Автоматика и телемеханика, № 2, 2005

Автоматизированные системы управления

© 2005 г. Д. С. ГУЗ
(Московский физико-технический институт),
М. Г. ФУРУГЯН, канд. физ.-мат. наук
(Вычислительный центр им. А.А. Дородницына РАН, Москва)

ПЛАНИРОВАНИЕ ВЫЧИСЛЕНИЙ В МНОГОПРОЦЕССОРНЫХ АСУ РЕАЛЬНОГО ВРЕМЕНИ С ОГРАНИЧЕНИЯМИ НА ПАМЯТЬ ПРОЦЕССОРОВ

Рассматривается задача построения допустимых расписаний в АСУ жесткого реального времени при заданных директивных сроках выполнения работ. В отличие от [1–3] дополнительно учитываются ограничения на память процессоров. Разработаны два метода решения поставленной задачи. Первый основан на сведении исходной задачи к поиску многопродуктового потока в сети специального вида, второй предлагает быстрый алгоритм нахождения допустимого расписания для однопроцессорного случая.

1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, работа которых представляется как набор последовательных тактов. Такты всех процессоров синхронизованы во времени, т.е. их начала и концы для всех процессоров совпадают. Процессор j ($j = 1, \ldots, m$) характеризуется производительностью s_j и объемом памяти V_j. Имеется n независимых друг от друга работ, каждая работа i ($i = 1, \ldots, n$) характеризуется объемом p_i, директивным интервалом исполнения $[r_i, d_i]$ и объемом памяти v_i, которую необходимо загрузить в процессоре для выполнения работы.

Без ограничения общности можно считать, что все r_i, d_i – натуральные числа (т.е. начала и концы всех директивных интервалов задают номера соответствующих тактов работы процессоров) и $\min\limits_i r_i = 0$, $\max\limits_i d_i = T$ (т.е. в задаче рассматривается T тактов работы процессоров). Таким образом, s_j – это объем работы, выполняемой процессором j за один такт, т.е. для того, чтобы полностью выполнить работу i, процессору j необходимо затратить $\lceil p_i/s_j \rceil$ тактов. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. Предполагается, что прерывание выполнения работы на процессоре j_1 с целью ее переключения на одном из последующих тактов на процессор j_2 или для возобновления на этом же процессоре ($j_2 = j_1$) требует дополнительной работы процессоров j_1 и j_2 в суммарном объеме $\tau_{j_1 j_2}$.

Связи между процессорами могут меняться во времени, что определяется массивом I размером $m \times T \times m \times T$. При этом $I(j_1, k_1, j_2, k_2) = 1$, если при выполнении работы на процессоре j_1 в конце такта k_1 ее можно прервать и возобновить на процессоре j_2 в начале такта k_2, и $I(j_1, k_1, j_2, k_2) = 0$, если такое переключение невозможно $(1 \le j_1,\ j_2 \le m,\ 1 \le k_1 < k_2 \le T)$.

Необходимо построить такое расписание, при котором все работы выполняются на процессорах в своих директивных интервалах, или показать, что такого расписания не существует.

2. Построение сети

Для решения этой задачи построим многопродуктовую потоковую сеть $N = (V, E)$, где V – вершины, E – дуги. Множество V состоит из следующих вершин: $u_1, u_2, \ldots$ $\ldots, u_n$ – источники; $w_1, w_2, \ldots, w_n$ – стоки; x_j^k, y_j^k $(j = 1, \ldots, m;\ k = 1, \ldots, T)$ – внутренние вершины (рис. 1). Множество E состоит из следующих дуг: $\left(u_i, x_j^k\right)$ $(i = 1, \ldots, n;\ j = 1, \ldots, m;\ k \in [r_i, d_i])$; $\left(x_j^k, y_j^k\right)$ $(j = 1, \ldots, m;\ k = 1, \ldots, T)$; $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$ для всех $j_1, j_2 = 1, \ldots, m$ и k_1, k_2 таких, что $k_1 < k_2$, $I(j_1, k_1, j_2, k_2) = 1$; $\left(y_j^k, w_i\right)$ $(i = 1, \ldots, n;\ j = 1, \ldots, m;\ k \in [r_i, d_i])$. Для сети, изображенной на рис. 1, наличие дуг $\left(u_i, x_j^k\right)$ и $\left(u_i, x_j^{k+1}\right)$ означает, что работу i можно начать выполнять на процессоре j как на k-м, так и на $(k+1)$-м такте, а наличие дуг $\left(y_j^k, w_i\right)$ и $\left(y_j^{k+1}, w_i\right)$ означает, что завершить работу i можно как в конце k-го, так и в конце $(k+1)$-го такта. Наличие дуги $\left(y_j^k, x_r^{k+1}\right)$ означает, что выполнение работы на процессоре j в конце k-го такта можно прервать и возобновить на процессоре r в начале $(k+1)$-го такта. Несложно заметить, что $|V| = 2n + 2mT$, $|E| \le mT\left(\frac{1}{2}m(T-1) + 2n + 1\right)$. Каждая дуга сети N имеет два параметра: пропускную способность и стоимость единицы потока по дуге. Пропускные способности всех дуг полагаем равными 1. Стоимость прохождения единицы потока по дуге $\left(x_j^k, y_j^k\right)$ полагаем равной s_j, по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$ – равной $-\tau_{j_1 j_2}$, а по дугам $\left(u_i, x_j^k\right)$ и $\left(y_j^k, w_i\right)$ – равной 0.

