УДК 65.012.122

СУБОПТИМАЛЬНЫЙ АЛГОРИТМ ПОСТРОЕНИЯ РАСПИСАНИЙ ДЛЯ ИЕРАРХИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Н. В. Колесов,
доктор техн. наук, профессор, начальник сектора
М. В. Толмачева,
ведущий инженер
ГНЦ РФ ЦНИИ «Электроприбор» (*Санкт-Петербург*)

Рассматривается субоптимальный алгоритм составления расписаний в иерархических вычислительных системах, практически не требующий перебора вариантов. Приводятся результаты исследований его эффективности на основе случайного генерирования примеров.

We propose a suboptimal scheduling algorithm for hierarchical computing systems which requires almost no case study of schedule versions. It is based on several optimal no-case-study scheduling algorithms. Each of the algorithms is intended for its own, rather narrow basic class of systems.

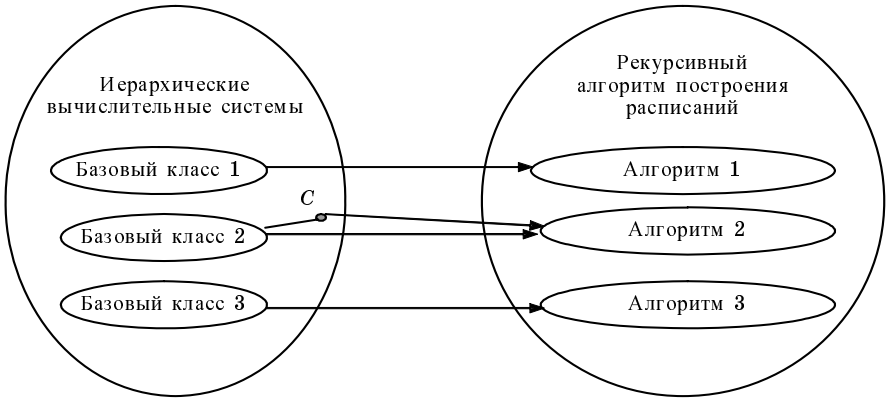
Введение

Проблема построения расписаний выполнения задач в многопроцессорных системах обработки информации является достаточно важной, и ей посвящены многочисленные публикации, ссылки на которые можно найти, например, в работах [1–3]. Данная проблема может рассматриваться в различной постановке. В настоящей статье она рассматривается применительно к иерархической вычислительной системе, которая решает некоторую последовательность задач. При этом каждая задача разбита на фрагменты по числу процессоров. Предполагаются известными времена решения для всех фрагментов каждой из задач. Известно [1–3], что решение проблемы составления расписаний в общем случае в той или иной степени предполагает перебор вариантов возможных расписаний и характеризуется высокой алгоритмической сложностью. Ниже предлагается субоптимальный алгоритм построения расписаний, практически не требующий перебора вариантов. Дальнейшее изложение отчасти коррелируется с предыдущей работой авторов [4], которая была посвящена существенно более узкому классу вычислительных систем – конвейерным системам.

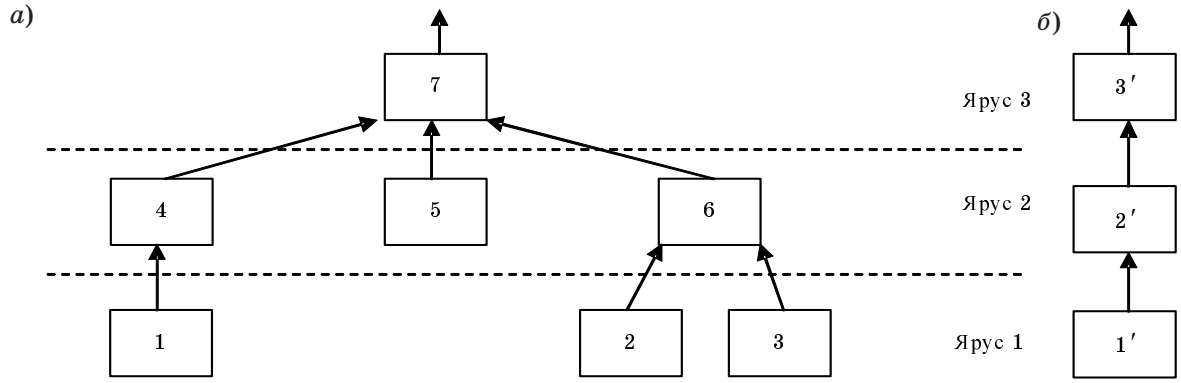
Постановка задачи и основная идея предлагаемого алгоритма

