткости будем называть $DS$-задачей *(Delay Scheduling Problem)*. В работе [9] в $DS$-задаче выделяются два частных случае:

- $UDS$-задача *(Uniform Delay Scheduling Problem)*; это — $DS$-задача, в которой $\delta(x,y) = \tau$, где величина $\tau$ зависит от графа $G(C)$ вцелом и может расти с ростом числа вершин в этих графах.
- $CDS$-задача *(Constant Delay Scheduling Problem)*; это $DS$-задача, в которой $\delta(x,y) = c$, где $c$ — универсальная константа, не зависящая от графа $G(C)$.

В [12] показано, что $UDS$-задача — $NP$-полная. Там же описан простой приближенный алгоритм ее решения, дающий расписание,

длина которого не превосходит удвоенной длины соответствующего кратчайшего расписания.

В работе [10] предложен алгоритм решения $UDS$-задачи, основанный на стратегии динамического программирования и имеющий сложность $n^{O(\tau)}$.

В [9] изучается $DS$-задача, в которой граф $G(C)$ является деревом и утверждается, что даже в этом случае она относится к числу трудных задач. В этой работе доказано, что $NP$-полными являются $UDS$-задача, если $G(C)$ — (1,2)-дерево, и $DS$-задача, если граф $G(C)$ — полное бинарное дерево. Там же предложен полиномиальный алгоритм решения $UDS$- задачи, в которой граф $G(C)$ является полным $k$-арным деревом. Сложность этого алгоритма $O(n^2 \log n)$. Там же доказано, что $CDS$-задача является полиномиальной задачей, если в графе $G(C)$ в каждую вершину входит не больше трех дуг, а длительность задержки равна одному такту.

Во всех уже рассмотренных работах неявно предполагалось, что ресурс памяти, в частности регистровой памяти, не ограничен. Если, например, ограниченным является объем регистровой памяти, то при вычислении арифметического выражения для хранения промежуточных результатов приходится использовать основную память. Так как не у каждой арифметической операции есть модификация, в которой любым аргументом может быть адрес основной памяти, то для записи в основную память и извлечения из нее промежуточных результатов в программу приходится включать операции обмена данными между основной памятью и регистрами, что удлиняет ее код. Именно такая ситуация рассматривается в работе [14].

В этой работе Вычислительная Система состоит из одного процессора, неограниченной памяти и $N \geq 1$ общих регистров. Набор операций процессора состоит из операций обмена между памятью и регистрами и бинарных арифметических операций. Каждая арифметическая операция свой результат помещает в регистр и имеет две модификации. В одной модификации оба ее аргумента должны перед ее выполнением находиться в регистрах, а в другой правый аргумент находится в памяти. В работе решается задача построения программы вычисления арифметического выражения, не содержащего общих подвыражений, имеющей минимальную длину.

Модель Вычислительной Системы и информационная структура набора выражений в нашей задаче очень просты, но все еще реалистичны. Эта простота позволяет задаче сохранить, как мы

увидим, полиномиальную сложность при любых значениях параметров Системы, а именно: числа процессоров и равного для всех операций времени выполнения на всех процессорах.

В [3] мы предложили полиномиальный алгоритм решения задачи, которая совпадает с рассматриваемой здесь по всем пунктам, кроме одного: в ней работает один процессор. Это — списочный алгоритм. Список, который строит его первая часть, совпадает со списком, который использует алгоритм, описанный в [8]. Вторая часть больше напоминает вторую часть алгоритма, описанного в [4] для решения задачи из [7]: она не требует специальной процедуры поиска операций в следующих группах, готовых стартовать в текущем такте; первая же в списке операция с этим свойством становится искомой. В [3] дано подробное обоснование того факта, что этот алгоритм строит кратчайшее расписание работы процессора в конвейерном режиме.

Наша задача является обобщением задачи из [3]. Такая естественная модификация Системы, как замена в ней единственного процессора на произвольное фиксированное их число, все же вызывает некоторые изменения и в модели расписания, использованной в [3]. В данной статье мы вносим в эту модель и в алгоритм из работы [3] соответствующие изменения и показываем, как они отражаются на ходе доказательства оптимальности построенного этим алгоритмом расписания.

## 1. Структура оптимального расписания

Множество наборов выражений, не содержащих общих подвыражений, обозначим через $C_T$.

Пусть $C$ — набор выражений из множества $C_T$. Через G(C) обозначим его граф зависимостей по данным. Согласно условиям задачи граф $G(C)$ является лесом.

Если операция не использует результатов других операций, то назовем ее листовой, а в противном случае — нелистовой операцией. Листовые операции соответствуют листовым вершинам графа $G(C)$. Если результат операции не используется другими операциями набора то соответствующую ей вершину графа $G(C)$ будем называть вершиной выхода.

Как и в [3] все операции набора выражений $C$ упорядочим в виде списка, который будем называть списком $A$. Для его определения введем следующие термины.

Длиной пути, соединяющего две вершины графа $G(C)$, назовем число входящих в этот путь дуг.

Удаленностью данной вершины $p$ от множества вершин выхода назовем длину пути с началом в вершине $p$ и концом в вершине выхода[1]. Удаленность вершины выхода от множества вершин выхода по этому определению равна нулю.

Все операции в списке $A$ разделены на группы. В одну группу входят операции, которым соответствуют вершины графа $G(C)$, равноудаленные от множества вершин выхода. Число групп обозначим через $k_{max}$. В списке группы располагаются в порядке уменьшения удаленности. Упорядоченность входящих в одну группу операций — произвольная (эти операции не связаны зависимостями по данным).

На рис.1 представлен пример списка $A$ для набора из двух выражений. В этом списке символом L(X) обозначена операция загрузки в регистр значения переменной X, находящегося в основной памяти. Список состоит из четырех групп ($k_{max} = 4$).