Рассмотрим в этой сети многопродуктовый поток, состоящий из n типов потоков, причем поток i-го продукта исходит из источника u_i и входит в сток w_i $(i = 1, \ldots, n)$. Эти потоки описывают выполнение работ многопроцессорной системой. Будем интерпретировать поток i-го продукта по дугам сети N следующим образом: по ду-

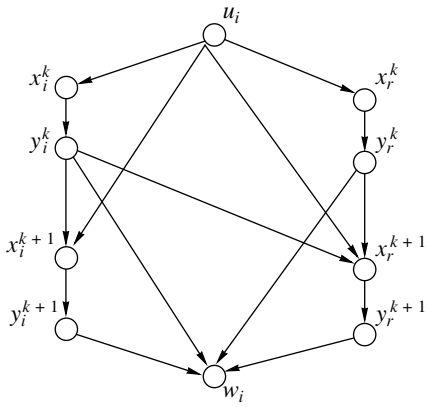

Рис. 1. Фрагмент сети N.

ге (u_i, x_j^k) – начало выполнения работы i (на такте k); по дуге $\left(x_j^k y_j^k\right)$ – выполнение работы i процессором j на такте k; по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$ – прерывание выполнения работы i процессором j_1 на такте k_1 и ее переключение на процессор j_2 на такте k_2; по дуге $\left(y_j^k, w_i\right)$ – конец выполнения работы i (на такте k).

При прохождении потока i-го продукта по дуге $\left(x_j^k, y_j^k\right)$ к его стоимости прибавляется величина, равная s_j (объем выполненной работы на j-м процессоре за один такт). При прохождении потока по дуге $\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right)$, которая соответствует прерыванию или переключению работы, из его стоимости вычитается величина $\tau_{j_1 j_2}$. При прохождении потока по дугам $\left(u_i, x_j^k\right)$ и $\left(y_j^k, w_i\right)$ его стоимость остается без изменений. Будем обозначать поток i-го продукта по дуге (a, b) как $f^i(a, b)$. Заметим, что в нашей модели $f^i(a, b) \in \{0, 1\}$ при всех a, b и i. Например, если для сети, изображенной на рис. 1, $f^i\left(x_{j_1}^k, y_{j_1}^k\right) = f^i\left(y_{j_1}^k, x_{j_2}^{k+1}\right) = f^i\left(x_{j_2}^{k+1}, y_{j_2}^{k+1}\right) = 1$, то на k-м и $(k+1)$-м тактах суммарный объем работы процессоров j_1 и j_2 по выполнению работы i составляет $s_{j_1} + s_{j_2} - \tau_{j_1 j_2}$.

3. Необходимые и достаточные условия существования допустимого расписания

Лемма. Допустимое расписание существует тогда и только тогда, когда в сети N существует n-продуктовый поток, обладающий следующими свойствами:

$$(1) \qquad \sum_{j,k} f^i\left(x_j^k, y_j^k\right) s_j - \sum_{j_1, j_2, k_1, k_2; k_1 < k_2} f^i\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right) \tau_{j_1 j_2} \geq p_i$$

при всех $i = 1, \ldots, n$, т.е. суммарная стоимость потока i-го продукта составляет не менее p_i;

$$(2) \qquad \sum_{i, r_1, r_2; r_1 < k < r_2} f^i\left(y_j^{r_1}, x_j^{r_2}\right) v_i + \sum_{i=1}^{n} f^i\left(x_j^k, y_j^k\right) v_i \leq V_j$$

при всех $j = 1, \ldots, m$; $k = 1, \ldots, T$, т.е. на каждом такте выполнено ограничение по объему памяти каждого процессора.

