Записки научных
семинаров ПОМИ
Том 539, 2024 г.

Н. С. Григорьева

ЦИКЛИЧЕСКИЕ РАСПИСАНИЯ ДЛЯ МНОГОПРОЦЕСОРНОЙ СИСТЕМЫ

Введение

При классическом планировании набор заданий V выполняется один раз, и цель состоит в том, чтобы построить оптимальное расписание, в котором минимизируется время завершения всех заданий, также называемое длиной расписания.

Задача циклического планирования – это задача планирования, в которой набор задач V должен повторяться бесконечное количество раз. Этот подход также применим, если количество повторений цикла достаточно велико. Циклическое планирование имеет несколько приложений, таких как робототехника [1,2], производственные и торговые системы [3, 15], связь и транспорт или многопроцессорные вычисления [4].

Приложения циклического планирования обычно имеют дело с периодическим расписанием, которое представляет собой расписание одной итерации, которое повторяется в течение фиксированного интервала времени, называемого периодом (или временем цикла). Цель циклического планирования – найти периодическое расписание с минимальным периодом. Проблемы циклического планирования изучались с нескольких точек зрения. Нас интересуют задачи для параллельных одинаковых процессоров.

Задача составления циклических расписаний для параллельных процессоров (PSIP–Periodic Scheduling Identical Processors Problem) – это циклическая версия классической задачи минимизации длины расписания на m-процессорах и NP-трудна в сильном смысле [3]. Циклическое планирование не менее сложно, чем нециклическое, поскольку задача нециклического планирования полиномиально сводится к циклической задаче, где последовательные итерации не должны перекрываться. В работе [5] рассмотрены две модели PSIP для случая

Ключевые слова: задача циклического планирования, частичный порядок, параллельные процессоры.

двух процессоров и для графа отношений частичного порядка, который является набором контуров, для которых предложены алгоритмы с полиномиальной сложностью.

В статье [14] автором были предложены алгоритмы для модели с двумя процессорами и графом отношения частичного порядка, который представляет собой дерево. В этой работе рассматриваются четыре задачи составления циклических расписаний и алгоритмы их решения. Для двух задач предложены алгоритмы с полиномиалной сложностью, которые генерируют оптимальные расписания.

Данная статья организована следующим образом. Краткое описание стандартной формулировки задачи периодического планирования для идентичных процессоров приведено в §1. В §2 рассматривается циклическая версия задачи планирования на идентичных процессорах с единичными временами выполнения заданий. В §3 рассмотрена задача с произвольными времена выполнения и разрешением прерываний. Алгоритм для параллельных процессоров и независимых заданий приведен в §4. Циклический вариант задачи $P|\prec|C_{\max}$ рассматривается в §5. В §6 приведены результаты вычислительного эксперимента. §7 содержит краткое изложение этой статьи.

§1. Циклические расписания для задачи с параллельными процессорами

Рассматривается циклический вариант задачи минимизации времени выполнения множества заданий на параллельных идентичных процессорах. Это классическая задача комбинаторной оптимизации. В схеме классификации с тремя полями, предложенной Грэмом и др. [7], эта задача обозначается $P|\prec|C_{\max}$. Задача является NP-трудной [8].

Сначала рассмотрим нециклическую многопроцессорную систему заданий $V = \{v_1, v_2, \ldots, v_n\}$. Время выполнения каждого задания $p(v_i)$ известно. Частичный порядок между заданиями представлен направленным ациклическим графом задач $G = (V, E)$. E – множество направленных дуг, дуга $e = (v_i, v_j) \in E$ тогда и только тогда, когда $v_i \prec v_j$. Выражение $v_i \prec v_j$ означает, что задание v_j может быть инициирована только после завершения задания v_i.

Набор заданий выполняется на m параллельных одинаковых процессорах, любое задание может выполняться на любом процессоре, и каждый процессор может выполнять не более одного задания одновременно. Прерывание выполнения заданий не допускается. Обычная

целевая функция – это время завершения всех запланированных заданий.

Расписание S для множества V – это определение для каждого задания $v_i \in V$ время начала выполнения $t(v_i)$ и процессора $f(v_i)$, на котором задание будет выполняться. Длина расписания S равна $C_{\max}(S) = \max\{t(v_i) + p(v_i) | v_i \in V\}$.

Теперь давайте рассмотрим циклическую задачу. Предположим, что набор задач $V = \{v_1, \ldots, v_n\}$ – шаблон, который повторяется потенциально бесконечно, например, задача представляют собой шаги по созданию одного проекта, и требуется построить расписание для серии проектов. При этом можно перекрыть во времени выполнение нескольких проектов. Циклическое расписание для задачи заключается в том, что каждые w единиц времени (время цикла) начинается новый проект и одно и то же расписание выполняются для каждого проекта. Предполагается, что время завершения каждого отдельного проекта $C_{\max}(S)$ больше, чем w.

Задача PSIP (Periodic Scheduling Identical Processors Problem) может быть описана следующим образом. Пусть $V = \{v_1, \ldots, v_n\}$ множество заданий, которое выполняется бесконечное число раз.

Обозначим $\langle v_i; k\rangle$ k-е выполнение задания v_i.

Отношения частичного порядка определяются графом $G = (V, E)$ с вершинами V и дугами E. Для каждой дуги $(v_i, v_j) \in E$ задаются два значения длина $L_{ij} = p(v_i)$ и H_{ij}. H_{ij} называется высотой дуги и задает отношения предшествования между заданиями различных проектов. Если $(v_i, v_j) \in E$, то для каждой итерации $k > 0$, задание $\langle v_i; k\rangle$ должно быть закончено до начала выполнения задания $\langle v_j; k + H_{ij}\rangle$.

Множество заданий выполняется на m параллельных идентичных процессорах.

Требуется найти для каждого задания $\langle v_i; k\rangle \in V$ время начала $t(v_i; k)$ и процессор $f(v_i; k)$.

Граф $G = (V, E)$ задает следующие ограничения

$$t(v_i; k) + p(v_i) \leqslant t(v_j; k + H_{ij}).$$

Также предполагаем, что $k + 1$-е выполнение задания v_i может быть начато только после окончания выполнения k-го . Таким образом получаются ограничения

$$t(v_i; k) + p(v_i) \leqslant t(v_i; k + 1).$$

Эти ограничения включены в граф G добавлением петель (i,i) с длиной $L_{ii} = p(v_i)$ и высотой $H_{ii} = 1$ в множество дуг E.