Пусть рассматриваемая система представляет собой иерархию из m процессоров $L = \{L_1, L_2, ..., L_m\}$, на вход которой поступает последовательность $(J_1, J_2, ..., J_n)$ из n задач. Каждая задача разбивается на m фрагментов (по числу процессоров). При этом i-й фрагмент j-й задачи решается на i-м процессоре за время $\tau_{i,j}$ $i = \overline{1, m}$, $j = \overline{1, n}$ и $\tau_{i,j} = t^{к}_{i,j} - t^{н}_{i,j}$, где $t^{н}_{i,j}$ и $t^{к}_{i,j}$ – времена начала и конца решения j-й задачи на i-м процессоре. Проблема состоит в определении расписания решения задач, которое не существенно проигрывает минимальному расписанию по суммарному времени решения всех задач. Заметим, что под минимальным расписанием понимается расписание, характеризующееся минимальным временем решения всех задач.

Для решения поставленной задачи предлагается субоптимальный рекурсивный алгоритм, основная идея которого сводится к следующему (рис. 1). В основе предлагаемого алгоритма лежат три одношаговых алгоритма построения расписаний, каждый из которых ориентирован на свой, в общем случае достаточно узкий, базовый класс систем. Важно то, что эти алгоритмы являются беспереборными и оптимальными. В предложенном алгоритме на каждом шаге рекурсии в будущем расписании размещаются одна или две задачи из числа не размещенных на предыдущих шагах. При этом используемый одношаговый алгоритм размещения соответствует тому базовому классу, к которому наиболее близка рассматриваемая на данном шаге система. Эти действия повторяются до исчерпания заданного списка задач.



■ ***Рис. 1.*** *Иллюстрация основной идеи предлагаемого алгоритма*



■ ***Рис. 2.*** *Примеры иерархических систем: а – иерархическая система общего вида; б – конвейерная система*

Базовые классы иерархических систем

Обсудим предварительно основные понятия и, прежде всего, понятие иерархической системы. На рис. 2, *а* приведен пример трехъярусной иерархической системы из семи процессоров. Следует подчеркнуть, что в данном контексте речь идет об иерархических информационных связях. При этом их физическая реализация может быть другой и иметь, например, магистральную архитектуру. Система на рис. 2, *а* не имеет циклов и разветвлений, включает четыре входных процессора *1–3* и *5* и один выходной *7*. Отметим, что конвейерная система является частным случаем иерархической системы, когда каждый ярус содержит лишь один процессор (рис. 2, *б*).

Каждый входной процессор L_i связан с выходным процессором L_m некоторым вычислительным путем (последовательностью процессоров) $p_k = L_i, L_j, ..., L_m$. Назовем величину $t_{k,j}$ временем выполнения вычислительного пути p_k на j-й задаче. Определим ее как сумму времен выполнения фрагментов j-й задачи процессорами, принадлежащими этому пути, что с использованием нумерации процессоров вдоль пути p_k можно записать:

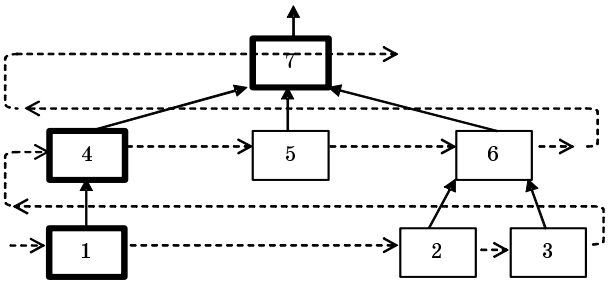
$$t_{k,j} = \sum_{i=1}^{m_k} \tau_{i,j},$$

где m_k – число процессоров, принадлежащих пути p_k.

