УДК 519.86
ББК 22.18

ЭФФЕКТИВНЫЕ АЛГОРИТМЫ ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Гончар Д. Р.[1], **Фуругян М. Г.**[2]
(*ФГБУН Вычислительный центр им.А.А. Дородницына РАН, Москва*)

Исследуется задача составления многопроцессорного расписания в системах реального времени. Рассматриваются случаи, когда (*а*) *работы допускают прерывания и переключения с одного процессора на другой;* (*б*) *прерывания и переключения не допускаются;* (*в*) *часть работ допускает прерывания и переключения, а часть не допускает. Разработан ряд приближенных алгоритмов. Приводятся результаты машинных экспериментов. Проведен сравнительный анализ разработанных алгоритмов.*

Ключевые слова: многопроцессорная система, допустимое расписание, прерывание, директивные интервалы, задача на быстродействие.

1. Введение

Одна из основных задач, возникающих при разработке программного обеспечения многопроцессорных вычислительных систем жесткого реального времени (МПВСРВ, систем, в которых заданиям сопоставляются директивные сроки, не подлежа-

[1] *Дмитрий Русланович Гончар*, *кандидат технических наук* (*rtsccas@ya.ru*).
[2] *Меран Габибуллаевич Фуругян*, *кандидат физико-математических наук*, *доцент* (*Москва, ул. Вавилова, д. 40, тел.* (*499*) *135-40-29*).

щие нарушению), заключается, во-первых, в создании математической модели, адекватной реальной вычислительной системе, и, во-вторых, в расчете с помощью этой модели режима функционирования системы, в результате которого получается расписание, показывающее, когда и какой программе должны быть выделены те или иные ресурсы ЭВМ. Это необходимо для обеспечения работоспособности и надежности таких систем. В данной статье предлагаются эффективные алгоритмы составления расписаний для МПВСРВ.

Рассматриваемый класс задач имеет, помимо чисто научной, большую практическую важность. Потребность в быстрых алгоритмах, составляющих многопроцессорные расписания, часто возникает в задачах оперативного управления на основе обработки и анализа поступающих в реальном времени данных. В качестве примеров можно привести задачи управления работой ядерных реакторов, управления испытаниями летательных аппаратов, анализа текущей ситуации в космосе на предмет оповещения об атаке баллистическими ракетами и многие другие практические задачи управления, в которых для принятия адекватных решений нужно успевать обрабатывать данные в темпе поступления. Корректность систем реального времени зависит не только от правильности результатов ее вычислений, но и от времени, за которое эти результаты были получены. В секторе проектирования систем реального времени вычислительного центра им. А.А. Дородницына РАН разрабатывается инструментальная система автоматизации проектирования вычислительных систем реального времени. Составление расписаний реального времени – важная часть подобных систем, так как все задания должны быть выполнены в срок. В настоящей статье предложены некоторые алгоритмы, разработанные авторами для данной системы. В разделе 2 рассматривается задача составления многопроцессорных расписаний с прерываниями и переключениями с одного процессора на другой для случая, когда заданы директивные интервалы. В разделе 3 рассматривается задача составления оптимального по быстродействию расписания без прерываний и переключений. В разделе 4 рас-

сматривается задача с директивными интервалами для случая, когда часть работ допускает прерывания и переключения, а часть не допускает.

2. Построение расписаний с прерываниями

В настоящем разделе рассматривается задача составления допустимого расписания с прерываниями при заданных сложностях (или объемах) работ, директивных интервалах и производительностях процессоров. Разработаны эвристические алгоритмы для случая, когда издержки на обработку прерываний и переключений не учитываются [19]. Вычислительная сложность предложенных алгоритмов значительно меньше сложности точного потокового алгоритма, что подтверждается также машинными экспериментами (выигрыш во времени составляет до нескольких тысяч раз). При этом процент некорректной работы алгоритмов незначительный (от 1 до 20% в зависимости от параметров задачи). Эвристический алгоритм работает некорректно, если он не находит допустимого расписания в том случае, когда оно на самом деле есть.

2.1. ПОСТАНОВКА ЗАДАЧИ

Рассматривается вычислительная система, состоящая из m процессоров. Каждый процессор j, $j = 1, ..., m$, характеризуется производительностью s_j. Предполагается, что процессоры упорядочены по невозрастанию производительностей ($s_1 \geq s_2 \geq \ldots \geq s_m$). Задан набор работ $N = \{1, 2, ..., n\}$, подлежащих выполнению. Каждая работа i характеризуется своим директивным интервалом $A_i = [b_i, f_i]$ (т.е. работа i может быть начата не ранее момента времени b_i и должна быть выполнена не позднее момента времени f_i), а также сложностью (или объемом работы процессоров по ее выполнению) Q_i, $i = 1, ..., n$. Работа сложности Q_i может быть выполнена на процессоре j за время Q_i/s_j. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором и каждый процессор может выполнять не более одной работы. При выпол-

нении работ допускаются прерывания и переключения с одного процессора на другой. Предполагается, что прерывания и переключения не требуют временных затрат. Задача состоит в том, чтобы определить, существует ли допустимое расписание (т.е. расписание, позволяющее выполнить все работы в их директивные интервалы на имеющихся процессорах) и, если оно существует, указать такое расписание.

2.2. ТОЧНЫЙ АЛГОРИТМ

Алгоритм, находящий точное решение рассматриваемой в этом разделе задачи, был разработан и реализован как комбинация полиномиальных алгоритмов, предложенных в [23] и [25]. Будем предполагать, что имеется t типов процессоров, каждый из которых характеризуется скоростью s_j, $j = 1, ..., t$, всего имеется m_j, $j = 1, \ldots, t$, процессоров j-го типа, $m = \sum_{j=1}^{t} m_j$.

В [23] доказывается, что такая задача может быть сведена к задаче поиска максимального потока в сети, построенной специальным образом. Для поиска максимального потока в построенной сети применялся алгоритм «поднять-и-в-начало», описанный в [9]. Предложенный алгоритм находит точное решение задачи о многопроцессорном расписании за время $O(t^3n^3)$, допуская при этом не более $2(n^2 + 2mn - 3n - m + 1)$ прерываний. Несмотря на то, что предложенный алгоритм является полиномиальным, время его работы в ряде случаев является слишком большим для практического применения в задачах достаточно большой размерности, имеющих практический смысл и возникающих при разработке МПВСРВ. Одним из решений возникшей проблемы может быть разработка существенно более быстрого эвристического алгоритма, который находил бы правильное решение задачи в достаточно широком спектре случаев.

Построим и проанализируем два эвристических алгоритма, эффективно решающих поставленную задачу. Оба эти алгоритма будут являться обобщениями однопроцессорного алгоритма Коффмана [15] на случай нескольких процессоров с различны-

ми производительностями. Назовем эти два алгоритма Эвристика 1 и Эвристика 2.