Определение 1. *Расписание называется периодическим с временем цикла w, если $t(v_i;k) = t(v_i;1) + (k-1)w$ для всех $v_i \in V$, $k \geqslant 1 \in N$.*

Цель состоит в том, чтобы определить минимальное время цикла w_{opt} и время начала каждой операции v_i для всех $v_i \in V$. Так как время начала выполнения задания в k-м цикле зависит только от времени начала задания v_i в первом цикле, то достаточно вычислить время начала $t_i = t(v_i;1)$.

Определение 2. *Расписание называется периодическим по ресурсам с периодом K, если $f(v_i;k+K) = f(v_i;k)$.*

Рассмотрим контур μ в графе G. Пусть $L(\mu) = \sum_{(i,j)\in\mu} p(v_i)$ и $H(\mu) = \sum_{(i,j)\in\mu} H_{ij}$. Тогда $z(\mu) := \frac{L(\mu)}{H(\mu)}$ называется характеристикой контура μ. Контуры с максимальной характеристикой и положительной высотой называются критическими контурами. Тогда характеритика критического контура $z(G)$ является нижней оценкой оптимального времени цикла и $LB = \max\{\sum_{i=1}^{n} p(v_i)/m, z(G)\}$ – нижняя граница времени цикла [4].

Задача построения циклического расписания является циклической версией простой задачи составления расписания, в которой длина расписания ограничена снизу величиной критического пути. Критический контур является обобщением понятия критического пути для нашей задачи.

Рассмотрим частный случай задачи периодического планирования. Пусть $V = \{v_1, \ldots, v_n\}$ – набор заданий.

Время выполнения каждого задания $p(v_i)$ известно. Все времена выполнения являются целыми числами.

Частичный порядок между заданиями представлен ориентированным ациклическим графом $G = \langle V, E\rangle$.

Каждая дуга $(v_i, v_j) \in E$ характеризуется двумя значениями $L_{ij} = p(v_i)$ и $H_{ij} = 0$. Это постановка задачи нециклического многопроцессорного планирования.

Также предполагается, что $(k+1)$-е вхождение задания v_i может начаться только после завершения k-го вхождения. Эти ограничения включены в граф G добавлением петель E_1. Для петли $(i,i) \in E_1$ заданы длина $L_{ii} = p(v_i)$ и высота $H_{ii} = 1$, тогда однородный граф

$G^* = (V, E \cup E_1)$. В статье рассматриваются модели с данным типом однородных графов: граф G^* – ациклический граф G с петлями E_1.

Можно сгенерировать оптимальное расписание для $G = \langle V, E\rangle$ и принять длину расписания за время цикла. Но для минимизации времени цикла можно увеличить длину расписания

$$C_{\max} = \max\big\{t(v_i;1) + p(v_i) \,|\, i \in 1:n\big\} - \min\big\{t(v_i;1) \,|\, i \in 1:n\big\},$$

т.е. время от начала первой задачи до времени окончания последней задачи.

Контурами с максимальной характеристикой (критическими контурами) в данной задаче являются петли (v_i, v_i) в графе G. Тогда значение критического контура равно $\max\{p(v_i)\}$ и

$$LB = \max\Big\{\sum_{i=1}^{n} p(v_i)/m, \max\{p(v_i)\Big\}$$

– это нижняя граница оптимального времени цикла.

Задача с петлями может быть сформулирована следующим образом. Найти периодическое расписание с минимальным временем цикла z для графа G^* с петлями, удовлетворяющее условиям:

$$t(v_i;k) = t(v_i;1) + (k-1)z, \quad \forall v_i \in V,$$

$$t(v_i;k) + p(v_i) \leqslant t(v_i;k+1), \quad \forall v_i \in V,\ \forall k \geqslant 1 \in N,$$

$$t(v_i;k) + p(v_i) \leqslant t(v_j;k), \quad \forall (v_i, v_j) \in E,$$

$$f(v_i,k) = f(v_i,k+1)m, \quad \forall k \geqslant 1 \in N.$$

В статье рассматриваются четыре специальные задачи с петлями: задача с единичными временами выполнения заданий, задача с приизвольными временами выполнения, но с разрешением прерываний, задача с независимыми заданиями и задача с произвольным временем выполнения заданий без разрешения прерываний.

§2. Периодические расписания для модели с единичными временами выполнения

В данном разделе рассматривается задача, в которой задание v_i имеет единичное время выполнения. Критическими контурами являются петли (v_i, v_i) в графе G^*. Ищем целое значение времени цикла z в этой задаче. Тогда нижняя граница оптимального время цикла равна

$LB = \lceil n \rceil$. Задачу можно сформулировать следующим образом. Найти периодическое расписание с минимальным целым временем цикла z для графа G^* с петлями, удовлетворяющее условиям:

$$t(v_i; k) = t(v_i; 1) + (k-1)z, \quad \forall v_i \in V,$$

$$t(v_i; k) + 1 \leqslant t(v_i; k+1), \quad \forall v_i \in V, \ \forall k \geqslant 1 \in Z.$$

$$t(v_i; k) + 1 \leqslant t(v_j; k), \quad \forall (v_i, v_j) \in E,$$

$$f(v_i, k) = f(v_i, k+1), \quad \forall k \geqslant 1 \in N.$$

Сначала рассмотрим нециклическую задачу $P|p_j = 1, \prec |C_{\max}$, в которой задания с единичным временем выполнения должны быть запланированы на идентичных параллельных машинах. Эту нециклическую задачу можно решить за $O(n^2)$ действий алгоритмом Коффмана–Грэма [8]. Алгоритм является списочным алгоритмом и состоит из двух шагов.

Сначала все задания получают приоритеты, и на каждом этапе готовое задание с максимальным приоритетом назначается на свободный процессор. Задание считается готовым, если все его предшественники к этому этапу уже были выполнены.