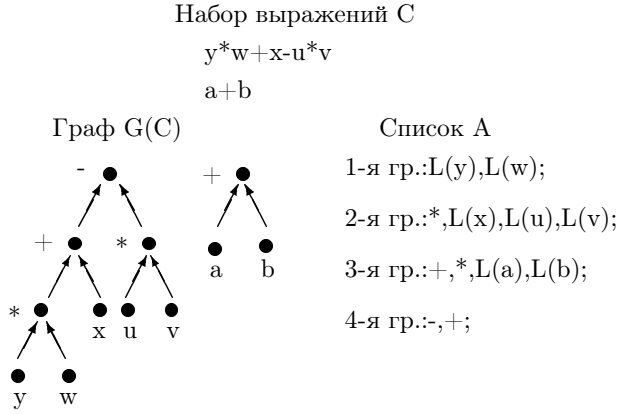


*Рис. 1.* Набор выражений, его граф и список A.

Когда процессоры Системы выполняют операции в конвейерном режиме, интервал времени вычисления набора выражений можно

[1]Так как граф $G(C)$ является лесом, то из каждой его вершины, которая не является вершиной выхода, выходит только один путь с концом в вершине выхода.

разбить на две части. Первая часть содержит первые такты выполнения (то есть стартовые такты) всех операций набора. Эту часть мы будем называть стартовой частью интервала вычисления.

В течение второй части интервала завершают свое выполнение последние операции набора. Так как все операции имеют одну и ту же длительность выполнения на всех процессорах, равную $t$ тактам, то длительность второй части всегда равна $t-1$ тактам.

Задать расписание вычисления набора выражений значит указать для каждой его операции стартовый такт и процессор, на котором она будет выполняться.

В отличие от [3] в нашей Вычислительной Системе одновременно могут стартовать не одна, а не более $Q$ операций, так как один процессор в любом такте может начать выполнение только одной операции.

Множество операций, начинающих свое выполнение (стартующих) в одном и том же такте, мы будем называть *пакетом* операций.

Такт интервала вычисления, в котором по данному расписанию стартует пакет из $Q$ операций, мы будем называть *насыщенным* тактом, а в противном случае — *ненасыщенным* тактом. В частности, ненасыщенный такт, в котором не стартует ни одна операция, будем называть пустым тактом. Таким образом, вторая часть интервала вычисления состоит из пустых тактов.

Если операция $p$ имеет вычисляемые аргументы, то операцию, которая по данному расписанию завершает их вычисление, мы будем называть операцией, завершающей подготовку операции $p$ к выполнению, и обозначать через $c(p)$. Возможны расписания, по которым вычисление аргументов одновременно завершают две операции (операция $p$ может иметь два вычисляемых аргумента). В этом случае в качестве операции $c(p)$ выбирается любая из них.

По условиям задачи листовая операция $p$ может начать свое выполнение в любом такте стартовой части интервала вычисления. Нелистовая может стартовать только после того, как закончит выполнение ее операция $c(p)$.

Такты, следующие за последним тактом выполнения операции $c(p)$, будем называть тактами готовности операции $p$ к выполнению и обозначать через $\theta(p)$. Первый из этих тактов будем обозначать через $\theta_1(p)$.

Тактом $\theta_1$ для каждой листовой операции является первый такт интервала вычисления.

Мы будем говорить, что расписание допустимо по времени, если по нему каждая нелистовая операция $p$ стартует в одном из тактов $\theta(p)$. Далее термином расписание мы будем обозначать только допустимые по времени расписания.

```
L y,r1;      L w,r2;
NOP;         NOP;

L x,r3;      NOP;         L v,r5;      M r1,r2,r1;
L u,r4;      NOP;         NOP;         NOP;

L a,r6;      NOP;         A r1,r3,r1;
L b,r7;      NOP;         M r4,r5,r4;

A r6,r7,r6; NOP;         S r1,r4,r1;
NOP;         NOP;         NOP;
```

*Рис. 2.* Расписание s1.

На рис.2 приведено расписание вычисления набора выражений $C$ изображенного на рис.1. Вычислительная Система в этом примере состоит из двух процессоров ($Q = 2$). Длительность выполнения всех операций равна трем тактам ($t = 3$). Символом NOP обозначается ситуация, когда в данном такте на данном процессоре не стартует ни одна операция. Легко проверить допустимость этого расписания.

Оптимальным по времени расписанием мы будем называть расписание, по которому набор выражений вычисляется за минимальное время.

Каково бы ни было допустимое расписание, всегда можно указать в каждой группе списка $A$ одну или несколько (числом не больше $Q$) операций, которые по этому расписанию входят в один пакет операций, возможно, с операциями из других групп списка $A$ и стартуют по времени последними среди операций своей группы. Совокупность всех таких операций из всех групп списка $A$ будем называть *критическим* набором операций данного расписания.

По любому расписанию можно указать $k_{max}$ тактов, в которых стартуют все операции критического набора. Этими тактами разобьем стартовую часть интервала вычисления на отрезки которые

назовем стартовыми интервалами. Пронумеруем группы в порядке их расположения в списке $A$, а стартовые интервалы в порядке их следования во времени. По условиям задачи все операции на всех процессорах имеют одну и ту же длительность выполнения. Поэтому каждая группа и стартовый интервал, в последнем такте которого стартуют ее операции, входящие в критический набор, получат один и тот же номер. Стартовый интервал с номером $k$ назовем стартовым интервалом $k$-ой группы списка $A$.

При любом $k : 1k < k_{max}$, ни одна операция $k$-ой группы не может стартовать в стартовом интервале с номером больше $k$.

Пусть $S$ — расписание вычисления набора выражений $C \in C_T$.