2.3. ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ

Пусть $\tau_1 < \tau_2 < \ldots < \tau_k$, $k \le 2n$, – все различные моменты времени, в каждый из которых либо завершается выполнение одной или нескольких работ, либо наступает готовность одной или нескольких работ, либо происходят оба эти события. Работа i завершается, когда суммарный объем работы процессоров по ее выполнению становится равным Q_i (если работа в течение интервала времени величиной Δ выполняется процессором j, то объем выполненной работы составляет $s_j\Delta$). В момент τ наступает готовность работы i, если $\tau = b_i$. Величины τ_l вычисляются в ходе работы алгоритма следующим образом. Пусть $a_1 < a_2 < \ldots < a_p$ – все различные величины b_i, $i = 1, \ldots, n$. Тогда полагаем $\tau_1 = a_1$. Пусть $\Delta_j = \overline{Q_i} / s_j$, если на процессоре j в момент τ_l выполняется работа i ($\overline{Q_i}$ – невыполненный объем работы i к моменту τ_l), и $\Delta_j = 0$, если процессор j в момент τ_l простаивает. Пусть $\Delta = \min\limits_{j=1,\ldots,m} \{\Delta_j : \Delta_j > 0\}$. Далее, полагаем $\tau_{l+1} = \min\{\tau_l + \Delta, a_\nu\}$, если $\tau_l \in [a_{\nu-1}, a_\nu)$, $1 \le \nu \le p$, и $\tau_{l+1} = \tau_l + \Delta$, если $\tau_l \ge a_p$.

Для каждого момента τ_l, $l = 1, \ldots, k$, определим два множества: C – множество работ, выполняемых в момент τ_l, и D – множество работ, готовых к выполнению в момент τ_l. Введем обозначения: если $C \ne \varnothing$, то i_1 – это работа из C с наибольшим директивным сроком, т.е. $d_{i_1} = \max\limits_{i \in C} f_i$; если $D \ne \varnothing$, то i_2 – это работа из D с минимальным директивным сроком, т.е. $d_{i_2} = \min\limits_{i \in D} f_i$. Алгоритм Эвристика 1 основан на следующих трех процедурах.

Процедура 1. Если в момент времени l наступает готовность работы i_0, то i_0 включается в D. Если таких работ несколько, включение выполняется поочередно в произвольном порядке.

Процедура 2. Если в момент времени τ_l работа $i_0 \in C$, которая выполнялась на процессоре j_0, завершается, то она исключается из C, а процессор j_0 освобождается. Если таких работ несколько, данная процедура выполняется для каждой из них в произвольном порядке.

Процедура 3. Если в момент времени τ_l не все процессоры заняты, то работа $i_2 \in D$ назначается на свободный процессор с минимальным номером, включается в C и исключается из D. Если свободных процессоров нет, $d_{i_1} > d_{i_2}$ и работа i_1 выполнялась на процессоре j_0, то работа i_1 снимается с процессора j_0, работа i_2 исключается из D, назначается на процессор j_0 и включается в C, а работа i_1 включается в D.

Величины f_i для работ $i \in C$ хранятся в виде кучи с максимальным элементом в вершине, а величины f_i для работ $i \in D$ хранятся в виде кучи с минимальным элементом в вершине. Таким образом, в вершине первой кучи хранится значение d_{i_1}, а в вершине второй кучи – значение d_{i_2}. Добавление элементов в множества C и D и исключение их предполагает такие преобразования этих множеств, при которых сохраняется основное свойство кучи [9].

Алгоритм Эвристика 1.

Для каждого $l = 1, 2, \ldots, k$ выполнять шаги 1 – 3.

1) Вычислить τ_l.

2) Выполнить процедуру 1. (Пусть h – число работ, готовность которых наступила в момент τ_l).

3) Выполнить процедуру 2.

4) Выполнять процедуру 3 до тех пор, пока изменяется множество D (но не более h раз).

Вычислительная сложность шага 1 и процедур 1, 2 и 3 (для одной работы) составляет соответственно $O(m)$, $O(\log_2 n)$, $O(\log_2 n)$ и $O(m + \log_2 n)$, а алгоритма Эвристика 1 – $O(n(m + \log_2 n))$. Число прерываний в полученном расписании есть $O(n)$.

Алгоритм Эвристика 2 отличается от Эвристики 1 тем, что при назначении работы на процессор сначала она включается в множество C, после чего все работы из C переназначаются на процессоры так, что работа с меньшим директивным сроком назначается на процессор с меньшим номером (т.е. на процессор с большей производительностью). В этом случае сложность процедуры 3 (для одной работы) составляет $O(m \log_2 m + \log_2 n)$ (с учетом сортировки множества C), а сложность алгоритма Эвристика 2 составляет $O(n\,(m \log_2 m + \log_2 n))$. Число прерываний в полученном расписании есть $O(mn)$.

2.4. СРАВНИТЕЛЬНЫЙ АНАЛИЗ ТОЧНОГО И ЭВРИСТИЧЕСКИХ АЛГОРИТМОВ

Были проведены машинные эксперименты по сравнительному анализу точного и эвристических алгоритмов. Машинные программы были разработаны Д.С. Гузом, который также провел вычислительные эксперименты. В таблице 1 приводится среднее время (в условных единицах) работы алгоритмов при различных значениях параметров. Как видно из таблицы 1, алгоритм Эвристика 1 является более быстрым, чем Эвристика 2, а при больших размерностях задачи оба эти алгоритма в тысячи раз быстрее точного алгоритма.

Таблица 1. Среднее время работы точного и эвристических алгоритмов в зависимости от размерности задачи

Размерность задачи		Среднее время работы		
Число процессоров	Число работ	Эвристика 1	Эвристика 2	Точный алгоритм
4	10	0,03	0,06	1,23
8	25	0,11	0,32	14,32
16	50	0,28	1,36	1090,87
16	100	1,23	4,86	6311,1
64	500	3,45	14,6	39600

Исследуем теперь вопрос о корректности обоих эвристических алгоритмов. Для этого были проведены серии численных экспериментов. Каждый эксперимент заключался в запуске при

одних и тех же условиях последовательно Эвристики 1, Эвристики 2 и алгоритма поиска точного решения. Было проведено около 25 000 серий численных экспериментов с рандомизированными и плавно варьируемыми переменными в условиях исходной задачи. В приведенной ниже таблице 2 показано, какой процент от общего числа испытаний составляют эксперименты, в которых эвристические алгоритмы Эвристика 1 и Эвристика 2 отработали некорректно. На основании этой таблицы можно сделать вывод, что Эвристика 2 работает некорректно существенно реже, чем Эвристика 1, и поэтому является более предпочтительным алгоритмом (несмотря на то, что Эвристика 2 требует несколько больше вычислительных ресурсов, чем Эвристика 1, а также генерирует расписание с большим, чем эвристика 1, числом прерываний).

Таблица 2. Процент некорректной работы эвристических алгоритмов в зависимости от размерности задачи

Размерность задачи		% некорректной работы	
Число процессоров	Число работ	Эвристика 1	Эвристика 2
4	10	18%	2%
8	25	19%	3%
16	50	22%	2%
16	100	19%	1%
64	500	17%	2%

Таким образом, можно эффективно находить допустимые расписания при помощи алгоритмов Эвристика 1 и 2, лишь иногда (когда оба эвристических алгоритма дали отрицательный ответ) прибегая к точному алгоритму.