Доказательство. 1. Предположим, что допустимое расписание существует. Будем задавать его матрицей A размера $m \times T$, каждый элемент a_{jk} которой может принимать целые значения в диапазоне от 0 до n, причем если $a_{jk} = i$ $(1 \leq i \leq n)$, то это означает, что на k-м такте работа i выполняется процессором j, а если $a_{jk} = 0$, то на k-м такте процессор j простаивает. Отметим, что в каждом столбце матрицы A значение только одного элемента может быть равным i, поскольку каждая работа может одновременно выполняться только на одном процессоре. Для каждой работы i выполним следующие действия. Пусть $k_0 < k_1 < \cdots < k_l$ – номера всех столбцов матрицы A, в каждом из которых есть элемент, равный i, и пусть $a_{j_0 k_0} = a_{j_1 k_1} = \cdots = a_{j_l k_l} = i$. Тогда полагаем значения потоков $f^i\left(u_i, x_{j_0}^{k_0}\right)$, $f^i\left(x_{j_0}^{k_0}, y_{j_0}^{k_0}\right)$, $f^i\left(y_{j_0}^{k_0}, x_{j_1}^{k_1}\right)$, $f^i\left(x_{j_1}^{k_1}, y_{j_1}^{k_1}\right)$, $\ldots$, $f^i(x_{j_{l-1}}^{k_{l-1}}, y_{j_l}^{k_l})$, $f^i(x_{j_l}^{k_l}, y_{j_l}^{k_l})$, $f^i\left(y_{j_l}^{k_l}, w_i\right)$ равными 1. Потоки i-го продукта по всем остальным дугам полагаем равными 0.

Поскольку согласно допустимому расписанию каждая работа должна быть выполнена полностью, то неравенство (1) выполнено. Выполнение неравенства (2) следует из того, что допустимое расписание должно также удовлетворять ограничениям по памяти процессоров. Действительно, первое слагаемое в (2) задает поток по всем дугам, которые имеют своим началом вершины, относящиеся к процессору j

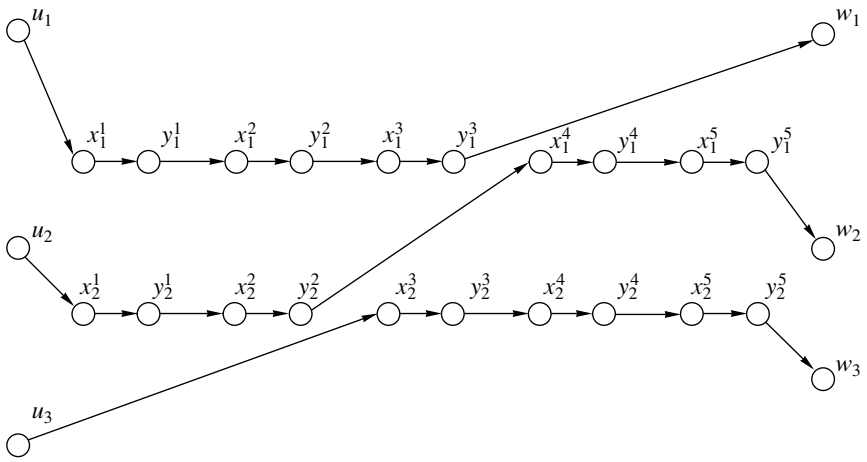

Рис. 2. Потоки в сети N, отображающие допустимое расписание для рассматриваемого примера.

на всех тактах, предшествующих текущему такту k, и окончание в вершинах, относящихся к этому же процессору j и какому-либо последующему такту. Ненулевой поток какого-либо i-го продукта по одному из подобных ребер означает, что задача i выполнялась на этом процессоре, затем была отложена и позднее возобновлена на нем же без переключения на другие процессоры, т.е. во время рассматриваемого такта она хотя и не выполняется, но находится в памяти j-го процессора и занимает объем v_i. Второе же слагаемое соответствует собственно выполнению некоторой работы процессором j на k-м такте. Поэтому сумма указанных двух величин не превышает V_j.

2. Предположим теперь, что в сети N существует многопродуктовый поток, удовлетворяющий условиям (1) и (2). Построим по этому потоку допустимое расписание путем заполнения матрицы A. Заметим, что каждому элементу a_{jk} матрицы A соответствует ребро $\left(x_j^k, y_j^k\right)$ сети N. Поэтому матрица A может быть заполнена по следующему простому правилу: $a_{jk} = i$, если $f^i\left(x_j^k, y_j^k\right) = 1$, и $a_{jk} = 0$, если $f_i\left(x_j^k, y_j^k\right) = 0$. Действительно, именно положительный поток i-го продукта по ребру $\left(x_j^k, y_j^k\right)$ соответствует выполнению работы i процессором j на k-м такте, и никаких дополнительных данных для задания расписания не требуется. В силу (1) все работы полностью выполняются, а структура сети N гарантирует, что каждая работа выполняется строго в своем директивном интервале. В силу условия (2) объем памяти каждого процессора на каждом такте при выполнении работ не будет превышен. Следовательно, построенное расписание является допустимым. Лемма доказана.