**Замечание 1.** *Любой такт $\tau$ стартовой части интервала вычисления может стать стартовым тактом для пакета не более, чем из $Q$ операций. Некоторые из этих операций или даже все они могут завершать подготовку к выполнению других операций, причем каждая — только одной, так как граф $G(C)$ — лес. Следовательно, такт $\tau + 1$ может стать тактом $\theta_1$ не более, чем для $Q$ нелистовых операций.*

Для стартовых интервалов в нашей задаче справедлива следующая теорема, которая характеризует их и в задаче, рассмотренной в [3]. Доказательство этой теоремы почти дословно повторяет ее доказательство в [3]. Поэтому здесь мы его опускаем.

Обозначим через $\lceil X \rceil$ — округленное до целого с избытком вещественное число $X$.

**Теорема 1.** *Каково бы ни было расписание вычисления набора выражений, длительность каждого стартового интервала, кроме 1-го, не может быть меньше $t$ тактов, то есть длительности выполнения операций. Длительность 1-го стартового интервала не может быть меньше величины $\lceil m/Q \rceil$, где $m$ — число операций в 1-ой группе списка $A$.*

Через $m(S, k)$ обозначим число всех операций, стартующих по расписанию $S$ в $k$-ом стартовом интервале, а через $T(S, k)$ — его длительность.

Стартовый интервал с номером $k$ будем называть:

- полным интервалом, если $T(S, k) = t$ и $m(S, k) = t * Q$;
- неполным интервалом, если $T(S, k) = t$, а $m(S, k) < t * Q$;
- переполненным интервалом, если $k = 1$, или $m(S, k) > t * Q$ и $T(S, k) = \lceil m(S, k)/Q \rceil$.

Из этого определения следует, что каждый полный интервал состоит только из насыщенных тактов, так как в противном случае он имел бы длительность больше $t$ тактов. Неполный интервал имеет ненасыщенные такты, среди которых могут быть и пустые.

Можно показать, что существуют расписания, по которым все, кроме, может быть, последнего, такты каждого переполненного интервала являются насыщенными тактами. Далее мы будем рассматривать только такие расписания. 1-й стартовый интервал по определению всегда является переполненным интервалом независимо от числа стартующих в нем операций.

Расписание $S$ будем называть нормализованным расписанием, если каждый стартовый интервал по этому расписанию является либо полным, либо неполным, либо переполненным интервалом.

Фрагментом расписания назовем такую его часть, в которой указаны стартовые такты всех операций, стартующих в одном или нескольких следующих подряд друг за другом стартовых интервалах. Эти интервалы будем называть стартовыми интервалами, принадлежащими данному фрагменту расписания.

Будем говорить, что расписание *почти распадается* на два фрагмента, если выполнены условия:

- первому фрагменту принадлежат первые $kf1 < k_{max}$ стартовых интервалов, а второму — остальные $k_{max} - kf1$ интервалов;
- стартовый интервал с номером $kf1$ может быть только полным или переполненным интервалом;
- если стартовый интервал с номером $kf1$ является полным интервалом, то ни одна операция из группы с номером больше $kf1$ списка $A$ не стартует в стартовом интервале с номером не больше $kf1$;
- если стартовый интервал с номером $kf1$ является переполненным интервалом, то операции из групп с номерами больше $kf1$ списка $A$ могут стартовать не раньше последнего такта этого интервала.

**Теорема 2.** *(достаточные условия оптимальности расписания по времени) Любое расписание оптимально по времени, если оно почти распадается на два фрагмента, удовлетворяющие условиям: все такты стартовых интервалов, принадлежащих первому*

*фрагменту, кроме последнего такта $kf1$-го интервала, являются насыщенными тактами, а длительность каждого стартового интервала, принадлежащего второму фрагменту, равна $t$ тактам.*

*Доказательство.* Суммарная длительность стартовых интервалов, принадлежащих по отдельности каждому из фрагментов - минимальна.

После объединения фрагментов в одно расписание улучшить его не возможно. Действительно, единственным средством его улучшения мог бы стать перенос старта операций из стартовых интервалов одного фрагмента в стартовые интервалы другого.

Перенести из интервалов первого фрагмента в интервалы второго можно только старт тех операций, которые стартуют в последнем такте $kf1$-го стартового интервала и входят в группы списка $A$ с номерами больше $kf1$. Но это изменение расписания не уменьшит длительность ни одного стартового интервала. Перенос старта остальных операций, стартующих в интервалах первого фрагмента, противоречит определению стартового интервала. Согласно этому определению, ни одна операция не может стартовать в стартовом интервале с номером больше номера ее группы списка $A$.

Перенос же операций в обратном направлении только ухудшит расписание, так как после него длительность каждого интервала-отправителя не уменьшится, ибо она и так минимальна, а длительность каждого интервала-получателя может только возрасти. □

**Замечание 2.** *С несколько более жестким условием теорема 2 имеет место и в задаче с одним процессором, рассмотренной в [3]. Она требует, чтобы расписание «совсем» распадалось на два фрагмента, а именно: чтобы по оптимальному расписанию ни одна операция из группы списка $A$ с номером больше $kf1$ не стартовала в стартовых интервалах с номерами не больше $kf1$.*

*В доказательстве теоремы 2 не используется условие $C \in C_T$. Поэтому критерий оптимальности расписания, высказанный в этой теореме, как и в [3], справедлив для любого набора выражений.*

Расписание s1 на рис.2 в нашем примере не оптимально. По нему интервал вычисления состоит из 12-и тактов и содержит 8 ненасыщенных тактов. Обе эти величины можно уменьшить. По расписанию $s2$ на рис.3 интервал вычисления состоит из 10-и тактов и содержит 5 ненасыщенных тактов. Это расписание (даже

"совсем") распадается на два фрагмента, из которых первому принадлежит 1-й переполненный интервал ($kf1 = 1$), а остальные три, все длительностью три такта, — второму. Расписание $s2$ улучшить уже нельзя.