3. Алгоритмы решения минимаксной задачи составления расписания без прерываний

Для задачи составления оптимального по быстродействию расписания без прерываний и переключений в многопроцессорной системе разработан ряд точных и приближенных алгорит-

мов [16, 20]. Определяется эффективность алгоритмов и проводится их сравнительный анализ.

3.1. ПОСТАНОВКА ЗАДАЧИ

В отличие от задачи, рассмотренной в разделе 2, в настоящем разделе предполагается, что при выполнении работ не допускаются прерывания и переключения. Кроме того, директивные интервалы не задаются, а решается задача на быстродействие. Время выполнения работы i на процессоре j равно

$\tau_{ij} = Q_i/s_j,\ \ i = 1, \ldots, n,\ \ j = 1, \ldots, m.$

Под расписанием выполнения работ будем понимать разбиение множества N на m непересекающихся подмножеств

$$N_1, N_2, \ldots, N_m\ \ \left(N = \bigcup_{j=1}^{m} N_j;\ \ N_{j_1} \bigcap N_{j_2} = \varnothing\ \text{ при } j_1 \neq j_2\right).$$

Работы из множества N_j приписываются процессору j и выполняются на нем одна за другой в произвольном порядке. Величина $B_j = \sum_{i \in N_j} \tau_{ij}$ – загруженность процессора j, $j = 1, \ldots, m$, а $\max_{j=\overline{1,m}} B_j$ – это длина расписания. Задача заключается в построении оптимального по быстродействию расписания, т.е. расписания минимальной длины.

Известно [9], что данная задача является NP-трудной в сильном смысле и все известные точные алгоритмы их решения имеют переборный характер, а точных полиномиальных (эффективных) алгоритмов в настоящее время не известно. Число шагов переборного метода, как правило, растет экспоненциально в зависимости от размеров задачи. Поэтому возникает необходимость в разработке приближенных методов.

Подобные задачи широко освещены в литературе. Отметим, например, такие методы, применяемые при их решении, как случайный и исчерпывающий поиск [8, 13], методы математического программирования [14], метод ветвей и границ [1], муравьиные алгоритмы [11, 21], поиск с запретами [24], вероятностные алгоритмы [27], генетические алгоритмы [12], метод

имитации отжига [26, 28], различные эвристические алгоритмы [4, 10, 22] и др.

3.2. ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ.

Эвристический алгоритм «*Процессор с ранним окончанием первым*» (*ПРОП*). Работа этого алгоритма состоит в следующем. На k-м шаге задание k назначается на тот процессор, суммарное время выполнения работ на котором с учетом данной работы минимальное. Иными словами, минимизируется по j величина $(B_{k-1}^{j} + \tau_{kj})$, где B_{k-1}^{j} – суммарное время выполнения работ, назначенных на j-й процессор на первых $k-1$ шагах. После чего работа k назначается на процессор j. Вычислительная сложность алгоритма составляет $O(nm)$.

Вероятностный алгоритм (*ВА*). Предлагаемый алгоритм основан на методе решения задачи упаковки, изложенном в [27]. Запишем поставленную задачу в виде задачи булевого линейного программирования:

(1) $\sum_{j=1}^{m} x_{ji} = 1, \;\; i = 1, ..., n,$

(2) $\sum_{i=1}^{n} x_{ji} \tau_{ij} \le B, \;\; j = 1, ..., m,$

(3) $x_{ji} \in \{0, 1\},$

(4) $Z = B \to \min.$

При этом работа i выполняется на процессоре j тогда и только тогда, когда $x_{ji} = 1$. В силу условий (1) и (3) каждая работа будет выполнена ровно на одном процессоре (расписание без прерываний). Условие (2) – это ограничение на длину расписания, а в силу условия (4) будет найдено расписание с минимальной длиной.

Сначала найдем решение релаксационной задачи линейного программирования, в которой $x_{ji} \in [0, 1]$. Это может быть сделано, например, симплексным методом или одним из полиномиальных алгоритмов. Пусть в результате получена матрица с элементами x_{ji}^{*}. Затем найдем такую целочисленную матрицу

$\| \bar{x}_{ji} \|$, аппроксимирующую x_{ji}^*, что $P(\bar{x}_{ji} = 1) = x_{ji}^*$ и выполняется условие (1). Это можно сделать следующим образом. Заполним элементы матрицы $\| \bar{x}_{ji} \|$ по строкам: для каждого j будем брать последовательно элементы $\bar{x}_{ji}$ с i от 1 до n, полагая $\bar{x}_{ji}$ равным 1 с вероятностью x_{ji}^*. Если некоторый элемент $\bar{x}_{ji}$, полученный таким образом, окажется равным 1, то остальные элементы строки j полагаем равными 0. Матрица $\| \bar{x}_{ji} \|$ определяет искомое расписание.

Псевдополиномиальный алгоритм (*ПА*). Теперь будем предполагать, что система состоит из m идентичных процессоров. Длительность работы $i \in N$ на любом процессоре равна t_i. Задан общий директивный срок T выполнения работ N. Предполагается, что $t_i \le T$ при всех $i \in N$. Требуется определить, существует ли допустимое расписание для работ N (т.е. такое расписание без прерываний и переключений, при котором каждая работа завершается не позднее момента времени T), и построить его, если оно существует. Будем строить допустимое расписание с помощью $(n + 1)$-уровневого дерева решений, каждый лист которого соответствует одному из возможных вариантов распределения работ по процессорам при заданном директивном сроке. Корень дерева (нулевой уровень) соответствует множеству всех вариантов распределения. С корнем ребрами связаны m вершин первого уровня, соответствующих множеству всех вариантов распределения, в которых первая работа назначена на определенный процессор. Каждой такой вершине соответствует m-мерный вектор, j-я компонента которого равна временной загруженности j-го процессора. Таким образом, вершинам первого уровня дерева решений соответствуют m-мерные векторы $(t_1, 0, \ldots, 0)$, $(0, t_1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, t_1)$ (всего m векторов). Каждая вершина первого уровня связана ребрами с m вершинами второго уровня, соответствующими множеству всех вариантов распределения работ по процессорам, в которых первые две работы закреплены за определенными процессорами. Например, вершина первого уровня

$(0, \ldots, 0, t_1, 0, \ldots, 0)$ связана с m вершинами второго уровня, которым соответствуют m-мерные векторы $(t_2, 0, \ldots, 0, t_1, \ldots, 0)$, $(0, t_2, 0, \ldots, 0, t_1, \ldots, 0), \ldots, (0, \ldots, t_2, t_1, 0, \ldots, 0), \quad (0, \ldots, t_1 + t_2, 0, \ldots, 0), \ldots, (0, \ldots, t_1, t_2, \ldots, 0), (0, \ldots, t_1, 0, \ldots, t_2)$. Далее, с вершинами второго уровня дерева решений связаны вершины третьего уровня и т.д. Каждая вершина $(c_1, c_2, \ldots, c_m)$ уровня k связана ребрами с m вершинами уровня $k + 1$, $k = 2, \ldots, n - 1$, которым соответствуют m-мерные векторы $(c_1 + t_{k+1}, c_2, \ldots, c_m)$, $(c_1, c_2 + t_{k+1}, \ldots, c_m), \ldots, (c_1, c_2, \ldots, c_m + t_{k+1})$.