Рассмотрим пример, в котором $T = 5$; $n = 3$; $m = 2$; $s_1 = s_2 = 1$; $p_1 = p_2 = p_3 = 3$; $r_1 = r_2 = r_3 = 0$; $d_1 = d_2 = d_3 = 5$; $\tau_{12} = \tau_{21} = 1$; $I(j_1, k_1, j_2, k_2) = 1$ при всех $1 \le j_1, j_2 \le 2$; $1 \le k_1 < k_2 \le 5$; $v_1 = v_2 = v_3 = 1$; $V_1 = V_2 = 1$. На рис. 2 изображен фрагмент сети N для данного примера, включающий только те дуги, поток по которым равен 1.

Поток первого продукта проходит по пути $u_1 - x_1^1 - y_1^1 - x_1^2 - y_1^2 - x_1^3 - y_1^3 - w_1$, и его стоимость равна 3, т.е. первая работа выполняется первым процессором на первых трех тактах и получает три единицы процессорного времени. Поток второго продукта проходит по пути $u_2 - x_2^1 - y_2^1 - x_2^2 - y_2^2 - x_1^4 - y_1^4 - x_1^5 - y_1^5 - w_2$, и его стоимость равна $2 - 1 + 2 = 3$, т.е. вторая работа выполняется первые два такта

вторым процессором, затем переключается на первый процессор, на котором выполняется четвертый и пятый такты. Суммарный объем работы рассматриваемой двухпроцессорной системы по выполнению второй работы равен 4 единицам процессорного времени, однако одна из этих единиц расходуется на переключение и на выполнение работы остаются необходимые 3 единицы. Поток третьего продукта проходит по пути $u_3 - x_2^3 - y_2^3 - x_2^4 - y_2^4 - x_2^5 - y_2^5 - w_3$, и его стоимость равна 3, т.е. третья работа выполняется вторым процессором на третьем, четвертом и пятом тактах и также получает три единицы процессорного времени. Таким образом, условие 1 леммы выполнено. Нетрудно заметить, что первое слагаемое в левой части условия 2 леммы всегда равно 0, а второе равно 1, поэтому условие 2 также выполняется. Таким образом, допустимое расписание существует и определяется матрицей $A = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 3 & 3 & 3 \end{pmatrix}$.

Итак, показано, что предлагаемое сведение задачи о поиске многопроцессорного расписания к задаче о поиске многопродуктового потока в сети N корректно, и однозначно интерпретирована одна задача в терминах другой. Поскольку поток по каждой дуге равен 0 или 1, то для нахождения искомого потока можно воспользоваться методами булевого линейного программирования [4, 5].

4. Однопроцессорный случай

В многопроцессорном случае получился достаточно трудоемкий алгоритм, что во многом обусловлено сложным строением сети N. Однако для случая одного процессора можно построить гораздо более эффективный полиномиальный алгоритм. Сформулируем задачу для однопроцессорного случая.

Имеется однопроцессорная система и n работ, которые нужно выполнить за время, не превосходящее $T_{\max}$. Такты работы процессоров можно считать сколь угодно малыми, переходя в пределе к непрерывному случаю. Работа i $(i = 1, \ldots, n)$ имеет объем p_i, которая определяется как время ее выполнения. До начала исполнения работы i в память процессора должны быть загружены соответствующие данные. Предполагается, что время загрузки данных для работы i прямо пропорционально загружаемому объему памяти, поэтому без ограничения общности можно считать, что оно равно их объему. Обозначим эту величину через t_i $(i = 1, \ldots, n)$. Объем памяти также определен во временных единицах, и пусть всю память процессора можно полностью загрузить за время R. Память может быть загружена (частично или полностью) некоторыми данными уже в начальный момент времени. Удаление данных из памяти происходит мгновенно сразу после выполнения соответствующей работы. Считаем, что архитектура системы такова, что загрузка данных в память процессора и выполнение им работ – независимые процессы, которые могут идти параллельно, не влияя друг на друга. При выполнении работ и загрузке данных допускаются прерывания, затратами на которые можно пренебречь. Необходимо определить, существует ли расписание, позволяющее выполнить все работы за время, не превосходящее $T_{\max}$, и в случае положительного ответа указать такое расписание.

Допустимое расписание при такой формулировке задачи фактически состоит из двух расписаний: собственно расписания выполнения работ на процессоре и расписания загрузки данных в память процессора. Ниже приведены несколько очевидных соображений, на которые будем ссылаться при дальнейших рассуждениях.