Расписание $S \in S_n$ будем называть каноническим расписанием и множество таких расписаний обозначать через $S_c$, если по нему любые две операции $c(p)$ и $p$ стартуют либо в разных стартовых интервалах либо в одном, но переполненном интервале, причем операция $p$ стартует в последнем такте этого интервала.

```
L y,r1;
L w,r2;

L x,r3;      L v,r5;     M r1,r2,r1;
L u,r4;      L a,r6;     L b,r7;

NOP;         M r4,r5,r4; A r1,r3,r1;
NOP;         NOP;        A r6,r7,r6;

NOP;         NOP;        S r1,r4,r1;
NOP;         NOP;        NOP;
```

*Рис. 3.* Расписание s2.

В следующем разделе будет показано, что предлагаемый нами алгоритм строит каноническое расписание. В доказательстве оптимальности этого расписания используется следующая теорема, которая справедлива и в условиях одно-процессорной Вычислительной Системы. Но доказательство ее в данном случае несколько сложнее. Поэтому мы здесь его приводим, но в сокращенном виде.

**Теорема 3.** *Пусть $C \in C_T$, и $S \in S_c$. Пусть $k$ и $(k+1)$ - номера двух стартовых интервалов, причем по расписанию $S$ $k$-й интервал — неполный, а $(k+1)$-й интервал — полный или переполненный. Тогда в $(k+1)$-ом интервале стартует, по меньшей мере, одна операция, которая готова стартовать и в $k$-ом интервале.*

*Доказательство.* Обозначим через $m'$ число стартующих в $(k+1)$-ом интервале листовых операций и таких нелистовых операций $p$, что операции $c(p)$ стартуют до начала этого интервала. Сначала

покажем, что, каким бы ни был $(k+1)$-й интервал (полным или переполненным), справедливо неравенство $m(S,k) < m'$.

Так как $k$-й интервал — неполный, то $m(S,k) < t * Q$.

Если $(k+1)$-й интервал — полный, то у каждой стартующей в нем нелистовой операции $p$ ее операция $c(p)$ стартует до начала этого интервала, так как $S \in S_c$. Следовательно, в этом случае $m' = m(S,k+1) = t*Q$, а это значит, что неравенство выполняется.

Пусть $(k+1)$-й интервал — переполненный. Так как $S \in S_c$, то в число $m'$ операций входят все операции, стартующие в первых $T(S,k+1)-1$ тактах, и по меньшей мере, одна операция $k+1$-ой группы списка $A$, стартующая в последнем такте этого интервала. Поэтому $m' > (T(S,k+1)-1)Q \geq t*Q$, и следовательно, неравенство справедливо и в этом случае.

Далее в точности так же, как в [3], мы приходим к выводу, что в $(k+1)$-ом интервале стартуют $m'-m(s,k)$ операций, старт которых может быть перенесен в $k$-й интервал. □

**Следствие** *(из теоремы 3). Пусть в условиях теоремы 3 в $(k+1)$-ом стартовом интервале стартуют $n > t*Q$ операций из $(k+1)$-ой группы списка $A$. Тогда среди них найдется операция, готовая стартовать в $k$-ом стартовом интервале.*

*Доказательство.* Для того, чтобы операция $p$, стартующая в $(k+1)$-ом интервале, была готова стартовать в $k$-ом, надо, чтобы операция $c(p)$ стартовала до начала $k$-го интервала.

Пусть $m$ — число операций $(k+1)$-ой группы списка $A$, стартующих в $(k+1)$-ом интервале, чьи операции $c(p)$ стартуют в $k$-ом интервале. Так как $m(S,k) < t*Q$, и $C \in C_T$, то $m < t*Q$. По условиям следствия $n > t*Q$. Следовательно, $n-m > 0$ операций $(k+1)$-ой группы, стартующие в $(k+1)$-ом интервале, — либо листовые, либо их операции $c(p)$ стартуют до начала $k$-го интервала, и потому каждая из них готова стартовать в $k$-ом интервале. □

## 2. Алгоритм построения расписания вычисления набора выражений $C \in C_T$

Следующий очень простой алгоритм (далее алгоритм $S$, см. рис. 4), имеющий полиномиальную сложность, строит расписание вычисления набора выражений, не содержащих общих подвыражений, которое, как мы далее покажем, является оптимальным по времени.

В процессе работы алгоритма список $A$ корректируется. Поэтому текущим списком в описании алгоритма мы будем называть результат его последней коррекции и обозначать этот текущий список через $A_c$. Начальным состоянием списка $A_c$ является список $A$.

Будем предполагать, что все такты интервала вычисления пронумерованы числами $1, 2, \ldots$.

Обозначим расписание, построенное алгоритмом $S$ через $S_a$ и покажем, что оно оптимально.

Будем предполагать, что алгоритм $S$ просматривает список $A_c$, начиная всегда с первого элемента и в качестве операции $p$ выбирает первую в этом списке подходящую операцию.

```
   S(A)
{
   Ac= A;
   for v= 1 while Ac ≠ 0 do
      for q= 1 while q <= Q do
         if в Ac есть операция p, готовая стартовать в такте v
        then begin
            назначить старт операции p в такте v на процессоре q;
           удалить операцию p из списка Ac
        end
        else begin
        считать такт v ненасыщенным;
        выйти из цикла по q
        end
        q=q+1
     done
     v=v+1
  done
}
```

*Рис. 4.* Алгоритм построения расписания.

Через $N(k)$ обозначим множество операций $k$-ой группы списка $A$, которые входят в список $A_c$ перед началом поиска операции, готовой к выполнению в первом такте $k$-го стартового интервала. Через $n(k)$ обозначим число этих операций.