Если хотя бы одна компонента вектора, приписываемого вершине, превышает величину T, то данная вершина в дерево решений не включается и дальнейшее ветвление из нее не производится. Допустимое расписание в поставленной задаче существует в том случае, если построенное дерево решений содержит хотя бы одну вершину n-го уровня (т.е. если вектор $(\tau_1, \ldots, \tau_m)$, соответствующий некоторой вершине n-го уровня, содержится в m-мерном кубе с ребром T или $\tau_j \le T$ при всех $j = 1, \ldots, m$). Для построения допустимого расписания следует построить путь, соединяющий вершину n-го уровня с корнем дерева решений. Если вершина k-го уровня ($k = 1, \ldots, n$) в этом пути соответствует процессору j, то работа k назначается на процессор j.

Определим вычислительную сложность предложенного алгоритма. Число вершин дерева решений, которые могут быть получены в результате работы алгоритма, не превосходит числа точек с целочисленными координатами в m-мерном кубе с ребром T, т.е. $(T + 1)^m$. Для каждой вершины строится не более m вершин следующего уровня. Таким образом, сложность алгоритма составляет $O(m(T + 1)^m)$ операций с m-мерными векторами. Выполнение одной операции заключается в сложении одной компоненты вектора с числом t_i, $i \in N$. Максимальное значение одной компоненты каждого вектора равно $\sum_{i \in N} t_i$, т.е. длина слова, соответствующего одной компоненте каждого вектора, с которым работает алгоритм, есть $O(\log_2 \sum_{i \in N} t_i)$. Таким образом,

предложенный алгоритм является псевдополиномиальным при фиксированном значении m.

Агрегирующий алгоритм. Метод агрегирования для поставленной задачи заключается в том, что исходное множество заданий разбивается на несколько подмножеств, для каждого из которых применяется один из точных алгоритмов построения расписания. Из полученных таким образом расписаний строится расписание выполнения исходного множества заданий. Авторами были разработаны и исследованы несколько агрегирующих алгоритмов для случая, когда все процессоры идентичны. В этом случае для каждого задания i задается его длительность t_i, которая не зависит от процессора. Лучшие результаты были получены для многоуровневого агрегирующего алгоритма (МАА), который может быть описан следующим образом. Множество всех заданий, предварительно упорядоченных по не возрастанию длительностей, разбивается на k_1 подмножеств. Если n кратно k_1, то в каждое подмножество попадает $[n/k_1]$ соседних заданий. В противном случае в некоторые подмножества может попасть на одно задание больше, при этом разброс длительностей в пределах одного подмножества должен быть минимальным. Для каждого из полученных подмножеств с помощью ПА строится оптимальное m-процессорное расписание. Задания из одного подмножества, назначенные на один и тот же процессор, объединяются в одно задание, длительность которого равна сумме длительностей соответствующих заданий. В результате получаем множество из mk_1 агрегированных заданий, которое, свою очередь, может быть аналогично разбито на k_2 подмножеств. После выполнения нескольких процедур агрегирования будет получено решение исходной задачи. При реализации данного метода полагалось $k_i = [k_{i-1}/2]$. Отметим, что подобный подход для решения задачи коммивояжера описан в [17].

3.3. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

Продемонстрируем результаты работы описанных выше алгоритмов. Машинные программы были разработаны

Д.В. Красовским, который также провел вычислительные эксперименты. Используемые обозначения: B – длина полученного расписания; $\Delta = \frac{B - \underline{B}}{\underline{B}} \times 100\,\%$ – относительная погрешность (в процентах); t – время работы алгоритма (в условных единицах); m – количество процессоров; n – количество работ; k_1 – число подмножеств заданий на первом уровне агрегирования при работе МАА. Для каждого набора (m, n) проводилось 50 экспериментов со значениями длительностей работ, полученных с помощью программного генератора случайных чисел, позволяющего получать псевдослучайные числа с равномерным распределением на отрезке [1, 1000]. Полученные значения погрешности и времени работы алгоритма затем усреднялись путем отбрасывания 5 самых лучших и 5 самых худших значений и вычисления среднего арифметического оставшихся значений.

Для случая различных процессоров в таблице 3 приведены результаты работы алгоритмов ПРОП, ВА и ПА, а также, для сравнения, одного из лучших ранее известных алгоритмов – алгоритма, основанного на методе имитации отжига (МИО), для пяти вариантов данных. Как видно из результатов расчета, из приближенных алгоритмов наиболее предпочтительным является ВА. ПА позволяет находить точные решения, однако для задач больших размерностей время его работы существенно превосходит время работы остальных алгоритмов. Для случая идентичных процессоров в таблице 4 приведены результаты работы алгоритмов МАА, ПРОП и МИО для шести вариантов данных. Как видно из результатов расчета, наименьшую погрешность имеет МАА, однако для задач больших размерностей время его работы существенно превосходит время работы остальных алгоритмов. Время работы алгоритма ПРОП существенно меньше времени работы остальных алгоритмов. Что касается погрешности, то ПРОП ненамного уступает МИО, а в ряде случаев превосходит его.

Таблица 3. Сравнительные характеристики алгоритмов ПРОП, ВА, ПА и МИО

Условия задачи	ПРОП	ВА	ПА	МИО
$m = 2$, $n = 20$	$\Delta = 18{,}2$, $t = 0{,}001$	$\Delta = 6{,}1$, $t = 0{,}1$	$\Delta = 0$, $t = 0{,}02$	$\Delta = 12{,}6$, $t = 1{,}4$
$m = 2$, $n = 50$	$\Delta = 14{,}2$, $t = 0{,}002$	$\Delta = 3{,}1$, $t = 0{,}2$	$\Delta = 2{,}9$, $t = 320$	$\Delta = 16{,}4$, $t = 2{,}5$
$m = 8$, $n = 30$	$\Delta = 47{,}6$, $t = 0{,}001$	$\Delta = 46{,}2$, $t = 0{,}1$	$\Delta = 0$, $t = 120$	$\Delta = 46{,}3$, $t = 1{,}8$
$m = 2$, $n = 1000$	$\Delta = 13{,}6$, $t = 0{,}016$	$\Delta = 0{,}2$, $t = 1{,}0$	–	$\Delta = 12{,}3$, $t = 8{,}7$
$m = 4$, $n = 5000$	$\Delta = 25{,}1$, $t = 0{,}042$	$\Delta = 0{,}1$, $t = 56$	–	$\Delta = 22{,}9$, $t = 1{,}8$