Используем алгоритм Коффмана–Грэма с помощью процедуры ListCG$(G, S_L, C_{\max})$, где граф G – граф G^* без петель, S_L – расписание, построенное по алгоритму Коффмана–Грэма, $C_{\max}$ – длина построенного расписания, $t(v_i)$ время начала выполнения задания v_i в расписании. Если G – дерево или число процессоров $m = 2$, то алгоритм Коффмана–Грэма генерирует оптимальное расписание S_{opt}.

Нижняя граница времени цикла равна для этой задачи $LB = \lceil n/m \rceil$.

Предлагаемый алгорим строит оптимальное циклическое расписание, с временем цикла равным нижней границе.

Если оптимальное циклическое расписание имеет период $C_{\max}$, то при заданном количестве процессоров m невозможно начать выполнение заданий следующего проекта до полного окончания выполнения предыдущего проекта.

2.1. Алгоритм решения задачи периодического планирования с единичным временем выполнения A_1. Рассмотрим граф $G = (V, E)$ и m процессоров. $V = \{v_1, \ldots, v_n\}$.

Шаг 1. Определим нижнюю границу времени цикла $LB = \lceil n/m \rceil$, положим $z = LB$.

Шаг 2. Зададим вектор $\alpha(v_i) := 0$;

Шаг 3. Построим расписание S_L, найдем время начала для каждого задания $t(v_i)$ и длину расписания $C_{\max}$, используя процедуру ListCG $(G; S_L, C_{\max})$.

Шаг 4. Если $C_{\max} = LB$, то $z_{\text{opt}} = C_{\max}$ и оптимальное циклическое расписание $\sigma = (t, z_{\text{opt}})$, переходим к шагу 11.

Шаг 5. Найдем два множества заданий $D(z) = \{v_i \mid t(v_i) < z\}$ and $F(z) = \{v_i \mid t(v_i) \geqslant z\}$.

Шаг 6. Положим $\alpha(v_i) := \alpha(v_i)+1$ и $t(v_i) = 0$ для заданий $v_i \in F(z)$.

Шаг 7. Создадим новый граф: $G_1 = (F(z), E_1)$, G_1 является подграфом графа $G = (V, E)$. $(v_i, v_j) \in E_1$ в том и только том случае, если $(v_i, v_j) \in E$ и $v_i, v_j \in F(z)$.

Шаг 8. Заданиям из множества $F(z)$ назначаются приоритеты процедурой ListCG $(G_1; S_L, C_{\max})$.

Шаг 9. В расписании S_L рассматриваем свободные интервалы и назначаем готовое задание с максимальным приоритетом в свободный слот.

Шаг 10. Находим новое расписание S_L и его длину $C_{\max}$. Если $C_{\max} = LB$, то $z_{\text{opt}} = LB$ иначе переходим к пункту 5.

Шаг 11. Построено циклическое расписание $\sigma = (t, z_{\text{opt}})$, $t(v_i) := t(v_i) + z_{\text{opt}}\alpha(v_i)$.

Пример.

Задан граф $G = (V, E)$, $n = 9; m = 3$ (рис. 1).

Шаг 1. Найдем нижнюю границу для времени цикла $LB = n/m = 3$.

Шаг 2. Определим $\alpha(V) = (0, 0, 0, 0, 0, 0, 0, 0, 0)$.

Шаг 3. Построим расписание S_L, используя процедуру ListCG$(G, S_L, C_{\max})$. (рис. 2)

Расписание S_L представлено на рис. 2, $C_{\max} = 7$.

Шаг 4. $C_{\max} > 3$, переходим к следующему шагу.

Шаг 5. Определим $F(z) = \{v_i \in V | t_i \geqslant z\} = \{v5, v6, v7, v8, v9\}$ и $D(z) = \{v1, v2, v3, v4\}$.

Шаг 6. Определим $\alpha(V) = (0, 0, 0, 0, 1, 1, 1, 1, 1)$.

Шаг 7. Создадим новый граф $G_1 = (F(z), E_2)$

Шаги 8 и 9. Построим расписание S_2, используя алгоритм ListCG$(G_1, S_2, C_{\max})$ и устанавливая задания $F(z)$ в свободные слоты расписания S_L. (рис. 2.).

Определим $t(v5) = t(v6) = 0; t(v7) = 1; t(v8) = 1; t(v9) = 3$.

Шаг 10. $C_{\max} = 4$, $C_{\max} > z$, переходим к Шагу 5.

Шаг 5. Определим $F(z) = \{v_i \in V | t_i \geqslant z\} = \{v9\}$ and $D(z) = V \setminus v9$.

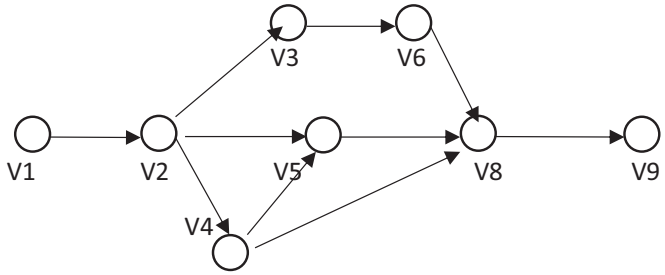

Рис. 1. $G = (V, E)$, $t(v_i) = 1$.

V1	V2	V3	V5	V7	V8	V9
		V4	V6			

V1	V2	V3	V9	
V5	V7	V4		
V6		V8		

V1	V2	V3
V5	V7	V4
V6	V9	V8

Рис. 2. Расписание S_L, $C_{\max} = 7$; расписание S_2, $C_{\max} = 4$; циклическое расписание.

Шаг 6. Положим $\alpha(V) = (0, 0, 0, 0, 1, 1, 1, 1, 2)$.

Шаг 7. Новый граф состоит из одного задания $v9$. Установим задание $v9$в свободный слот $t(v9) := 2$.

Шаг 8 и 9. Установим задание $v9$в свободный слот $t(v9) := 2$.

Шаг 10. $C_{\max} = 3$. Построено оптимальное расписание.

Шаг 11. Образец для оптимального расписания представлен на рис. 2. $t(V) = (0, 1, 2, 2, 4, 4, 5, 6, 8)$, $z_{\text{opt}} = 3$.