Перед началом формирования $k$-го стартового интервала операции множества $N(k)$ занимают первые места в списке $A_c$. Поэтому в течение формирования этого интервала алгоритм просматривает

их в первую очередь. Найденные при просмотре операции он удаляет из списка. Процесс формирования интервала заканчивается с удалением последней из них.

Покажем сначала, что расписание $S_a$ — нормализованное. Для этого докажем две следующие леммы.

Свойство, о котором говорится в первой из этих лемм, характеризует полные и неполные интервалы и в работе одно-процессорной системы. Доказательство этой леммы почти слово в слово повторяет ее доказательство в [3]. Поэтому здесь мы его опускаем.

**Лемма 1.** Если $1 < kk_{max}$ и $n(k)t * Q$, то по расписанию $S_a$ длительность $k$-го стартового интервала равна $t$ тактам.

**Лемма 2.** Если $1 < kk_{max}$ и $n(k) > t * Q$, то по расписанию $S_a$ первые $T(S_a, k) - 1$ тактов $k$-го стартового интервала являются насыщенными тактами, и в каждом из них стартуют операции только $k$-ой группы списка $A$.

*Доказательство.* Из определения стартового интервала следует, что все операции множества $N(k)$, должны стартовать в этом интервале.

Выделим в множестве $N(k)$ подмножество $N1$ операций, у каждой из которых такт $\theta_1$ совпадает с одним из первых $t$ тактов $k$-го стартового интервала. Остальные операции (обозначим их множество через $N2$) становятся готовыми к выполнению до начала $k$-го интервала.

Таким образом, в каждом из $t$ первых тактов $k$-го интервала готовы стартовать все операции множества $N2$, а в некоторых из них еще и операции множества $N1$, причем в количестве не более $Q$ штук.

Поскольку $|N1| + |N2| = n(k) > t * Q$, и все эти операции становятся готовыми к выполнению не позднее $t$-го такта $k$-го стартового интервала, то в каждом из первых $t$ тактов интервала в множестве $N(k)$ найдутся $Q$ операций, которые готовы в нем стартовать. А это значит, что эти такты станут насыщенными тактами.

Все остальные $n(k) - t * Q$ операций готовы стартовать в каждом из следующих тактов интервала. Поэтому и из этих тактов каждый, кроме может быть последнего, так же станет насыщенным тактом. Для "насыщения"последнего такта операций множества $N(k)$ будет недостаточно, если $n(k)$ не кратно $Q$.

В течение формирования $k$-го стартового интервала еще не удаленные из списка $A_c$ операции множества $N(k)$ остаются в нем на

первых местах. Поэтому алгоритм $S$ будет находить именно в этом множестве для каждого очередного такта $k$-го интервала, за исключением, возможно, его последнего такта, по $Q$ операций, готовых в нем стартовать, пока не получат стартовые такты все операции множества $N(k)$. В последнем такте $k$-го интервала могут стартовать меньше $Q$ операций $k$-ой группы списка $A$, если $n(k)$ не кратно $Q$. □

**Замечание 3.** *По определению стартового интервала в последнем такте $k$-го интервала должна стартовать хотя бы одна операция $k$-ой группы списка $A$. Но в условиях леммы 2, кроме них, в этом такте могут стартовать и операции из групп списка $A$ с номерами больше $k$. Но и тогда этот такт может оказаться ненасыщенным.*

**Следствие** *(из леммы 2). Если $1 < kk_{max}$ и $n(k) > t * Q$, то по расписанию $S_a$ стартовый интервал с номером $k$ является переполненным интервалом.*

*Доказательство.* Так как $n(k) > t*Q$, и все операции множества $N(k)$ по расписанию $S_a$ стартуют в $k$-ом интервале, то $m(S_a, k) > t * Q$.

Так как, согласно лемме 2, первые $T(S_a, k) - 1$ тактов этого интервала по расписанию $S_a$ — насыщенные, а последний его такт, согласно замечанию 3, может оказаться ненасыщенным, то $T(S_a, k) = \lceil m(S_a, k)/Q \rceil$.

Стартовый интервал с такими свойствами является переполненным интервалом. □

**Замечание 4.** *Все операции 1-ой группы списка $A$ являются листовыми операциями и перед началом формирования 1-го стартового интервала в списке $A_c$ занимают первые места. Поэтому по расписанию $S_a$ при $T(S_a, 1) > 1$ $T(S_a, 1) - 1$ тактов этого интервала являются насыщенными тактами, и в каждом из них стартуют операции только 1-ой группы списка $A$. Напомним, что 1-й стартовый интервал является по определению переполненным интервалом независимо от числа стартующих в нем операций.*

Из леммы 1, следствия из леммы 2 и замечания 4 следует, что расписание $S_a$ — нормализованное.

**Лемма 3.** *Расписание $S_a$ — каноническое.*

*Доказательство.* Так как длительность полных и неполных по расписанию $S_a$ интервалов равна $t$ тактам, то старт в любом из них

обеих операций $c(p)$ и $p$ не возможен.

Если $k$-й стартовый интервал по расписанию $S_a$ — переполненный, то, согласно лемме 2, во всех его тактах, кроме, может быть, последнего, стартуют только операции $k$-ой группы списка $A$. Поэтому, если в этом интервале и стартуют обе операции хотя бы одной пары $c(p), p$, то операция $p$ стартует в его последнем такте, а это допускается определением канонического расписания. □

Покажем теперь, что расписания $S_a$ удовлетворяет условиям теоремы 2.

Рассмотрим подробнее неполные стартовые интервалы по расписанию $S_a$. Пусть $k$ — номер одного из таких интервалов.

Пусть по расписанию $S_a$ $\tau 0$ — номер первого в $k$-ом интервале ненасыщенного, возможно даже пустого, такта. Мы покажем, что если такт $\tau 0$ не является его последним тактом, то операции, стартующие в следующих тактах $k$-го интервала, стартуют в первых тактах готовности к выполнению.