1. Для существования допустимого расписания необходимо выполнение следующих условий:

$$\sum_{i=1}^{n} p_i \le T_{\max}, \tag{3}$$

т.е. суммарное время выполнения всех работ не превышает $T_{\max}$;

$$\max_i t_i \le R, \tag{4}$$

т.е. данные, необходимые для выполнения каждой работы, могут целиком поместиться в память;

$$\sum_{i=1}^{n} t_i - R \le T_{\max} - \min_i p_i, \tag{5}$$

т.е. суммарное время загрузки в память данных для всех работ не превышает времени, отведенного на это условиями задачи. Величина $-R$ в левой части неравенства (5) означает возможность полной загрузки памяти уже в нулевой момент времени, а стоящее справа выражение показывает, что загрузка последней задачи должна быть произведена до начала ее выполнения, т.е. до момента времени $t_{\max} - \min_i p_i$. Если хотя бы одно из условий (3)–(5) не выполнено, то допустимого расписания не существует. Поэтому первой стадией алгоритма является проверка каждого из этих неравенств.

2. Прерывания выполнения работ не оправданы, поскольку все работы имеют один и тот же директивный интервал, а прерывания какой-либо работы потребуют более продолжительного нахождения ее данных в памяти.

3. Последней должна выполняться работа с наименьшим временем исполнения p_i, поскольку в этом случае на загрузку данных всех работ будет оставаться максимальное время, равное $R + T_{\max} - p_{i_{\min}}$, где $p_{i_{\min}} = \min_i p_i$.

4. Поскольку у всех работ одинаковый директивный интервал $[0; T_{\max}]$, то оптимальным порядком выполнения работ является тот, который обеспечивает минимальное суммарное время простоев процессора. Такие простои могут происходить только тогда, когда закончено выполнение очередной работы, а данные для выполнения следующей еще не успели загрузиться в память процессора.

5. Построение оптимального порядка выполнения работ

Опишем алгоритм построения оптимального порядка работ. Будем задавать его последовательностью $B = \{b_1, \ldots, b_n\}$, каждый член которой задает номер соответствующей по порядку работы. Построим эту последовательность при помощи следующего алгоритма, состоящего из n шагов.

Шаг 1. Положить $b_n = i_{\min}$; $Q_n = t_{i_{\min}}$; $B = \{b_n\}$.

Шаг r. Пусть на предыдущих $(r-1)$ шагах определены величины $b_{n-r+2}, \ldots, b_n$, $Q_{n-r+2}, \ldots, Q_n$ и множество $B = \{b_{n-r+2}, \ldots, b_n\}$. Тогда если существует такое $i \notin B$, что $p_i \le Q_{n-r+2}$, то положить $b_{n-r+1} = i_0$, $Q_{n-r+1} = Q_{n-r+2} - p_{b_{n-r+1}} + t_{b_{n-r+1}}$, где i_0 таково, что одновременно выполнены соотношения $p_{i_0} = \max_{i \notin B} p_i$ и $p_{i_0} \le Q_{n-r+2}$, (т.е. среди пока не принадлежащих множеству B номеров ищем номер i_0 такой работы, которая имеет максимальную, но при этом не превышающую Q_{n-r+2} сложность p_{i_0}). Если же такого i не существует, то положить $b_{n-r+1} = i_1$, $Q_{n-r+1} = t_{b_{n-r+1}}$, где i_1 таково, что $p_{i_1} = \min_{i \notin B} p_i$ (т.е. в этом случае просто ищем работу с минимальной сложностью, номер которой пока не принадлежит B). В обоих случаях в конце шага множество B принимает вид $B\{b_{n-r+1}, \ldots, b_n\}$.

Обоснуем корректность предложенного алгоритма. Начальные значения переменных на шаге 1 следуют из замечания 3 раздела 4. Величина Q, вычисляемая на каждом шаге, – это разница между временем начала загрузки и временем начала выполнения последней добавленной в множество B работы. Выбирая на каждом шаге

очередную работу с временем выполнения, максимально приближающимся к предыдущему значению Q, но все еще меньшим его, достигаем максимально интенсивного использования времени, отведенного на загрузку данных в память, избегая перерывов в этом процессе. Если же таких работ не осталось, берем работу с наименьшим возможным значением p_i для того, чтобы минимизировать этот перерыв. Согласно замечанию 4 раздела 4 порядок работ, обеспечивающий минимальное время простоя процессора, является оптимальным. Вычислительная сложность предложенного алгоритма $O(n^2 \log n)$ (число шагов равно n, а самой трудоемкой операцией на каждом шаге является сортировка оставшихся вне множества B работ).

Проиллюстрируем работу алгоритма построения оптимального порядка работ на конкретном примере. Пусть $n = 7$; $T_{\max} = 40$; $R = 8$; $p_1 = 1$, $t_1 = 3$; $p_2 = 5$, $t_2 = 4$; $p_3 = 7$, $t_3 = 8$; $p_4 = 9$, $t_4 = 7$; $p_5 = 4$, $t_5 = 6$; $p_6 = 8$, $t_6 = 2$; $p_7 = 5$, $t_7 = 8$.

Шаг 1. Наименее трудоемкой является 1-я работа: $b_7 = 1$; $Q_7 = 3$; $B = \{1\}$.