Таблица 4. Сравнительные характеристики алгоритмов МАА, ПРОП и МИО

Условия задачи	МАА	ПРОП	МИО
$m = 2$, $n = 20$, $s_1 = 2$	$\Delta = 0{,}05$, $t < 1$	$\Delta = 0{,}5$, $t < 0{,}01$	$\Delta = 3{,}7$, $t = 1{,}5$
$m = 2$, $n = 100$, $s_1 = 8$	$\Delta = 0{,}0$, $t = 1$	$\Delta = 2$, $t < 0{,}01$	$\Delta = 0{,}1$, $t = 2{,}1$
$m = 4$, $n = 20$, $s_1 = 2$	$\Delta = 0{,}5$, $t = 2{,}1$	$\Delta = 15$, $t < 0{,}01$	$\Delta = 14{,}9$, $t = 1{,}6$
$m = 4$, $n = 100$, $s_1 = 16$	$\Delta = 0{,}5$, $t = 12$	$\Delta = 1$, $t = 0{,}01$	$\Delta = 1{,}4$, $t = 2{,}4$
$m = 2$, $n = 1000$, $s_1 = 16$	$\Delta = 0{,}5$, $t = 12$	$\Delta = 3{,}5$, $t = 0{,}02$	$\Delta = 0{,}8$, $t = 9{,}1$
$m = 4$, $n = 1000$, $s_1 = 100$	$\Delta = 2{,}4$, $t = 317$	$\Delta = 3{,}5$, $t = 0{,}02$	$\Delta = 3{,}2$, $t = 12{,}4$

4. Алгоритмы составления многопроцессорного расписания для неоднородного множества работ с директивными интервалами

В настоящем разделе рассматривается задача составления многопроцессорного расписания с директивными интервалами для случая, когда часть работ допускает прерывания и переключения с одного процессора на другой, а часть не допускает. Предлагается два приближенных алгоритма, минимизирующих максимальное запаздывание [6].

4.1. ПОСТАНОВКА ЗАДАЧИ

В отличие от постановок, рассмотренных в разделах 2 и 3, предполагается, что множество работ $N = N_1 \cup N_2$, $N_1 \cap N_2 = \varnothing$, где N_1 – непрерываемые работы, N_2 – работы, допускающие прерывания и переключения с одного процессора на другой. Работы N_1 могут выполняться только процессорами $j = 1, \ldots, m_1$, $m_1 < m$ (процессорами первой группы), а работы N_2 – как процессорами первой, так и процессорами второй группы ($j = m_1 + 1, \ldots, m$).

Заданы объемы w_i работ $i \in N$. Для работ $i \in N_1$ установлен единый директивный интервал $[0; T]$, который не может быть нарушен. Для каждой работы $i \in N_2$ установлен директивный интервал $[b_i, f_i]$ $(f_i - b_i \geq t_i)$. Выполнение работы $i \in N_2$ может быть начато не ранее момента b_i, а если она завершится в момент $\bar{f}_i$, то штраф за несвоевременное выполнение работ N_2 составит величину $F = \max\limits_{i \in N_2} \max(\bar{f}_i - f_i, 0)$. Требуется найти такое расписание выполнения работ N, при котором все работы N_1 выполняются в интервале $[0; T]$ и при этом штраф F минимален.

Задачи со смешанным типом работ мало освещены в литературе. Так, например, в [5] исследована задача на быстродействие, в [2, 3] предполагается, что каждая работа строго закреплена за конкретным процессором, на множестве работ

задан частичный порядок выполнения и, кроме того, только один из приборов допускает прерывания. В [18] рассмотрены случаи, когда директивные интервалы одинаковые, а также, когда директивные интервалы могут различаться, но с рядом дополнительных ограничений.

4.2. ПЛАНИРОВАНИЕ ВЫПОЛНЕНИЯ НЕПРЕРЫВАЕМЫХ РАБОТ

Для построения расписания выполнения работ N_1 на m_1 процессорах первой группы используется приближенный оптимизационный мультиоценочный алгоритм с калибровкой [7], который, как показали численные эксперименты, имеет высокую точность и является достаточно быстрым. Если длина полученного расписания не превышает T, строится расписание прерываемых работ (раздел 4.3). Далее будем предполагать, что построено расписание выполнения работ N_1 и его длина не превосходит T. Введем следующие обозначения:

$M_1 = \{j: j = 1, \ldots, m_1\}$; $M_2 = \{j: j = m_1 + 1, \ldots, m\}$;

$M = M_1 \cup M_2$.

Для $j \in M_1$ определим величины: Q_j – длина интервала загруженности процессора j; $L_j = T - Q_j$, $L_j \geq 0$; $N_1(j)$ – номера работ из N_1, назначенных на процессор j; a_{ij}, $i \in N_1(j)$, – момент начала выполнения работы i процессором j.

4.3. ПЛАНИРОВАНИЕ ВЫПОЛНЕНИЯ ПРЕРЫВАЕМЫХ РАБОТ

Для выполнения работ N_2 используются процессоры второй группы и частично процессоры первой группы. Каждой работе $i \in N_2$ приписывается заданная величина p_i, характеризующая ее срочность. Работы с меньшей величиной p_i являются более срочными. Рассмотрим два варианта вычисления величин p_i: в первом варианте $p_i = f_i$; во втором $p_i = f_i - b_i - t_i$.

Пусть $d_1 < d_2 < \ldots < d_s$ – все различные величины b_i, $i \in N_2$. Для интервала $[d_k; d_{k+1}]$, $k = 1, \ldots, s - 1$, введем следующие обозначения:

$N_2(d_k)$ – список номеров работ $i \in N_2$, готовых к выполнению в интервале $[d_k; d_{k+1}]$ (т.е. работ $i \in N_2$, для которых $b_i \le d_k$); список $N_2(d_k)$ будем хранить упорядоченным по неубыванию величин p_i (т.е. первая работа в списке самая срочная);

τ_j – момент времени, начиная с которого на процессор j можно назначить очередную работу из $N_2(d_k)$;

ν_j, $j \in M$, – максимальное время, которое может быть выделено процессором j на выполнение работ в интервале $[d_k; d_{k+1}]$;

$M_{11}(d_k)$ – номера процессоров первой группы, на которые можно дополнительно назначить работы из $N_2(d_k)$ в интервале $[d_k; d_{k+1}]$, корректируя при этом построенное ранее расписание выполнения работ N_1;

$M_{12}(d_k)$ – номера процессоров первой группы, на которые можно дополнительно назначить работы из $N_2(d_k)$ в интервале $[d_k; d_{k+1}]$, не корректируя при этом построенное ранее расписание выполнения работ N_1.

При описании алгоритма составления расписания выполнения работ N_2 будут использованы следующие процедуры.

1. Процедура $\Pi_1(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_2$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; \nu_{j_0})$. Работу i_0 назначить на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$. Положить $\tau_{j_0} = \tau_{j_0} + \tau$; $w_{i_0} = w_{i_0} - \tau \cdot s_{j_0}$; $\nu_{i_0} = \nu_{i_0} - \tau$. Работу i_0 исключить из $N_2(d_k)$. Если $w_{i_0} \ne 0$ и $k < S$, то работу i_0 включить в $N_2(d_{k+1})$.