Обозначим через $A_c(\tau, q)$ состояние списка $A_c$ перед началом поиска алгоритмом $S$ операции, готовой стартовать в такте $\tau$ на $q$-ом процессоре.

Индукцией по тактам $\tau > \tau 0$ можно обосновать следующее замечание.

**Замечание 5.** *Пусть $\tau > \tau 0$ — номер непустого такта в $k$-ом интервале. Тогда на шагах алгоритма $S$, на которых переменная $v$ принимает значение $\tau$, а переменная $q$ — значения $1, 2, \ldots, q(\tau)$, список $A_c(\tau, q)$ содержит только операции $p$ с тактами $\theta_1(p) \geq \tau$.*

В самом деле, если бы этот список содержал операции с тактами $\theta_1 < \tau$, то это бы означало, что алгоритм $S$ мог назначить эти операции на старт в тактах интервала $[\tau 0 : \tau - 1]$, но этого не сделал, что, как следует из его описания, не возможно.

**Замечание 6.** *Все операции, стартующие в $k$-ом интервале после такта $\tau 0$, стартуют в первых тактах своей готовности к выполнению.*

Действительно, согласно замечанию 5, когда $v > \tau 0$ список $A_c$ содержит только операции с тактами $\theta_1 \geq v$. В этом случае в текущий пакет алгоритм может включить только операции, у которых $\theta_1 = v$.

Обозначим через $k_{01}$ номер последнего по расписанию $S_a$ переполненного стартового интервала.

Согласно лемме 3, расписание $S_a$ — каноническое. Доказательство двух следующих лемм, то есть лемм 4 и 5, основано на теореме 3, следствии из нее и замечании 6. По смыслу эти леммы почти совпадают с утверждениями, доказанными в [3] (леммы 5 и 6 в [3]). Отличие состоит лишь в том, что утверждение нашей леммы 5, формально более жесткое, чем в лемме 6 из [3], не распространяется на последние такты переполненных интервалов. Доказательства обеих лемм в обоих случаях практически совпадают. Поэтому здесь мы их опускаем.

**Лемма 4.** *Каждый не переполненный по расписанию $S_a$ интервал с номером меньше $k_{01}$ может быть только полным интервалом.*

**Лемма 5.** *Пусть $k1$ и $k2 > k1$ — номера двух стартовых интервалов, причем по расписанию $S_a$ $k1$-й интервал — полный, а $k2$-й — ближайший к нему переполненный интервал. Тогда $k1$-й интервал может содержать стартовые такты операций из групп списка $A$ только с номерами $k1, k1+1, \ldots, k2$.*

Итак, из лемм 2 и 4 следует, что в стартовых интервалах с номерами не больше $k_{01}$ каждый такт, кроме последних тактов переполненных интервалов, по расписанию $S_a$ является насыщенным тактом. Следующая лемма распространяет свойство насыщенности и на последние такты переполненных интервалов с номерами меньше $k_{01}$.

**Лемма 6.** *Пусть $k_{01} > 1$. Последний такт каждого переполненного по расписанию $S_a$ интервала с номером $k < k_{01}$ является насыщенным тактом.*

*Доказательство.* Обозначим через $\tau$ и через $q$ соответственно номер последнего такта $k$-го интервала и число операций $k$-ой группы списка $A$, стартующих в этом такте. Очевидно $q > 0$, так как, согласно замечанию 3, в последнем такте каждого стартового интервала должна стартовать хотя бы одна операция из его группы. Лемма утверждает, что такт $\tau$ — насыщенный.

Если $q = Q$, то такт $\tau$ — действительно насыщенный, и лемма доказана.

Пусть $q < Q$. В этом случае для насыщения такта $\tau$ в нем должны стартовать $Q - q$ операций из групп списка $A$ с номерами больше $k$.

Предположим, вопреки утверждению леммы, что по расписанию $S_a$ такт $\tau$ — ненасыщенный, и в нем стартуют только $q$ опера-

ций $k$-ой группы списка $A$. Такое расписание $S_a$ обозначим через $S_a^*$. Для расписания $S_a^*$ справедливы утверждения всех предыдущих лемм.

Покажем, что среди операций, стартующих по расписанию $S_a^*$ в $(k+1)$-ом стартовом интервале, найдутся $Q-q$ операций, которые готовы стартовать и в такте $\tau$, и что при формировании пакета операций, стартующих в этом такте, алгоритм $S$ эти операции просматривал и, следовательно, должен был предоставить им для старта именно этот такт. Данное противоречие и будет доказывать лемму.

Обозначим через $M(k+1)$ множество всех стартующих в $(k+1)$-ом интервале операций, если этот интервал по расписанию $S_a^*$ полный, или множество стартующих в нем операций только $(k+1)$-ой группы списка $A$, если он по этому расписанию переполненный. Упомянутые $Q-q$ операций мы будем искать именно в множестве $M(k+1)$.

Если $(k+1)$-й интервал — полный, то $|M(k+1)| = m(S_a^*, k+1) = t * Q$. Если же этот интервал — переполненный, то из леммы 2 следует неравенство $|M(k+1)| > t * Q$. Таким образом, величина $|M(k+1)|$ удовлетворяет равенству $|M(k+1)| = t * Q + d$ при $d \geq 0$.

Через $M1$ и $M2$ обозначим множества соответственно всех нелистовых и всех листовых операций, входящих в множество $M(k+1)$.

Очевидно, $|M1|+|M2| = |M(k+1)|$. Так как $|M(k+1)| = t * Q + d$, то $|M1| + |M2| = t * Q + d$. Из этого равенства получаем

$$|M1| = t * Q + d - |M2|. \tag{27}$$

Каждая операция из множества $M2$ готова стартовать в любом такте $k$-го интервала.

Так как в каждом стартовом интервале с номером больше 1 должна стартовать, по меньшей мере, одна нелистовая операция, то $|M1| > 0$.