Шаг 2. Поскольку не существует такого $i \notin B$, что $p_i \le Q_7 = 3$, то $\min\limits_{i \notin B} p_i = 4 = p_5$, отсюда $i_1 = 5$. Поэтому $b_6 = 5$; $Q_6 = t_5 = 6$; $B = \{5, 1\}$.

Шаг 3. Существует $i \notin B$ такое, что $p_i \le Q_6 = 6$, поэтому $p_{i_0} = 5$, $i_0 = 7$. Отсюда $b_5 = 7$; $Q_5 = Q_6 - p_5 + t_5 = 6 - 5 + 8 = 9$; $B = \{7, 5, 1\}$.

Шаг 4. Существует $i \notin B$ такое, что $p_i \le Q_5 = 9$, поэтому $p_{i_0} = 9$, $i_0 = 4$. Отсюда $b_4 = 4$; $Q_4 = Q_5 - p_4 + t_4 = 9 - 9 + 7 = 7$; $B = \{4, 7, 5, 1\}$.

Шаг 5. Существует $i \notin B$ такое, что $p_i \le Q_4 = 7$, поэтому $p_{i_0} = 7$, $i_0 = 3$. Отсюда $b_3 = 3$; $Q_3 = Q_4 - p_3 + t_3 = 7 - 7 + 8 = 8$; $B = \{3, 4, 7, 5, 1\}$.

Шаг 6. Существует $i \notin B$ такое, что $p_i \le Q_3 = 8$, поэтому $p_{i_0} = 8$, $i_0 = 6$. Отсюда $b_2 = 6$; $Q_2 = Q_3 - p_6 + t_6 = 8 - 8 + 2 = 2$; $B = \{6, 3, 4, 7, 5, 1\}$.

Шаг 7. Поскольку не существует такого $i \notin B$, что $p_i \le Q_2 = 2$, то $\min\limits_{i \notin B} = 5 = p_2$ (осталась единственная работа, пока не принадлежащая B), отсюда $i_1 = 2$. Поэтому $b_1 = 2$; $Q_1 = t_2 = 4$; $B = \{2, 6, 3, 4, 7, 5, 1\}$.

Таким образом, в результате работы алгоритма получили искомый оптимальный порядок выполнения заданных работ – $\{2, 6, 3, 4, 7, 5, 1\}$.

6. Алгоритм построения однопроцессорного расписания

Перенумеруем все работы в соответствии с найденным по алгоритму из раздела 5 оптимальным порядком B. Пусть T_i – момент окончания выполнения работы i $(i = 1, \ldots, n)$. Поскольку в силу замечания 2 раздела 4 прерывания отсутствуют, то величины T_i полностью определяют расписание выполнения работ. Временные интервалы загрузок данных в память будем задавать множеством $C = \{C_1, \ldots, C_n\}$, где каждое C_i $(i = 1, \ldots, n)$ состоит из одной или нескольких пар вида $\left\{C_i^{2j+1}, C_i^{2j+2}\right\}$, элементы которых обозначают соответственно момент начала и окончания j-го интервала загрузки данных для работы i (данные могут загружаться с прерываниями, за несколько интервалов, $j = 0, 1, 2 \ldots$ – номер интервала загрузки данных для работы i). Пусть t_i^0 – объем недогруженных данных для работы i к моменту времени T_{i-1}. Опишем алгоритм построения допустимого расписания.

1. Присвоить следующие начальные значения определенным выше величинам: $T_i = \sum\limits_{j=1}^{i} p_j$, $T_0 = 0$, $t_{-1} = -R$, $C_i = \varnothing$, $t_i^0 = t_i$, $i = 1, \ldots, n$.

2. Для $r = n - 1, \ldots, 0$ выполнить: если $(T_r - T_{r-1}) - t_{r+1} > 0$, то включить в C_r пару $\{T_r - t_{r+1}, T_r\}$ и положить $t_{r+1}^0 = 0$; если $(T_r - T_{r-1}) - t_{r+1} \le 0$, то включить в C_r пару $\{T_{r-1}, T_r\}$ и положить $t_{r+1}^0 = t_{r+1} - (T_r - T_{r-1})$.

Таким образом, после шага 2 будут определены множество C и величины t_i^0, $i = 1, \ldots, n$.

3. Для $r = 1, \ldots, n$ выполнить:

3.1. Если $C_r^1 > T_{r-2}$, то:

3.1.1. положить $\Delta_1 = T_{r-2}$; $\Delta_2 = C_r^1$;

3.1.2. Обозначим $r_1 = r+1$. Если $t_{r_1}^0 = 0$, то перейти на следующую итерацию шага 3. Если $t_{r_1}^0 > 0$, то перейти на 3.1.3.