Теорема 1. *Алгоритм A_1 генерирует оптимальное циклическое расписание S_z с временем цикла $z_{\text{opt}} = LB$ за время $O(mn^2)$ и длина расписания $C_{\max} \leqslant mz_{\text{opt}}$.*

Доказательство. Сначала алгоритм A_1 формирует допустимое расписание S_L, определяет время начала $t(v_i)$ и находит длину расписания $C_{\max}$, используя процедуру ListCG $(G; S_L, C_{\max})$. (Шаг 3).

Далее множество заданий разбивается на два $D(z)$ и $F(z)$. Задания множества $D(z)$ уже поставлены в образец и для них выполнены

соотношения частичного порядка. Задания множества $F(z)$ будут выполняться в следующих циклах, следовательно, если задания $v \in D(z)$, $u \in F(z)$, то соотношения $v \prec u$ будут выполнены. В силу чего алгоритм строит допустимое расписание.

Задания множества $F(z)$ алгоритм будет ставить на свободные места в расписании. Пусть n_1 – это количество заданий в $F(z)$, а n_2 – это количество свободных мест в расписании S_L в интервале $[0, z_{\text{opt}})$. Тогда $n_1 \leqslant n_2$.

Покажем, что алгоритм построит расписание за конечное число итераций. Все процессоры идентичные, поэтому можно переставить задания на процессорах так, чтобы в расписании был, по крайней мере, один процессор, на котором нет свободных слотов. Все свободные слоты распределены по $m-1$ процессору.

В худшем случае алгоритм за одну итерацию заполняет свободные слоты только на одном процессоре, следовательно для построения циклического расписания S_z требуется m итераций алгоритма A_1 где $z_{\text{opt}} = LB$.

Следовательно, расписание можно построить за время $O(mn^2)$ и $C_{\max} \leqslant mz_{\text{opt}}$. Эта оценка достигается, если m единичных заданий связаны в одну цепочку и выполняются на m процессорах. □

Задача нециклического планирования в случае произвольного графа и числа процессоров больше двух является NP-трудной [8], а для циклической задачи с петлями предложен алгоритм с полимомиальной трудоемкостью.

§3. Задача составления расписаний с прерываниями

Рассмотрим задачу с произвольными временами заданий, но с разрешением прерываний.

Обозначим $t^*(v_i; k)$ время окончания выполнения задания v_i в k-м цикле, $t(v_i; k)$ – время начала выполнения задания v_i в k-м чикле.

Тогда математическая формулировка задачи:

$$\min z$$

$$t(v_i; k) = t(v_i; 0) + kz, \quad \forall v_i \in V.$$

$$t^*(v_i; k) \leqslant t(v_i; k+1), \quad \forall v_i \in V, \ \forall k \in Z.$$

$$t^*(v_i; k) \leqslant t(v_j; k), \quad \forall (v_i, v_j) \in E.$$

$$f(v_i, k) = f(v_i, k+1), \quad \forall k \geqslant 1, \ k \in N.$$

Алгоритм Мунца и Коффмана [17] использует в качестве приритета уровень задания v_i, который является величиной самого длинного пути между v_i и конечным заданием. Алгоритм имеет вычислительную сложность $O(n^2)$. В алгоритме используется понятие расписания с разделением процессоров, в котором задание получает некоторую долю β общей вычислительной мощности процессоров.

Пусть алгоритм Мунца и Коффмана строит расписание S_p. Каждое задание v_i может выполняться в нескольких интервалах, k_i число интервалов. Для каждого задания v_i алгоритм определяет время его начала $t_j(v_i)$ в j-ом блоке, где $j \in 1 : k_i$. Задание v_i выполняется $p_j(v_i)$ времени в интервале $[t_j(v_i), t_j(v_i) + p_j(v_i)]$.

$\sum_{j=1}^{k_i} p_j(v_i) = p(v_i)$. Если какое-то задание v_i не прерывается в расписании S_p, тогда $k_i = 1$ и $p_1(v_i) = p(v_i)$. Используем алгоритм Мунца и Коффмана процедурой ListMC($G; S_L, C_{\max}$).

3.1. Алгоритм A_2**.** Шаг 1. Определим нижнюю границу времени цикла $LB = \max\{\lceil(\sum_{i=1}^{n} p(v_i))/m\rceil, \max p(v_i)\}$ для оптимального времени цикла z_{opt}.
Положим $k := 0$ и $G_k(V_k, E_k) := G(V, E)$;

Шаг 2. Найдем расписание S_L для задачи с графом $G_k = (V_k, E_k)$ и его длину $C_{\max}$, используя ListMC($G_0; S_L, C_{\max}$).

Шаг 3. Если $C_{\max} = LB$, то расписание S_L является оптимальным циклическим расписанием и время цикла равно $C_{\max}$, тогда перейдем к шагу 16, иначе $z := LB$.

Шаг 4. Определим три набора заданий:
$D(z) = \{v_i \in V_k \ | t_{k_i}(v_i) + p_{k_i}(v_i) \leqslant z)\}$,
$B(z) = \{v_i \in V_k \mid (t_j(v_i) < z) \& (t_j(v_i) + p_j(v_i) > z)\}$,
$F(z) = \{v_i \in V_k \mid (t_j(v_i) \geqslant z) \& (v_i \notin B(z))\}$.

Множество $D(z)$ состоит из работ, которые завершились раньше z. Прерываем все задания из $B(z)$, которые выполняются в момент времени z.

Шаг 5. Для каждой работы v_i из $B(z)$ найдем j_0, такое, что $(t_{j_0}(v_i) < z) \& (t_{j_0}(v_i) + p_{j_0}(v_i) > z)$.

Найдем новое время выполнения заданий из $B(z)$.
Положим $p_{j_0}(v_i) = z - t_{j_0}(v_i)$, тогда $p(v_i) = \sum_{j=j_0}^{k_i} p_j(v_i)$.

Шаг 6. Найдем новое время выполнения заданий для заданий из $F(z)$.
Положим $I(v_i) = \{j \in 1 : k_i \mid t_j(v_i) \geqslant z\}$, тогда $p(v_i) = \sum_{j \in I(v_i)} p_j(v_i)$.

Шаг 7. Положим $k := k + 1$. Построим новый граф $G_k(V_k, E_k)$, который является подграфом графа $G_{k-1} = (V_{k-1}, E_{k-1})$ и $V_k = F(z) \cup B(z)$.