Для того, чтобы операция $p \in M1$ была готова стартовать в такте $\tau$ необходимо и достаточно, чтобы операция $c(p)$ стартовала не позднее такта $\tau_c = \tau - t$. Так как $k$-й интервал — переполненный, то такт $\tau_c$ принадлежит этому интервалу.

Далее будем предполагать, что порядок старта операций в $k$-ом интервале по расписанию $S_a^*$ таков, что операции, завершающие

подготовку к выполнению операций из множества $M1$, стартуют в его последних тактах. Обозначим этот порядок через $P_o$.

Если при порядке старта операций $P_o$ в $k$-ом интервале найдется хотя бы одна операция $c(p)$, стартующая не позднее такта $\tau_c$, и такая, что $p \in M1$, то не меньше операций $c(p)$ с такими свойствами найдется и при любом другом порядке старта операций в этом интервале. Иными словами, если мы докажем, что лемма верна при порядке старта операций $P_o$, то она останется верной и при любом другом порядке их старта в $k$-ом интервале.

Через $M3$ обозначим множество таких операций $p \in M1$, что операции $c(p)$ стартуют не позднее такта $\tau_c$ при порядке старта $P_o$.

Так как $k$-й интервал — переполненный, то, согласно лемме 2, по расписанию $S_a^*$ в нем стартуют операции только $k$-ой группы списка $A$. Это значит, что результаты всех стартующих в нем операций используются только операциями $(k+1)$-ой группы, стартующими в $(k+1)$-ом интервале.

Для нас не представляют интереса операции, стартующие в $t$ последних тактах $k$-го интервала. Первые $t-1$ из этих тактов — насыщенные, и, значит, в них стартуют $(t-1)Q$, а в последнем — только $q$ операций.

Таким образом, из $|M1|$ операций $c(p)$, таких, что $p \in M1$, не позднее такта $\tau_c$ стартуют только $|M1| - ((t-1)Q+q)$ операций. Исключая из этого выражения с помощью равенства (27) величину $|M1|$, получаем:

$$|M3| = d + Q - q - |M2|.$$

Итак, любая операция множества $M^* = M2 \cup M3$ готова стартовать в такте $\tau$. Их число $|M^*|$ удовлетворяет равенствам:

$$|M^*| = |M2| + |M3| = |M2| + d + Q - q - |M2| = d + Q - q,$$

откуда следует, что $|M^*| \geq Q - q$.

На шагах формирования пакета операций, стартующих в такте $\tau$, все операции множества $M^*$ еще находятся в списке $A_c$. Поэтому алгоритм $S$ выберет среди них $Q-q$ операций, которым предоставит свободные процессоры для старта в такте $\tau$. $\square$

**Лемма 7.** *В последнем такте каждого переполненного по расписанию $S_a$ интервала с номером $k < k_{01}$ стартуют операции из групп списка $A$ с номерами не больше $k_{01}$.*

*Доказательство.* Вернемся к доказательству предыдущей леммы.

Если по расписанию $S_a^*$ $(k+1)$-й стартовый интервал является переполненным интервалом, то в такте $\tau$, последнем такте $k$-го интервала, кроме операций $k$-ой группы списка $A$ стартуют только операции из группы с номером $k+1$, так как множество $M(k+1)$, из которого в доказательстве леммы 6 выбираются эти операции, состоит в данном случае только из операций этой группы. Так как по условиям леммы 7 $k < k_{01}$, то $k+1 k_{01}$, а это в данном случае и есть утверждение леммы.

Пусть теперь $(k+1)$-й интервал является по расписанию $S_a^*$ полным интервалом. В этом случае множество $M(k+1)$ образуют все операции, стартующие в этом интервале.

Обозначим через $k2$ номер ближайшего справа к интервалу с номером $k+1$ переполненного по расписанию $S_a^*$ интервала.

Согласно лемме 5, в $(k+1)$-ом интервале по расписанию $S_a^*$ стартуют операции из групп списка $A$ с номерами не больше $k2$, и, следовательно, множество $M(k+1)$ состоит именно из этих операций. Покажем, что выполняется неравенство $k2 k_{01}$, из которого будет следовать утверждение леммы 7 в рассматриваемом случае.

Так как некоторые операции, например $Q-q$ операций из множества $M(k+1)$, по расписанию $S_a^*$ стартуют позднее, чем по расписанию $S_a$, то [2] каждый переполненный по расписанию $S_a$ интервал является переполненным интервалом и по расписанию $S_a^*$. Поэтому $k_{01}$-й интервал по расписанию $S_a^*$ — также переполненный. Так как $k2$-й интервал по выбору является ближайшим справа к интервалу с номером $k+1$ переполненным интервалом, то $k2 k_{01}$. $\square$

**Теорема 4.** *Расписание $S_a$ оптимально по времени.*

*Доказательство.* Из лемм 2, 5 и 7 следует, что если $k_{01} < k_{max}$, то расписание $S_a$ почти распадается на два фрагмента, из которых первому фрагменту принадлежат первые $k_{01}$ стартовых интервалов.

Если $k_{01} = k_{max}$, то положим, что расписание $S_a$ состоит только из одного фрагмента, который мы называем первым фрагментом.

Из лемм 2, 4 и 6 следует, что все такты интервалов первого фрагмента, за исключением, возможно, последнего такта $k_{01}$-го интервала, являются насыщенными тактами.

[2] В контексте доказательства леммы 6 по расписанию $S_a$ такт $\tau$— насыщенный, а по расписанию $S_a^*$ в нем стартуют $q < Q$ операций.

Так как $k_{01}$-й интервал является последним переполненным интервалом, то если $k_{01} < k_{max}$, стартовые интервалы с номерами больше $k_{01}$, то есть интервалы, принадлежащие второму фрагменту расписания, являются полными и неполными интервалами.