3.1.3. если $t_{r_1}^0 \le \Delta_2 - \Delta_1$, то:

3.1.3.1. включить в C_{r_1} пару $\{\Delta_1, \Delta_1 + t_{r_1}^0\}$;

3.1.3.2. положить $\Delta_1 = T_{r-2} + t_{r_1}^0$, $t_{r_1}^0 = 0$;

3.1.3.3. перейти на шаг 3.1.2;

3.1.4. если $t_{r_1}^0 > \Delta_2 - \Delta_1$, то:

3.1.4.1. уменьшить текущее значение $t_{r_1}^0$ на $\Delta_2 - \Delta_1$;

3.1.4.2. включить в C_{r_1} пару $\{\Delta_1, \Delta_2\}$;

3.1.4.3. перейти на следующую итерацию шага 3.

3.2. Если $C_r^1 \le T_{r-2}$, то:

3.2.1. увеличить текущее значение C_r^2 на t_r^0;

3.2.2. для всех $r_1 \ge r-1$ увеличить текущие значения T_{r_1} на t_r^0. Если при этом получили, что $T_n > T_{\max}$, то допустимого расписания не существует и выполнение алгоритма можно закончить;

3.2.3. положить $t_r^0 = 0$;

3.2.4. увеличить на t_r^0 все текущие значения C_i^j для всех i, j таких, что $C_i^j > C_r^2$;

3.2.5. перейти на следующую итерацию шага 3.

Для большей наглядности проиллюстрируем работу изложенного выше алгоритма, продолжив рассмотрение примера из раздела 5, для которого был найден оптимальный порядок и все работы были перенумерованы в соответствии с этим порядком. Теперь $p_1 = 5$, $t_1 = 4$; $p_2 = 8$, $t_2 = 2$; $p_3 = 7$, $t_3 = 8$; $p_4 = 9$, $t_4 = 7$; $p_5 = 5$, $t_5 = 8$; $p_6 = 4$, $t_6 = 6$; $p_7 = 1$, $t_7 = 3$. Напомним, что $n = 7$; $T_{\max} = 40$; $R = 8$.

1. При инициализации получаем $T_{-1} = -8$; $T_0 = 0$; $T_1 = 5$; $T_2 = 13$; $T_3 = 20$; $T_4 = 29$; $T_5 = 34$; $T_6 = 38$; $T_7 = 39$.

2. Подставляя в процедуру 2 конкретные значения T_r и t_r, получаем: $C_6 = \{35, 38\}$, $t_7^0 = 0$; $C_5 = \{29, 34\}$, $t_6^0 = 1$; $C_4 = \{21, 29\}$, $t_5^0 = 0$; $C_3 = \{13, 20\}$, $t_4^0 = 0$; $C_2 = \{5, 13\}$, $t_3^0 = 0$; $C_1 = \{3, 5\}$, $t_2^0 = 0$; $C_0 = \{-4, 0\}$, $t_1^0 = 0$.

3. Теперь, применяя процедуру 3 (т.е. совершая прямой проход во времени), проверим, существует ли при заданных условиях задачи допустимое расписание, и если да, то зададим его окончательными значениями T_i (концы выполнения работ) и множеством C интервалов загрузок данных в память. В результате этого прохода условие пункта 3.2 оказалось истинным только на одном шаге, а именно на 6-м ($r = 6$), поэтому там произошел сдвиг выполнения и загрузки 6- и 7-го заданий на величину $t_6^0 = 1$. Поскольку $T_n = T_7 = 40 \le T_{\max} = 40$, то допустимое расписание существует. Оно изображено на рис. 3. Жирными цифрами на этом рисунке проставлены исходные номера работ, до перенумерования их в соответствии с найденным оптимальным порядком.

Дадим обоснование предложенного алгоритма. После выполнения шага 1 все работы будут расположены в соответствии с их порядком вплотную у левого края интервала $[0, T_{\max}]$. В силу неравенства (3) $T_n \le T_{\max}$. При выполнении шага 2 мы, фактически, двигаемся по временной оси от момента T_{n-1} к нулю. Загрузка памяти

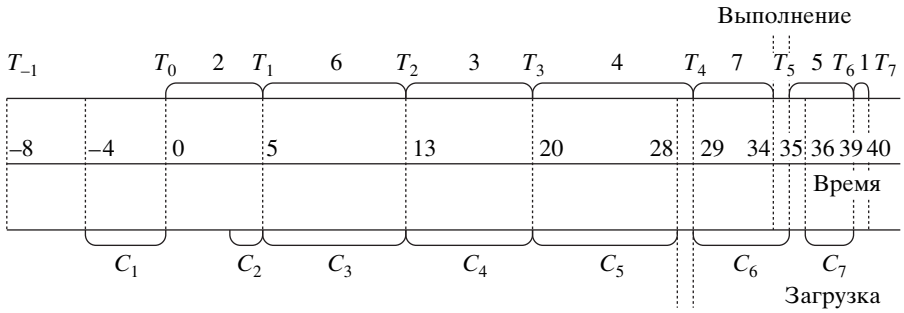

Рис. 3. Допустимое расписание, построенное в результате работы алгоритма. Стрелками показан сдвиг, произошедший на шаге 3.2 прямого прохода алгоритма относительно расписания, полученного при обратном проходе алгоритма (п.2).