2. Процедура $M_{11}(d_k)$ строит множество $M_{11}(d_k)$ и вычисляет величины ν_j и τ_j, $j \in M_{11}(d_k)$.

Множество $M_{11}(d_k)$ определяется по правилу:

$M_{11}(d_k) = \{j: j \in M_1, L_j > 0,\ Q_j \ge d_{k+1};\ \exists\, i \in N_1(j),\ d_k \le a_{ij} \le d_{k+1}\}$

Для каждого $j_0 \in M_{11}(d_k)$ положить $\tau_{j_0} = \min\limits_{i \in N_1(j_0)} \{a_{ij_0} : d_k \le a_{ij_0}\}$;

$\nu_{j_0} = \min(d_{k+1} - \tau_{j_0}; L_{j_0})$.

3. Процедура $\Pi_2(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_{11}(d_k)$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; \nu_{j_0})$.

Для всех работ $i \in N_1(j_0)$, у которых $a_{ij_0} \geq \tau_{j_0}$, положить $a_{ij_0} = a_{ij_0} + \tau$.

Назначить работу i_0 на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$.

Положить $w_{i_0} = w_{i_0} - \tau \cdot s_{j_0}$; $L_{j_0} = L_{j_0} - \tau$; $\tau_{j_0} = \tau_{j_0} + \tau$; $Q_{j_0} = Q_{j_0} + \tau$; $v_{j_0} = v_{j_0} - \tau$.

Работу i_0 исключить из $N_2(d_k)$. Если $w_{i_0} \neq 0$ и $k < s$, то работу i_0 включить в $N_2(d_{k+1})$.

Если для j_0 условие принадлежности множеству $M_{11}(d_k)$ не выполняется, то исключить j_0 из $M_{11}(d_k)$.

4. Процедура $M_{12}(d_k)$ строит множество $M_{12}(d_k)$ и вычисляет величины v_j и τ_j, $j \in M_{12}(d_k)$.

Множество $M_{12}(d_k)$ определяется по правилу:

$M_{12}(d_k) = \{j : j \in M_1, Q_j < d_{k+1}\}$.

Для каждого $j_0 \in M_{12}(d_k)$ положить

$\tau_{j_0} = \max\{Q_{j_0}; d_k\}$;

$v_{j_0} = d_{k+1} - \tau_{j_0}$.

5. Процедура $\Pi_3(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_{12}(d_k)$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; v_{j_0})$.

Работу i_0 назначить на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$.

Положить $\tau_{j_0} = \tau_{j_0} + \tau$; $v_{j_0} = v_{j_0} - \tau$; $Q_{j_0} = Q_{j_0} + \tau_{j_0}$.

Перейдем к описанию алгоритма 1 распределения прерываемых работ и построения окончательного расписания выполнения работ N. Алгоритм является обобщением известного однопроцессорного алгоритма относительной срочности (RU-алгоритм Э.Г. Коффмана [15]), который при $N_1 = \varnothing$, $m = 1$ и $p_i = f_i$ находит допустимое расписание, если оно существует.

***Алгоритм* 1.**

1. Положить $k = 1$.
2. Включить в $N_2(d_k)$ все работы $i \in N_2$, для которых $b_i = d_k$.
3. Если $k < s$, перейти на шаг 4,
   если $k = s$, перейти на шаг 10.
4. Положить $\tau_j = d_k$, $\nu_j = d_{k+1} - d_k$ для $j \in M_2$.
5. С помощью процедуры $M_{11}(d_k)$ построить множество $M_{11}(d_k)$ и вычислить величины τ_j и ν_j, $j \in M_{11}(d_k)$. С помощью процедуры $M_{12}(d_k)$ построить множество $M_{12}(d_k)$ и вычислить величины τ_j и ν_j, $j \in M_{12}(d_k)$.
   Для $j \in M_1 \backslash (M_{11}(d_k) \cup M_{12}(d_k))$ положить $\nu_j = 0$.
6. Если $N_2(d_k) = \varnothing$, перейти на шаг 9;
   если $N_2(d_k) \neq \varnothing$, перейти на шаг 7.
7. Пусть $\max\limits_{j \in M} v_j = v_{j_0}$.
   Если $v_{j_0} = 0$, перейти на шаг 9;
   если $v_{j_0} \neq 0$, перейти на шаг 8.
8. Пусть i_0 – номер первой в списке $N_2(d_k)$ работы.
   Если $j_0 \in M_2$, выполнить процедуру $\Pi_1(i_0, j_0$
   если $j_0 \in M_{11}(d_k)$ выполнить процедуру $\Pi_2(i_0, j_0)$;
   если $j_0 \in M_{12}(d_k)$, выполнить процедуру $\Pi_3(i_0, j_0)$.
9. Положить $k = k + 1$. Перейти на шаг 2.
10. Положить $\tau_j = \max(Q_j; d_s)$ для $j \in M_1$ и $\tau_j = d_s$ для $j \in M_2$.
11. Если $N_2(d_s) = \varnothing$, остановиться, расписание построено; если $N_2(d_s) \neq \varnothing$, перейти на шаг 12.
12. Пусть i_0 – первая в списке $N_2(d_s)$ работа; $\min\limits_{j \in M} \tau_j = \tau_{j_0}$; $t_{i_0} = w_{i_0} / s_{j_0}$. Назначить работу i_0 на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + t_{i_0}]$; исключить работу i_0 из $N_2(d_s)$; положить $\tau_{j_0} = \tau_{j_0} + t_{i_0}$; перейти на шаг 11.

Дадим некоторые пояснения к алгоритму 1. На шаге 2 к работам, которые могли быть включены в $N_2(d_k)$ при выполнении процедур Π_1 и Π_2, добавляются работы с начальным директив-

ным сроком b_i, равным d_k. На шаге 5 определяются процессоры $j \in M_1$, которые могут выполнять в интервале $[d_k; d_{k+1}]$ прерываемые работы. На шаге 7 определяется процессор, который может выделить в интервале $[d_k; d_{k+1}]$ наибольший объем процессорного времени. На шаге 8 выбирается наиболее срочная из числа готовых к выполнению работ из N_2 и назначается на процессор, выбранный на шаге 8. Шаги 10–12 описывают построение расписания после момента d_s, которое строится по «жадному» алгоритму.

Алгоритм 2 отличается от алгоритма 1 тем, что расписание выполнения работ N_1 остается неизменным, а процессор $j \in M_1$ может использоваться только после момента Q_j. На шаге 5 вызывается только процедура $M_{12}(d_k)$ (процедура $M_{11}(d_k)$ никогда не вызывается) и полагается $\nu_j = 0$ для $j \in M_1 \backslash (M_{12}(d_k))$. На шаге 8 проверка $j_0 \in M_{11}(d_k)$ не выполняется (и поэтому процедура Π_2 вообще никогда не вызывается). Остальные шаги алгоритма 1 не изменяются.