Шаг 8. Положим $\alpha(v_i) := \alpha(v_i) + 1$ и $t(v_i) = 0$, для $v_i \in V_k$

Шаг 9. Найдем интервалы простоя процессоров в расписании S_p.

Шаг 10. Задания V_k упорядочиваются в приритетный список процедурой $\mathrm{ListMC}(G_k; S_L, C_{\max})$.

Шаг 11. На каждом шаге готовое задание из V_k с наивысшим приоритетом назначаются на свободный процессор процедурой $\mathrm{ListMC}(G_k; S_L, C_{\max})$.

Шаг 12. Определим новое расписание S_L и его длину $C_{\max}$. Если $C_{\max} = LB$, то $z_{\mathrm{opt}} = C_{\max}$, перейти к шагу 13, в противном случае перейти к шагу 4.

Шаг 13. Построено циклическое расписание $\sigma = (t, z_{\mathrm{opt}}, \alpha(v_i))$.

Теорема 2. *Алгоритм A_2 генерирует оптимальное расписание S_z с временем цикла $z_{\mathrm{opt}} = LB$. Вычислительная сложность алгоритма $O(mn^2)$.*

Доказательство. Шаг 3 алгоритма A_2 генерирует допустимое расписание S_L, находит времена начала выполнения заданий $t(v_i)$ и вычисляет длину расписания $C_{\max}$, используя алгоритм Мунца и Коффмана, который имеет вычислительную сложность $O(n^2)$. На интервале $[0, z_{\mathrm{opt}})$ выполняются полностью задания множества $D(z)$ и задания из множеств $F(z) \cup B(z)$ могут выполняться частично и прерываться. Для заданий из множеств $F(z) \cup B(z)$ вычислены новые времена выполнения. Тогда $T_1 = \sum_{i \in F(z) \cup B(z)} p(v_i)$ – это сумма времен обработки для всех заданий из $F(z) \cup B(z)$. T_2 – это сумма всех простоев процессоров в интервале $[0, z_{\mathrm{opt}})$. Тогда $T_1 \leqslant T_2$, так как $z_{\mathrm{opt}} = \max\{\lceil (\sum_{i=1}^{n} p(v_i))/m \rceil, \max p(v_i)\}$. На каждой итерации алгоритма задания из множеств $F(z) \cup B(z)$ устанавливаются в свободные интервалы и по крайней мере на одном процессоре будут заполнены все свободные интервалы. Следовательно, для построения расписания требуется m итераций алгоритма A_2 и вычислительная сложность алгоритма $O(mn^2)$. При этом $C_{\max} \leqslant m z_{\mathrm{opt}}$. □

§4. Циклические расписания для многопроцессорной системы без прерываний

Множество заданий выполняется на параллельных идентичных процессорах. Прерывания выполнения заданий не допускается.

$$\min z$$
$$t(v_i;k) = t(v_i;1) + (k-1)z, \quad \forall v_i \in V,$$
$$t(v_i;k) + 1 \leqslant t(v_i;k+1), \quad \forall v_i \in V,\ \forall k \geqslant 1 \in Z.$$
$$t(v_i;k) + p(v_i) \leqslant t(v_j;k), \quad \forall (v_i,v_j) \in E.$$

Для этой задачи предлагается эвристический алгоритм формирования циклического расписания, в котором используется алгоритм CP (критического пути) для построения расписания. Критическим путем называется путь максимальной длины в графе. Алгоритм критического пути требует $O(n^2)$ действий для построения расписания [10] .

4.1. Алгоритм A_3. . Шаг 1. Определим нижнюю границу LB для оптимального времени цикла z_{opt}.

$LB = \max\{\lceil(\sum_{i=1}^{n} p(v_i))/m\rceil, \max p(v_i)\}$.

Шаг 2. Определим вектор вхождений $\alpha(v_i) := 0$; $k := 0$, $G_k(V_k, E_k)$ $:= G(V, E)$.

Шаг 3. Найдем расписание S для задачи A, найдем время начала выполнения заданий $t(v_i)$ и длину расписания $C_{\max}(S_L)$, используя алгоритм критического пути $CP(G; S_L, C_{\max})$.

Шаг 4. Если $C_{\max}(S_L) = LB$, то расписание S_L оптимальное циклическое расписание и время цикла равно $C_{\max}$. Тогда переходим к шагу 10 иначе $z := LB$.

Шаг 5. Определим два набора заданий

$$D(z) = \{v_i \in V_k \ | t(v_i) + p(v_i) \leqslant z\}$$

и

$$F(z) = \{v_i \in V_k \mid t(v_i) + p(v_i) > z\}.$$

Шаг 6. Положим $\alpha(v_i) := \alpha(v_i) + 1$ для $v_i \in F(z)$.

Шаг 7. Создим два новых графа: $T_1 = (D(z), U_1)$ и $T_2 = (F(z), U_2)$, где T_1 и T_2 – подграфы $G_k = (V_k, E_k)$.

Дуга $(v_i, v_j) \in U_1$ тогда и только тогда, когда $v_i, v_j \in D(z)$ и дуга $(v_i, v_j) \in U_2$ тогда и только тогда, когда $v_i, v_j \in F(z)$.

Построим новый граф $G_{k+1}(V_{k+1}, E_{k+1})$, где $V_{k+1} = D(z) \cup F(z)$. $E_{k+1} = U_1 \cup U_2$.

Шаг 8. Положим $k := k + 1$. Построим расписание S_z с помощью алгоритма СР процедурой $CP(G_k, S_z, C_{\max})$.

Шаг 9. Если $C_{\max} \geqslant 2LB$, то переходим к шагу 5.

Шаг 10. Построено циклическое расписание S_z, с временем цикла $z = C_{\max}$ и временами начал выполнения заданий $t(v_i) := t(v_i) + z\alpha(v_i)$.

Теорема 3. *Алгоритм A_3 генерирует допустимое циклическое расписание S_z и время цикла z_A удовлетворяет условию $z_A \leqslant 2z_{\text{opt}}$.*

Доказательство. Алгоритм A_3 формирует расписание S_L, определяет время начала выполнения заданий $t(v_i)$ и находит длину расписания $C_{\max}$, используя процедуру $CP(G; S_L, C_{\max})$. (Шаг 3).