Заметим, что в общем случае реальное время выполнения вычислений на этом пути будет больше из-за необходимости ожидания готовности исходных данных для процессоров вычислительного пути p_k, вычисляемых в процессорах, не принадлежащих этому пути.

Назовем вычислительный путь p_j^* критическим для j-й задачи, если время его выполнения на j-й задаче является максимальным среди всех остальных путей системы. Очевидно, что для разных задач, решаемых в одной и той же системе, критические вычислительные пути могут быть различными.

Для любой иерархической системы время $t_{i,j}^{н}$ начала обработки информации в некотором процессоре L_i определяется временем подготовки для него всех исходных данных, которые, в свою очередь, складываются из ряда частных данных, подготавливаемых предшествующими процессорами



■ ***Рис. 3.*** *Иллюстрация отношения доминирования для систем класса 1*

L_r по каждому r-му информационному входу L_i (так, на рис. 1, *а* процессору *7* предшествуют процессоры *4*, *5* и *6*). В результате время готовности данных для процессора L_i определяется максимальным среди времен $t^{к}_{r,j}$ подготовки частных данных, т. е.

$$t^{н}_{i,j} = \max_{r}\{t^{к}_{r,j}\}.$$

В свою очередь, время $t^{к}_{r,j}$ подготовки частных данных по некоторому вычислительному пути определяется не только временем обработки данных в процессорах этого пути, но также временем ожидания готовности данных и временем начала вычислений во входном процессоре на рассматриваемом вычислительном пути. Последняя величина имеет в общем случае разные значения для разных вычислительных путей для любой задачи, кроме первой.

Рассмотрим три базовых для данного рассмотрения класса иерархических систем. В дальнейшем этот список может быть расширен. Особенностью всех рассматриваемых классов является существование характерных упорядоченностей на множестве процессоров по определяемому ниже отношению доминирования. Первый класс связан с наличием убывающей упорядоченности, второй класс – с наличием возрастающей упорядоченности, третий класс – с наличием двух упорядоченностей (возрастающей и убывающей).

Введем на множестве процессоров отношение доминирования «>».

Определение. Процессор L_q доминирует над процессором L_r ($L_q > L_r$), если

$$\min_{j}\tau_{q,j} \;\geq\; \max_{j}\tau_{r,j}.$$

Определим три базовых класса иерархических систем.

Класс 1. Множество процессоров представляет собой последовательность $L_m > L_2 > ... > L_m$, убывающую по отношению доминирования как вдоль каждого из ярусов системы, так и от нижнего яруса к верхнему таким образом, что процессоры, являющиеся максимальными элементами каждого из ярусов, образуют путь (критический).

Класс 2. Множество процессоров представляет собой последовательность $L_1 < L_2 <...< L_m$, возрастающую по отношению доминирования как вдоль каждого из ярусов системы, так и от нижнего яруса к верхнему таким образом, что процессоры, являющиеся максимальными элементами каждого из ярусов, образуют путь (критический).

Класс 3. Множество процессоров представляет собой пару соединенных последовательностей $L_1 < L_2 <...< L_h >...> L_{m-1} > L_m$, $1 \le h \le m$, первая из которых возрастает, а вторая убывает по отношению доминирования как вдоль каждого из ярусов системы, так и от нижнего яруса к верхнему таким образом, что процессоры, являющиеся максимальными элементами каждого из ярусов, образуют путь (критический путь).

На рис. 3 приведена иерархическая система, соответствующая классу 1. Штриховой линией показано отношение доминирования, а жирными линиями – процессоры *1*, *4* и *7*, являющиеся максимальными элементами ярусов. Путь, который образуют эти процессоры, является критическим для всех задач системы.

Построение оптимальных расписаний для базовых классов

Покажем теперь, как сконструировать оптимальное расписание в системе из класса 1. При этом характеристики критического пути будем отмечать звездочкой (*).

Алгоритм 1

1. Выделим задачу J_s, которая удовлетворяет условию

$$\sum_{i=2}^{m^*}\tau^{*}_{i,s} = \min_{j}\left\{\sum_{i=2}^{m^*}\tau^{*}_{i,j}\right\}.$$

2. Сформируем расписание $\pi_1 = [\tilde{\pi}\;\; J_s]$, где $\tilde{\pi}$ – произвольное расписание для $(n-1)$ задач, не содержащее задачи J_s.