4.4. ВЫЧИСЛИТЕЛЬНАЯ СЛОЖНОСТЬ АЛГОРИТМА

Определим сначала вычислительную сложность процедур, используемых в предложенных алгоритмах. Вычислительная сложность процедуры $\Pi_1(i_0, j_0)$ есть $O(1)$, процедуры $\Pi_2(i_0, j_0) - O(|N_1|)$, процедуры $\Pi_3(i_0, j_0) - O(1)$, процедуры $M_{11}(d_k) - O(m_1|N_1|)$, процедуры $M_{12}(d_k) - O(m_1)$. Кроме того, $s = O(|N_2|)$. Шаги 2 – 10 выполняются s раз, поэтому сложность этой части алгоритма составляет $O(m_1|N_1||N_2|)$. Сложность части алгоритма, соответствующей шагам 11, 12, составляет $O(m|N_2|)$. Таким образом, вычислительная сложность алгоритмов 1 и 2 составляет $O(|N_2| \cdot \max(m_1|N_1|, m))$.

4.5. РЕЗУЛЬТАТЫ ЧИСЛЕННЫХ ЭКСПЕРИМЕНТОВ

Были проведены численные эксперименты по сравнительному анализу алгоритмов 1, 2 и алгоритма, описанного в [5], для решения задачи на быстродействие, в которой, как и в рассмотренной в настоящей работе, N_1 – непрерываемые работы, N_2 – работы, допускающие прерывания и переключения с одного

процессора на другой, $s_1, s_2, \ldots, s_m$ – производительности процессоров, w_i – объемы работ. Алгоритм, описанный в [5], будем называть алгоритмом 3. Отметим, что вычислительная сложность алгоритма 3 составляет $O(|N_1|\log|N_1| + (|N_2|\log|N_2|)\cdot\log\widehat{T})$ [5], а сложность алгоритмов 1 и 2, приспособленных для решения задачи на быстродействие, составляет $O(|N_2|\cdot\max(m_1|N_1|, m)\cdot\log\widehat{T})$. Таким образом, в случае, когда число прерываемых работ существенно превосходит число непрерываемых работ, алгоритм 3 является более трудоемким, чем алгоритмы 1 и 2. В остальных случаях более трудоемкими являются алгоритмы 1 и 2. Следует также отметить, что алгоритм 3 может быть использован только для решения задачи на быстродействие, а алгоритмы 1 и 2 – как для решения задачи минимизации максимального запаздывания, так и для решения задачи на быстродействие.

Вычисления проводились для частного случая, когда производительности процессоров одинаковые (т.е. заданы длительности $t_i = w_i$ выполнения работ $i \in N$, а производительности процессоров равны $s_j = 1$, $j = 1, 2, \ldots, m$). Для проведения сравнительного анализа алгоритмы 1, 2 были приспособлены для решения задачи на быстродействие (которую решает алгоритм 3). Для этого рассматривался отрезок $[0; \widehat{T}]$, где

$$\widehat{T} = \max(t_{\max}, 2\cdot T^*), \quad t_{\max} = \max_{i\in N} t_i, \quad T^* = \left\lceil \left(\sum_{i\in N} t_i \right) \Big/ m \right\rceil.$$

С помощью процедуры деления отрезка $[0; \widehat{T}]$ пополам определялось такое целое значение R, что $F \neq 0$ при значениях параметров $T = R$, $b_i = 0$, $f_i = R$ и $F = 0$ при $T = R + 1$, $b_i = 0$, $f_i = R + 1$ ($i \in N$).

Число работ n и число процессоров m полагались равными $n = 100$, $m = 20$, $n = 400$, $m = 60$ и $n = 1000$, $m = 100$ (см. таблицу 5). Эксперименты проводились для различных значений числа n_1 непрерываемых работ и числа n_2 прерываемых работ. Для каждого набора значений n, m, n_1, n_2 проводилось по 50 экспериментов с произвольными значениями длительностей работ, полученных с помощью программного генератора слу-

чайных чисел, позволяющего получать псевдослучайные числа с равномерным распределением на отрезке [1, 2600]. В каждом эксперименте для алгоритмов 1, 2, 3 вычислялось среднее значение Δ оценки погрешности (по 50 расчетам) для каждого набора n, m, n_1, n_2. Относительная погрешность алгоритма вычислялась по формуле $\Delta = ((R - T^*)/T^*) \times 100\ \%$.

Из результатов численных экспериментов можно сделать следующие выводы.

1. Алгоритм 3 во всех экспериментах показал наименьшую погрешность и поэтому является наиболее точным.

2. При увеличении доли числа непрерываемых работ в общем числе работ погрешность алгоритмов 2 и 3 уменьшается, а погрешность алгоритма 1 увеличивается.

Таблица 5. Результаты численных экспериментов

m	n_1	n_2	Δ (%)		
			Алгоритм 1	*Алгоритм* 2	*Алгоритм* 3
20	99	1	12,8758	12,8758	5,3422
	90	10	10,5927	9,9584	1,8969
	80	20	11,6868	9,7330	0,1402
	70	30	12,5248	8,3514	0,0350
	60	40	13,5868	7,4849	0,0485
	50	50	14,8923	6,5079	0,0110
	40	60	16,1697	5,3710	0,0100
	30	70	17,3155	4,9975	0,0017
	20	80	18,3677	4,9338	0,0000
	10	90	20,3087	3,8595	0,0000
	1	99	20,1817	2,2781	0,0001
60	399	1	10,7130	10,7130	2,1668
	350	50	10,5679	9,1204	0,2786
	300	100	13,6249	7,7860	0,0975
	250	150	13,0765	6,6231	0,0641
	200	200	14,1618	4,9706	0,0060
	150	250	14,8607	3,4407	0,0014
	100	300	15,3573	2,8331	0,0061
	50	350	15,8864	2,1975	0,0006
	1	399	16,6970	1,2249	0,0000

m	n_1	n_2	Δ (%)		
			Алгоритм 1	*Алгоритм* 2	*Алгоритм* 3
100	999	1	7,0942	7,0942	1,2892
	900	100	7,1364	6,1736	0,0746
	800	200	7,7099	5,4968	0,0662
	700	300	8,7774	5,0084	0,0172
	600	400	9,1372	3,9406	0,0218
	500	500	9,7690	3,3775	0,0215
	400	600	10,2068	2,6450	0,0002
	300	700	10,5510	0,5969	0,0041
	200	800	10,6646	2,1021	0,0000
	100	900	11,0718	1,0274	0,0005
	1	999	11,6089	0,4214	0,0006

5. Заключение

Для задачи составления многопроцессорного расписания с прерываниями разработано два эвристических алгоритма. Вычислительная сложность предложенных алгоритмов значительно меньше сложности точного потокового алгоритма, что подтверждается также машинными экспериментами (выигрыш во времени составляет до нескольких тысяч раз). При этом процент некорректной работы алгоритмов незначительный (от 1 до 20 % в зависимости от параметров задачи). Для задачи составления оптимального по быстродействию расписания без прерываний и переключений в многопроцессорной системе разработан ряд точных и приближенных алгоритмов. Определена эффективность алгоритмов и проведен их сравнительный анализ. Для задачи составления допустимого расписания со смешанным набором работ (часть работ допускает прерывания и переключения, а часть не допускает) разработано два приближенных алгоритма. Определена сложность алгоритмов и проведен их сравнительный анализ.