Алгоритм критического пути строит допустимое расписание на каждой итерации. Перед следующей итерацией граф $G_k = (V_k, E_k)$ разбивается на два графа $T_1 = (D(z), U_1)$ и $T_2 = (F(z), U_2)$ и для всех задач графа $T_2 = (F(z), U_2)$ $\alpha(v_i)$ увеличивается на единицу. Следовательно, отношение частичного порядка между заданиями графов T_1 и T_2 будет выполнено.

Длина критического пути в графе $G_{k+1}(V_{k+1}, E_{k+1})$, по сравнению с графом $G_k = (V_k, E_k)$, уменьшается хотя бы на одно задание, так как $z_{\text{opt}} \geqslant p_{\max}$. Следовательно, возможны два случая. На какой-то итерации будет выполнено неравенство $C_{\max} \leqslant 2LB$, и тогда алгоритм завершится. Во втором случае длина критического пути в графе $G_{k+1}(V_{k+1}, E_{k+1})$ $t_{cp} \leqslant LB$ (критический путь может состоять из одного задания продолжительностью $p_{\max}$).

Пусть $C_{\max}(S_A)$ – длина последнего построенного расписания S_A и верно $t_{cp} \leqslant LB$. Тогда $C_{\max}(S_A) = (\sum_{i=1}^{n} p(v_i) + I)/m$, где I суммарный простой процессоров в интервале времени от 0 до $C_{\max}(S_A)$. Известно, что $I \leqslant t_{cp}(m-1)$, t_{cp} – длина критического пути в графе [13]. Следовательно $C_{\max}(S_A) \leqslant LB(2 - 1/m)$. Полагаем $z_A = C_{\max}(S_A)$ и $z_A \leqslant 2LB$. В худшем случае за n итераций решение будет построено. На какждой итерации алгоритм китического пути генерирует расписание за $O(n^2)$ действий. Для построения расписания требуется не более $O(n^3)$ действий. □

Пример.

Даны задания $a, V_1, V_2, \ldots, V_m$, W и $m - 1$ независимое задание $U_1, U_2, \ldots, U_{m-1}$ (рис. 3).

Эти задания выполняются на m процессорах.

Любой списочный алгоритм сгенерирует расписание, представленное на рис.4, с длиной расписания $C_{\max} = 2m - 1$ [9]. Оптимальное расписание имеет длину $C_{\max} = m + \varepsilon$.

Шаг 1. Определим нижнюю границу LB для оптимального времени цикла z_{opt}. $LB = \max\{\lceil(\sum_{i=1}^{n} p(v_i))/m\rceil, \max p(v_i)\} = m + 1$.

Шаг 2. Определим вектор вхождений $\alpha(v_i) := 0$;

Шаг 3. Формируем расписание S_L с использованием алгоритма $CP(G, S_L, C_{\max})$ (рис. 4). $C_{\max} = 2m-1$. $C_{\max} > LB$, тогда $z := m+1$.

Шаг 4. Определим два набора заданий:

$$D(z) = \{a, v_1, \ldots, v_m, u_1, \ldots, u_{m-1}\}, \quad F(z) = w$$

Шаг 5. Установим $\alpha(w) := 1$.

Шаг 6. Построим расписание S_z с помощью алгоритма СР процедурой $CP(G_{new}, S_z, C_{\max})$.

Граф G_{new} представлен на рис.5 и расписание S_z на рис. 6, $C_{\max} = m + \epsilon$.

Алгоритм A_3 формирует оптимальное расписание S_z, $z := m + \epsilon$. (рис. 6)

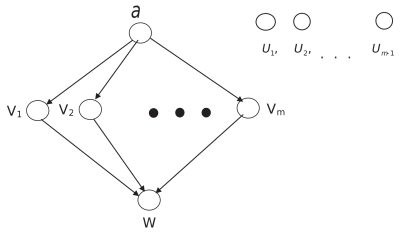

Рис. 3. Граф $t(a) = \varepsilon$; $t(V_i) = 1$; $t(U_i) = m - 1$; $t(W) = m - 1$.

§5. Циклические расписания для независимых заданий

Множество независимых заданий выполняется на параллельных идентичных процессорах. Прерывания выполнения заданий не допускается. В этой задаче откажемся от ограничения

$$t(v_i; k)+p(v_i) \leqslant t(v_i; k+1), \quad \forall v_i \in V, \quad \forall k \geqslant 1, \ \ k \in N.$$

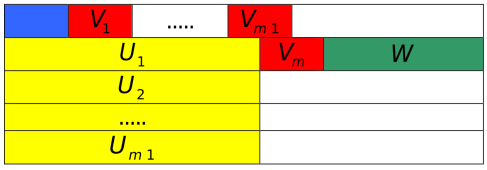

Рис. 4. Расписание S_L, длина расписания $C_{\max} = 2m - 1$.

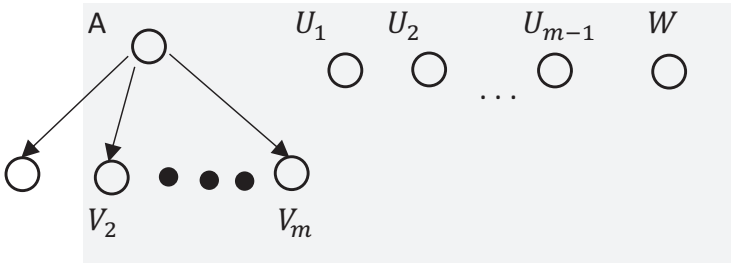

Рис. 5. Граф G_{new} : $t(a) = \varepsilon$; $t(v_i) = 1$; $t(u_i) = m - 1$; $t(w) = m - 1$.