Покажем оптимальность алгоритма 1. Предварительно отметим важную особенность систем из класса 1, заключающуюся в том, что ни перед одним из процессоров, принадлежащих критическому пути, кроме входного, никогда не образуется очередь из фрагментов задач, ожидающих решения. Очередь на входе процессора может возникать по двум причинам: из-за ожидания подготовленными на критическом пути исходными частными данными готовности других частных данных, а также из-за ожидания момента окончания решения на данном процессоре фрагмента предыдущей задачи. Ожидание в силу первой причины исключается, поскольку при любой задаче решение на любом процессоре критического пути будет всегда заканчиваться позже, чем на других процессорах того же яруса. Это будет происходить ввиду

того, что этот процессор является максимальным элементом яруса по отношению доминирования, а время начала вычислений на входном процессоре критического пути всегда будет наиболее поздним среди всех входных процессоров. Ожидание в силу второй причины также исключается, поскольку ввиду доминирования предшествующего процессора критического пути над последующим самый короткий для предшествующего процессора фрагмент любой задачи будет решен за большее время, нежели самый длинный фрагмент любой другой задачи на последующем процессоре.

С учетом сказанного можно записать выражение для времени $T_1(\pi)$ выполнения произвольного расписания π в системе из класса 1. Это время можно представить в виде суммы двух составляющих: времени ожидания в исходной очереди для фрагмента последней n-й задачи, соответствующего входному процессору критического пути, а также времени t_n^* выполнения критического пути для этой задачи. Такое представление возможно, поскольку момент начала ожидания для входного фрагмента критического пути n-й задачи совпадает с моментом начала выполнения расписания, а момент окончания выполнения критического пути – с моментом окончания выполнения расписания. Причем время ожидания будет определяться суммой времен решения входных фрагментов критического пути для всех задач, кроме последней. В результате с использованием нумерации процессоров вдоль критического пути имеем

$$T_1(\pi) = \sum_{j=1}^{n-1} \tau_{1,j}^* + t_n^* = \sum_{j=1}^{n-1} \tau_{1,j}^* + \sum_{i=1}^{m^*} \tau_{i,n}^*,$$

где $\tau_{i,j}^*$ – время решения фрагмента j-й задачи на i-м процессоре критического пути системы; m^* – число процессоров, принадлежащих критическому пути.

Перенося для удобства анализа первое слагаемое из второй суммы в первую сумму, получаем

$$T_1(\pi) = \sum_{j=1}^{n} \tau_{1,j}^* + \sum_{i=2}^{m^*} \tau_{i,n}^*.$$

Значение первой суммы представляет собой суммарное время, затрачиваемое на решение входных фрагментов критического пути для всех задач. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. В отличие от первой суммы, вторая зависит от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой суммы. Отсюда следует, что расписание, формируемое в алгоритме 1, является оптимальным для класса 1.

Отметим два существенных обстоятельства. Во-первых, в данном алгоритме отсутствует перебор вариантов расписаний. Во-вторых, для оптимальности расписания достаточно лишь правильного выбора последней исполняемой задачи, которая должна быть наиболее быстро решаемой на всех процессорах критического пути системы, кроме, возможно, первого. Упорядоченность же остальных задач не влияет на время исполнения расписания.

Рассмотрим правила формирования расписания во втором случае, который является в определенном смысле противоположным первому. Он также характеризуется определяющим свойством, но оно заключается в том, что перед каждым из процессоров критического пути системы всегда образуется очередь из фрагментов задач, ожидающих решения. В результате каждая задача, кроме первой, будет стартовать на любом процессоре критического пути с некоторой задержкой относительно момента ее прибытия на вход процессора, поскольку любой i-й фрагмент j-й задачи будет заканчиваться позже, нежели $(i-1)$-й фрагмент следующей за ней $(j+1)$-й задачи. Этот факт также следует из определения отношения доминирования. Алгоритм конструирования расписания в этом случае будет выглядеть следующим образом.