Литература

1. АЛЕКСЕЕВ О.Г. *Комплексное применение методов дискретной оптимизации.* – М.: Наука, 1986. – 247 с.
2. БУЛАНЖЕ Д.Ю., СУШКОВ Б.Г. *Алгоритмы управления вычислительными системами жесткого реального времени* // Изв. АН СССР, Техн. кибернетика. – 1982. – №6. – С. 160–169.
3. БУЛАНЖЕ Д.Ю. *Оптимальная коррекция директивных интервалов для задачи одного прибора*. – М.: ВЦ АН СССР, 1983. – 20 с.
4. ГОЛОВКИН Б.А. *Расчет характеристик и планирование параллельных вычислительных процессов*. – М.: Радио и связь, 1983. – 272 с.
5. ГОНЧАР Д.Р., ФУРУГЯН М.Г. *Алгоритмы управления многопроцессорными системами с неоднородным множеством работ* // Управление большими системами. – 2010. – №29. –С. 232–244.
6. ГОНЧАР Д.Р., ФУРУГЯН М.Г. *Алгоритмы составления многопроцессорного расписания для неоднородного множества работ с директивными интервалами и произвольными процессорами* // Системы управления и информационные технологии. – 2013. – № 3.1(53). – С. 204–208.
7. ГОНЧАР Д.Р. *Мультиоценочный алгоритм решения минимаксной задачи составления расписания* // Системы управления и информационные технологии. – 2007. – №1.3(27). – С. 324–328.
8. ГОНЧАРОВ Е.Н., КОЧЕТОВ Ю.А. *Вероятностный поиск с запретами для дискретных задач безусловной оптимизации* // Дискретный анализ и исследования операций. Сер. 2. – 2002. – Т. 9, №2. – С. 13–30.
9. КОРМЕН Т., ЛЕЙЗЕРСОН Ч., РИВЕСТ Р. и др. *Алгоритмы: построение и анализ*. – М.: МЦНМО, 2005. – 1291 с.
10. КОСТЕНКО В.А. *Алгоритмы построения расписаний для вычислительных систем реального времени, допускающие*

использование имитационных моделей // Программирование. – 2013. – №5 – С. 53–71.
11. КОСТЕНКО В.А., ПЛАКУНОВ А.В. *Алгоритм построения одноприборных расписаний, основанный на схеме муравьиных колоний* // Известия РАН. Теория и системы управления. – 2013. – № 6. – С. 87–96.
12. КОСТЕНКО В.А., СМЕЛЯНСКИЙ Р.Л., ТРЕКИН А.Г. *Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов* // Программирование. – 2000. – №5. – С. 63–72.
13. КОЧЕТОВ Ю., МЛАДЕНОВИЧ Н., ХАНСЕН П. *Локальный поиск с чередующимися окрестностями* // Дискретный анализ и исследования операций. Сер. 2. – 2003. – Т. 10, №1. – С. 11–44.
14. КОЧЕТОВ Ю.А., СТОЛЯР А.А. *Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограниченными ресурсами* // Дискретный анализ и исследования операций. Сер. 2. – 2003. – Т. 10, №2. – С. 29–56.
15. КОФФМАН Э.Г. *Введение в детерминированную теорию расписаний* // Теория расписаний и вычислительные машины / Под ред. Коффмана Э.Г. – М.: Наука, 1984. – С. 9–64.
16. КРАСОВСКИЙ Д.В., ФУРУГЯН М.Г. *Алгоритмы решения минимаксной задачи составления расписания* // Изв. РАН, ТиСУ. – 2008. – Т. 47, №5. – С. 732–736.
17. СИГАЛ И.Х., ИВАНОВА А.П. *Введение в прикладное дискретное программирование*. – М.: Физматлит, 2002. – 240 с.
18. СКИНДЕРЕВ С.А., ФУРУГЯН М.Г. *Алгоритмы планирования вычислений в многопроцессорных системах с неоднородным множеством работ*. – М.: ВЦ РАН, 2006. – 28 с.
19. ФУРУГЯН М.Г. *Некоторые алгоритмы анализа и синтеза многопроцессорных вычислительных систем реального времени* // Программирование. – 2014. – №1. – С. 36–44.
20. ФУРУГЯН М.Г. *Некоторые алгоритмы решения минимаксной задачи составления многопроцессорного расписания* // Изв. РАН, ТиСУ. – 2014. – №2. – С. 48–54.

21. ШТОВБА С.Д. *Муравьиные алгоритмы* // ExponentaPro. Математика в приложениях. – 2003. – №4(4). – С. 70–75.
22. BRUCKER P. *Scheduling Algorithms*. – Heidelberg, Springer, 2007. – 371 p.
23. FEDERGRUEN A., GROENEVEL H.T. *Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique* // Management Science. – 1986. – Vol. 32, №3. – P. 341–349.
24. GLOVER F., LAGUNA M. *Chapter 3: Tabu search* / Ed. R. Colin Reeves, Modern Heuristics Techniques for Combinatorial Problems. – Oxford, Blackwell Scientific Publications, 1993. – P. 70–150.
25. GONZALES T., SAHNI S. *Preemptive Scheduling of Uniform Processor Systems* // Journal of the Association for Computing Machinery, January. – 1978. – Vol. 25, №1. – P. 92–101.
26. LAARHOVEN P., AARTS E., LENSTRA J. *Job Shop Scheduling by Simulated Annealing* // Operations Research. – 1992. – Vol. 40(1). – P. 113–125.
27. RAGHAVAN R. *Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs* // J. Computer and System Sciences. – 1988. – Vol.37. – P. 130–143.
28. SHEN C., PAO Y., YIP P. *Scheduling multiple job problems with guided evolutionary simulated annealing approach* // Proc. First IEEE Conf. on Evolutionary Computations. – Orlando, 1994. – P. 702–706.

EFFICIENT SCHEDULING ALGORITHMS IN MULTI-PROCESSOR REAL TIME SYSTEM S

Dmitry R. Gonchar, Computing Center of RAS, Moscow, Cand. Sc. (rtsccas@ya.ru).
Meran G. Furugyan, Computing Center of RAS, Moscow, Cand. Sc. assistant professor (rtsccas@ya.ru).

Abstract: *We study a task scheduling problem for real-time multi-processor systems and consider the cases when (a) – jobs are pre-emptive and allow for processor switch, (b) – jobs are not pre-emptive and processor switch are prohibited, (c) – only a part of tasks are pre-emptive and allow for processor switch. We suggest a number of approximate algorithms, provide simulation results, and carry out comparative analysis of the developed algorithms.*

Keywords: multiprocessor system, preemptive and no preemptive jobs, scheduling.

Статья представлена к публикации
членом редакционной коллегии М.Ф. Караваем

Поступила в редакцию 14.02.2014.
Опубликована 31.05.2014.