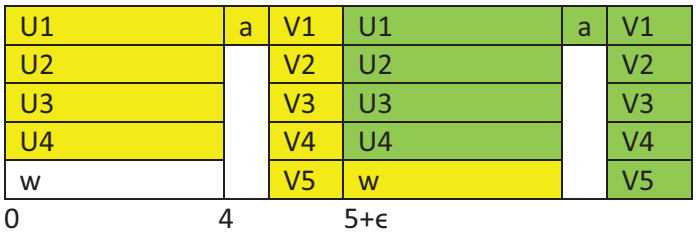

Рис. 6. Расписание S_z, время цикла $z = m + \epsilon$.

$$\min z$$
$$t(v_i; k) = t(v_i; 0) + kz, \quad \forall v_i \in V, \quad \forall k \geqslant 1, \ \ k \in Z.$$

Рассмотрим задачу $P||C_{\max}$, где набор независимых заданий требуется выполнить на одинаковых процессорах так, чтобы минимизировать длину расписания. Задача NP-трудная. Один из простейших алгоритмов – это алгоритм LPT, в котором задания упорядочены в порядке невозрастания $p(v_i)$. Затем всякий раз, когда процессор свободен, процессору назначается самое длинное задание, которое еще не обработано. Алгоритм LPT имеет гарантированную оценку точности $\frac{4}{3} - \frac{1}{3m}$. Эта оценка достигается в задаче с m процессорами и $2m+1$ заданиями, для которых заданы времена выполнения $2m-1, 2m-1, 2m-2, 2m-2, \ldots, m+1, m+1, m, m, m$.

Предлагается эвристический алгоритм A_4, который генерирует приближенное циклическое расписание.

5.1. Алгоритм A_4. Шаг 1. Определим нижнюю границу LB для оптимального времени цикла z_{opt}. $LB = \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil$.

Шаг 2. Найдем расписание S_L: время начала выполнения задания $t(v_i;1)$, $f(v_i;1)$ номер процессора, который выполняет задание v_i и длину расписания $C_{\max}$, используя процедуру $LPT(V; S_L, C_{\max})$.

Шаг 3. Если $C_{\max} = \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil$, то расписание S_L является оптимальным циклическим расписанием, время цикла $z = C_{\max}$ и $f(v_i; k+2) = f(v_i;)$, для $k \geqslant 1$, переходим к шагу 9.

Шаг 4. Определим s_j общее время работы каждого процессора j, $s_1, s_2, \ldots, s_m$. Пусть $\tau(j)$ – время начала работы процессора j, $f(v_i;1)$ – номер процессора, который выполняет задание v_i.

Шаг 5. Пусть $s_1 \geqslant s_2 \geqslant \ldots, \geqslant s_m$. Определим время цикла $z = \max\{(s_j + s_{m-j+1})/2 \,|1 \leqslant j \leqslant m\}$.

Шаг 6. Найдем $\tau(j)$ время начала работы каждого процессора j, $\tau(j) = (s_1 - s_j)/2$, $1 \leqslant j \leqslant m$.

Шаг 7. Если $f(v_i;1) = j$, то положим $t(v_i;1) := t(v_i;1) + \tau(j)$.

Шаг 8. $f(v_i;k) = m - f(v_i;k-1) + 1$, для $k \geqslant 2$.

Шаг 9. Построено циклическое расписание $\sigma = (t, f, z)$.

Теорема 4. *Алгоритм A_4 генерирует допустимое циклическое расписание S_z, $C_{\max}/C_{\text{opt}} \leqslant \frac{4}{3} - \frac{1}{3m}$ и $z_A \leqslant C_{\max}$.*

Доказательство. Алгоритм A_4 формирует расписание S_L, определяет время начала выполнения заданий $t(v_i)$ и находит длину расписания $C_{\max}$, используя процедуру $LPT(V; S_L, C_{\max})$. Время цикла $z = \max\{(s_j + s_{m-j+1})/2 \mid j \in 1:m\}$, $z \leqslant C_{\max}$. Докажем, что выполнено $z \geqslant \lceil (\sum_{i=1}^{n} p(v_i))/m \rceil$.

$\sum_{i=1}^{m} z \geqslant \sum_{i=1}^{m}((s_j + s_{m-j+1})/2)$. Следовательно, $mz \geqslant \sum_{i=1}^{n} p(v_i)$.

Докажем, что $t(v_i; 2) = t(v_i; 1) + z, \forall v_i \in V$. Пусть задание v_i выполняется на процессоре j в первой цикле, тогда во втором оно будет выполняться на процессоре $m-j+1$. Достаточно доказать, что время начала работы процессора $m-j+1$ в следующем цикле $\tau(m-j+1)+z$ больше либо равно времени окончания работы процессора j в предыдущем цикле $\tau(j) + s_j$. Докажем, что верно $\tau(j) + s_j \leqslant \tau(m - j + 1) + z$. Подставим в неравенство время начала работы процессоров $(s_1 - s_j)/2 + s_j) \leqslant (s_1 - s_{m-j+1})/2 + z$, следовательно, $(s_1 + s_j)/2) \leqslant (s_1 - s_{m-j+1})/2 + z$ и $(s_j + s_{m-j+1})/2 \leqslant z$, что верно $\forall j \in 1 : m$. Время окончания выполнения одного проекта не меняется, следовательно, $C_{\max}/C_{\text{opt}} \leqslant \frac{4}{3} - \frac{1}{3m}$. □

Пример.

Рассмотрим худший пример для алгоритма LPT для четырех процессоров и 9 заданий: $m = 4, n = 9$.

Времена выполнения заданий $p(V) = (7, 7, 6, 6, 5, 5, 4, 4, 4)$.

LPT алгоритм генерирует расписание S_L, $C_{\max} = 15$ (Рис. 7) Оптимальное расписание имеет длину $C_{\text{opt}} = 12$. $t(V)=(0, 0, 0, 0, 6, 6, 7, 7, 11)$ и $f(V)=(1, 2, 3, 4, 4, 3, 2, 1, 1)$.

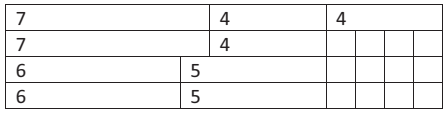

Рис. 7. Расписание S_L, $C_{\max} = 15$.

Построим циклическое расписание S_z. Найдем $s_1 = 15, s_2 = 11, s_3 = 11, s_4 = 11$. Найдем время начала работы каждого процессора $\tau_1 = 0, \tau_2 = 2, \tau_3 = 2, \tau_4 = 2$. Найдем t(V)=(0,2,2,2,8,8,9,7,11). $z = 13$. Циклическое расисание S_z представлено на рис. 8.