организована таким образом, что к моменту времени T_i данные $(i+1)$-й работы полностью загружены в память. В этот же момент из памяти удаляются данные i-й работы, так как она в момент времени T_i завершилась. Далее, если $(T_i - T_{i-1}) - t_{i+1} > 0$, то это означает, что данные $(i+1)$-й работы могут быть полностью загружены в процессе выполнения i-й работы. Тогда $C^1_{i+1} = (T_i - T_{i-1}) - t_{i+1}$ – время начала загрузки данных $(i+1)$-й работы. Если $(T_i - T_{i-1}) - t_{i+1} < 0$, то это означает, что данные $(i+1)$-й работы могут быть загружены в процессе выполнения i-й работы не в полном объеме. Обозначаем через $t^0_{i+1} = t_{i+1} - (T_i - T_{i-1})$ объем данных $(i+1)$-й работы, которые не удалось загрузить в процессе выполнения i-й работы. Считаем, что в этом случае $C^1_{i+1} = T_{i-1}$. В первом случае, очевидно, $t^0_{i+1} = 0$. На последней итерации шага 2 $(i = 0)$ $T_i = 0$, а $T_{i-1} = -R$. Таким образом, в результате выполнения шага 2 определяется множество C и величины t^0_i, $i = 1, \ldots, n$. Отметим, что пока в каждый момент времени в памяти находятся данные не более чем для одной работы. Поэтому в силу (4) ограничение на объем памяти не было нарушено.

На шаге 3 мы движемся от момента времени $-R$ к моменту T_n. В силу (4), если начать загрузку данных первой работы в момент времени $-R$, то они будут загружены не позднее момента времени 0 (момента начала выполнения первой работы). Далее, для каждого i-го момента времени, если $C^1_i > T_{i-2}$, то назначаем к загрузке на интервал $[T_{i-2}, C^1_i]$ данные работы с текущим ненулевым остатком t^0_i и минимальным номером i до тех пор, пока имеется свободная память. Если при этом оказывается, что для какого-то номера i_0 данные для i-й работы не успевают догрузиться к моменту времени T_{i_0-1}, т.е. остается ненулевой объем недогруженных данных $t^0_{i_0}$, то осуществляется сдвиг на $t^0_{i_0}$ части расписания, расположенной правее рассматриваемого момента времени. Если после некоторого сдвига оказалось, что $T_n > T_{\max}$, то это означает, что допустимого расписания не существует. Действительно, при сдвиге расписания в памяти могут находиться данные только для i-й работы (все остальные работы уже закончены и их данные удалены из памяти), а поскольку каждый раз выбирается t^0_i с минимальным i, то данные для последующих работ еще не начали загружаться. Если мы дошли до $i = n$ и неравенство $T_n \le T_{\max}$ было выполнено всегда, то это означает существование допустимого расписания, которое задается моментами $T_i - p_i$, $i = 1, \ldots, n$ начала выполнения работ и множеством C интервалов загрузок данных в память. Вычислительная сложность алгоритма составляет $O(n^2 \log n)$. Таким образом, сложность всего алгоритма для однопроцессорного случая (построение оптимального порядка работ и поиск допустимого расписания) также составляет $O(n^2 \log n)$.

7. Заключение

Разработан алгоритм решения задачи составления допустимого расписания с прерываниями в многопроцессорной системе при заданных директивных интервалах и объемах работ, ограничениях на память процессоров и связях между ними с учетом затрат на обработку прерываний и переключений. Алгоритм основан на сведении исходной задачи к многопродуктовой потоковой задаче в сети специального вида. Для случая однопроцессорной системы разработан полиномиальный алгоритм.

СПИСОК ЛИТЕРАТУРЫ

1. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. М.: Наука, 1984.
2. *Барский А.Б.* Параллельные процессы в вычислительных системах. Планирование и организация. М.: Радио и связь, 1990.
3. *Federgruen A., Groenevelt H.* Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique // Management Sci. 1986. V. 32. No. 3. March. P. 812–829.
4. *Мину М.* Математическое программирование. Теория и алгоритмы. М.: Наука, 1990.
5. *Vaidya P.M.* Speeding up linear programming using fast matrix multiplication // Proc. 30th Ann. Sympos. Foundations Comput. Sci. 1989. P. 332–337.

Статья представлена к публикации членом редколлегии В.М. Вишневским.

Поступила в редакцию 2.12.2003