Из всего сказанного следует, что расписание $S_a$ удовлетворяет условиям теоремы 2 и, следовательно, является оптимальным по времени расписанием. □

## Заключение

Итак, убедившись в том, что предложенный нами алгоритм строит в нашей задаче кратчайшее расписание, мы фактически доказали ее полиномиальность. Это означает, что результат, полученный в работе [5] для процессоров с режимом последовательного выполнения операций, справедлив и для процессоров, работающих в конвейерном режиме.

Оценка сложности предложенного нами алгоритма, включая процедуру построения списка $A$, очевидно, не превосходит $O(n^2)$. Здесь $n$ — число вершин в графе $G(C)$.

Действительно, если в качестве процедуры построения списка $A$ использовать алгоритм топологической сортировки, [2], то получим оценку ее сложности, равную $O(e + n)$, где $e$ — число дуг в графе $G(C)$. Но граф $G(C)$ — лес, и потому имеет $n - m$ дуг, где $m$— число его компонент связности, то есть число выражений в наборе $C$. Поэтому получаем, что оценка сложности этой процедуры, равна $O(2n - m)$. Эта оценка мажорируется оценкой $O(n^2)$.

Задача, которая решается в [5], отличается от нашей задачи не только типом процессоров. У нас число процессоров в Вычислительной Системе заранее фиксировано, а в [5] это ограничение отсутствует. Отсутствие такого ограничения означает, что при вычислении любого набора выражений в Системе всегда найдется процессор, на котором очередная операция сможет стартовать в первом такте ее готовности к выполнению.

Если в нашей задаче отказаться от ограничения числа процессоров в Системе, то в качестве кратчайшего расписания $S_a$ можно было бы взять список $A$. По такому расписанию все операции каждой группы списка $A$ стартовали бы в последнем такте своего стартового интервала, оставляя остальные его такты пустыми, и длительность интервала вычисления была бы равна $t*k_{max}$ тактам. Это значило бы, что все участвующие в вычислении набора выра-

жений процессоры работают параллельно и при этом имитируют последовательный режим выполнения операций.

Как мы сказали в начале статьи, рассматриваемая здесь задача является обобщением задачи, решенной в [3]. Дальнейшими обобщениями могут быть возможность вычислять наборы выражений, информационные связи в которых описывают орграфы, а также отказ от требования равенства длительностей выполнения операций и введение ограничений на время начала выполнения листовых операций.

Любое из этих обобщений требует изменения некоторых условий задачи. Но именно эти условия обеспечивают существование кратчайшего расписания, свойства которого описываются в данной статье теоремой 2. Поэтому исследование этих обобщений лежит уже за пределами возможностей модели расписания, которая использовалась в [3] и в данной статье. Однако мы надеемся, что полученные здесь результаты будут полезны и в этом исследовании.

## Список литературы


[1] *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи / Пер. с англ. Е. В. Левнера и М. А. Фрумкина под ред. А. А. Фридмана. М.: Мир, 1982. 415 с.

[2] *Кнут Д. Е.* Искусство программирования. Т. 1: Основные алгоритмы. 3-е изд. / Пер. с англ.; под общ. ред. Ю. В. Козаченко. Издательский дом «Вильямс». 2004. 712 с.

[3] *Сурин С. С.* Оптимальный алгоритм локального планирования инструкций при отсутствии структурных конфликтов // Системное программирование. Вып. 1. СПб.: Изд-во СПбГУ, 2005. С. 131–146.

[4] Теория расписаний и вычислительные машины / Под ред. Э. Г. Коффмана; Пер. с англ. В. М. Амочкина под ред. Б. А. Головкина. М.: Наука, 1984. 334 с.

[5] *Ху Т. С.* Параллельное упорядочивание и проблема линии сборки // Кибернетический сборник. Новая серия. Вып. 4: Сборник переводов. М.: Мир, 1967. С. 43–56.

[6] *Bernstein D., Gertner I.* Scheduling expressions on a pipelined processor with a maximal delay of one cycle // ACM Trans. Program. Lang. Syst. 11, 1. Jan. 1989. P. 57–66.

[7] *Coffman E. G., Jr and Graham R. L.* Optimal Scheduling for Two ProcessorSystems // Acta Informatica 1. 3. 1972. P. 200–213.

[8] *Gabow H. N.* An Almost-Linear Algorithm for Two-Processor Scheduling // J. ACM 29 3 Jul. 1982. P. 766–780.

[9] *Jakoby and A. and Reischuk R.* Scheduling trees with communication delays. http://citeseer.ist.psu.edu/348772.html.

[10] *Jung H., Kirousis L. and Spirakis P.* Lower bounds and efficient algorithms for multiprocessor sceduling of DAGs with communication delays // Proc. 1. Synposium on parallel algorithms and architectures. 1989. P. 254–264.

[11] *Lam S. and Sethi R.* Worst case analysis of two sceduling algorithm // SIAM J Comput 6. 1977. P. 518–536.

[12] *Papadimitriou C., Yannakakis M.* Towards an architecture-independent analysis of parallel algoritms // Proc. 20. ACM symposium on Theory of computing. 1988. P. 510–513; см. также: SIAM J. Comput. 19. 1990. P. 322–328.

[13] *Sethi R.* Sceduleng graphs on two processors // SIAM J. Comput 1. 1976. P. 73–82.

[14] *Sethi R., Ullman J. D.* The Generation of Optimal Code for Arithmetic Expressions // J. ACM 17, 4. Oct. 1970. P. 715–728.

[15] *Tarjan R.* E Efficiency of a good but not linear set union algorithm // J ACM 22. 2. Apr. 1975. P. 215–225.

[16] *Ulman J. D.* Polinomial complete schduling problems // Operating systems reviev 7. 4. 1973. P. 96–101.