§6. Вычислительный эксперимент

В этом разделе представлены результаты тестирования предложенных алгоритмов A_1 и A_3.

Рис. 8. Циклическое расписание S_z, время цикла $z = 13$.

Алгоритм A_1 формирует циклическое расписание для задачи с единичными временами выполнения с минимально возможным временем цикла $z_{\text{opt}} = LB$ в худшем случае за m итераций, время выполнения одного проекта $C_{\max}$ в циклическом расписании может увеличиться. Целью вычислительного эксперимента было определение среднего количества итераций необходимых для построения оптимального расписания. Алгоритм A_1 строит расписание за две итерации в 79% тестов. Время реализации одного проекта практически не увеличивается, поскольку завершение проекта не превышает в этих случаях двух циклов.

Алгоритм $A3$ генерирует приближенное решение для задачи с произвольными временами выполнения при запрете прерываний выполнения заданий. Качество алгоритма оценивалось по среднему отклонению значения целевой функции z_A от нижней оценки LB. Алгоритм строит расписание с временем цикла, не превышающим $2LB$. В среднем время цикла превысило нижнюю оценку на 17 %. Количество итераций алгоритма зависит от отношения величины критического пути в графе к нижней оценке времени цикла и в большинстве тестов не превышало этой величины.

Заключение

В статье рассмотрены четыре задачи многопроцессорного циклического планирования, где цель – найти периодическое расписание с минимальным временем цикла при ограничениях предшествования на множестве заданий. Рассматривался циклический вариант задачи $P|prec|C_{\max}$, в котором в граф, задающий частичный порядок на множестве заданий, добавлены петли. Предложен приближенный алгоритм решения задачи с гарантированной оценкой точности равной двум. Для задачи с единичными временами выполнения и для задачи

с прерываниями предложены алгоритмы с полиномиальной сложностью, генерирующие оптимальные расписания. Алгоритм для параллельных процессоров и независимых заданий для задачи $P||C_{\max}$ генерирует приближенное циклическое расписание, не увеличивая время выполнения одного проекта.

Список литературы

1. E. Levner, V. Kats, V. E. Levit, *An improved algorithm for a cyclic robotic scheduling problem.* — Eur. J. Oper. Res. **97** (1997) 500–508.
2. V. Kats, E. Levner, *Cyclic scheduling on a robotic production line.* — J. Scheduling **5** (2002) 23–41.
3. W. Kubiak, *Solution of the Liu-Layland problem via bottleneck just-in-time sequencing.* — J. Scheduling **8**, No. 4 (2005), 295–302.
4. C. Hanen,, A. Munier, *A study of the cyclic scheduling problem on parallel processors.* — Discrete Appl. Math. **57** (1995), 167–192.
5. A. Munier,*The complexity of a cyclic scheduling problem with identical machines and precedence constraints.* — Eur. J. Oper. Res. **91**, No. 3 (1996), 471–480.
6. P. Brucker , T. Kampmeyer, *A general model for cyclic machine scheduling problems.* — Discret. Appl. Math. **156**, Bo. 13 (2008), 2561–2572.
7. J. R. L. Graham, E. L. Lawner, R. Kan, *Ann. of Disc. Math.* **5**, 287 (1979).
8. J. Ullman, *J. Comp. Sys. Sci.* **171**, 384 (1975).
9. E. G. Coffman (ed.), *Computer and Job-Shop Scheduling Theory*, John Wiley, 1978).
10. E. G. Coffman, R. L. Graham, *Optimal schedule for two-processor systems.* – Acta Informat. **1** (1972) 200–213.
11. C. Hanen, A. Munier, *Cyclic scheduling on parallel processors: an overview.* — In: Chretienne Philippe, Coffman Edward G, Lenstra Jan Karel, Liu Zhen, editors. Scheduling theory and its applications. New York: John Wiley and Sons; 1994.
12. B. Dupont de Dinechin, A. M. Kordon, *Converging to periodic schedules for cyclic scheduling problems with resources and deadlines.* — Computers & Operations Research **51** (2014) 227–236
13. R. L. Graham, *Bounds for certain multiprocessing anomalies.* — Bell System Tech. J. **45** (1966), 1563–1581.
14. N. Grigoreva, *Cyclic Scheduling for Parallel Processors with Precedence Constrains.* — J. Physics. Conf. Ser. The 2 International Scientific and Practical Conference on Mathematical modeling, Programming and Applied mathematics (2020, 5–6 November, Veliky Novgorod, Russia)
15. N. Shirvani, R. Ruiz, S. Sharokh, *Cyclic scheduling of perishable products in parallel machine with release dates, due dates and deadline.* — Int. J. Production Economics **156** (2014), 1–12
16. A. Munier, *The complexity of a cyclic scheduling problem with identical machines and precedence constraints.* — Eur. J. Oper. Res. **91**, No. 3 (1996) 471–480.
17. R. R. Muntz, E. G. Coffman, *Preemptive scheduling of real-time tasks on multiprocessor systems.* — J. Assoc. Comput. Mach. **17**, No. 2 (1970), 324–338

18. P. Sucha, Z. Hanzalek, *Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times.* — Math. Comput. Model. **47**, Nos. 9–10 (2008), 925–942.

Grigorieva N. S. Cyclic scheduling problems for multiprocessor system.

We consider the multiprocessors scheduling problem, when a set of jobs V is done on m identical parallel processors and set of jobs V is to be repeated an infinitely number of times. The goal is to generate a periodic schedule, which is a schedule of one iteration that is repeated within a fixed time interval, which called the period (or cycle time). The aim of cyclic scheduling is to find a periodic schedule with the minimum period. We consider Periodic Scheduling on Identical Processors problem. Precedence constraints between jobs are represented by an uniform graph G. We propose algorithms for constructing cyclic schedules for four problems with parallel processors: the problem with unit processing times and three problems with arbitrary processing times. The problem with unit processing times and the problem with preemptions can be solved in polynomial time.

Санкт-Петербургский
государственный университет,
Университетская наб.7-9,
199036 Санкт-Петербург, Россия
E-mail: n.s.grig@gmail.com

Поступило 17 октября 2024 г.