Алгоритм 2

1. Выделим задачу J_l, которая удовлетворяет условию

$$\sum_{i=1}^{m^*-1} \tau_{i,l}^* = \min_j \left\{ \sum_{i=1}^{m^*-1} \tau_{i,j}^* \right\}.$$

2. Сформируем расписание $\pi_2 = [J_l \tilde{\pi}]$, где $\tilde{\pi}$ – произвольное расписание для $(n-1)$ задач, не содержащее задачи J_l.

Покажем оптимальность этого алгоритма. Запишем выражение для времени выполнения произвольного расписания для системы из класса 2. Учтем, что перед последним m^*-м процессором критического пути системы (как и перед всеми другими) всегда существует очередь, а значит, этот процессор всегда занят с момента появления на его входе первой и до момента ухода с него последней задачи. В связи с этим время $T_2(\pi)$ выполнения произвольного расписания π можно представить в виде суммы времени t_1, необходимого для решения первой задачи, и времени решения последних m^*-х фрагментов критического пути для всех остальных задач. Заметив, что время решения первой задачи равно времени выполнения критического пути, получаем

$$T_2(\pi) = t_1 + \sum_{j=2}^{n} \tau_{m^*,j}^* = \sum_{i=1}^{m^*} \tau_{i,1}^* + \sum_{j=2}^{n} \tau_{m^*,j}^*.$$

Перенося для удобства анализа последнее слагаемое из первой суммы во вторую сумму, получаем

$$T_2(\pi) = \sum_{i=1}^{m^*-1} \tau_{i,1}^* + \sum_{j=1}^{n} \tau_{m^*,j}^*. \qquad (*)$$

Значение второй суммы представляет собой суммарное время, необходимое для решения пос-

ледних фрагментов критического пути для всех задач. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. В отличие от второй суммы первая зависит от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой суммы. Отсюда следует, что расписание, формируемое в алгоритме 2, является оптимальным для класса 2.

Так же как и в предыдущем случае, в данном алгоритме отсутствует перебор вариантов расписаний, а для оптимальности расписания достаточно лишь правильного выбора первой исполняемой задачи, которая должна быть наиболее быстро решаемой на всех процессорах критического пути, кроме, возможно, последнего. Упорядоченность же остальных задач не влияет на время выполнения расписания.

Рассмотрим задачу построения оптимального расписания для систем из класса 3.

Алгоритм 3

1. Выделим задачу J_s, которая удовлетворяет условию

$$\sum_{i=1}^{h-1}\tau^*_{i,s}=\min_j\left\{\sum_{i=1}^{h^*-1}\tau^*_{i,j}\right\}.$$

2. Выделим задачу J_p, которая удовлетворяет условию

$$\sum_{i=h^*+1}^{m^*}\tau^*_{i,p}=\min_{\substack{j\\ j\neq s}}\left\{\sum_{i=h^*+1}^{m^*}\tau^*_{i,j}\right\}.$$

3. Сформируем расписание $\pi_3=[J_s\ \tilde{\pi}\ J_p]$, где $\tilde{\pi}$ – произвольное расписание для $(n-2)$ задач, не содержащее задач J_s и J_p.

4. Сформируем расписание $\pi_4=[J_q\ \tilde{\pi}\ J_r]$, повторив пп. 1–3, но выполнив пп. 1 и 2 в другой последовательности.

5. Выберем среди расписаний π_3 и π_4 наилучшее расписание.

Покажем оптимальность этого алгоритма. Разобьем критический путь системы и, соответственно, всю систему на две части. В первую часть критического пути включим процессоры с первого по (h^*-1)-й, а во вторую – с h^*-го по m^*-й процессоры. Соответственно в первую часть системы включим процессоры ярусов с 1-го по (h^*-1)-й, а во вторую – все остальные процессоры. Очевидно, что первая часть системы соответствует системе из класса 2. В результате фрагмент каждой задачи перед решением на любом процессоре соответствующей части критического пути попадает в очередь. Поскольку то же самое справедливо и для h^*-го процессора, можно утверждать, что вторая часть системы соответствует системе из класса 1. Действительно, не только имеет место необходимая упорядоченность процессоров, но и фрагменты задач выходят на соответствующую часть критического пути без каких-либо дополнительных задержек так, как будто они все одновременно стоят в очереди к h^*-му процессору. Запишем время $T_3(\pi)$ выполнения произвольного расписания в рассматриваемой системе. Представим искомое время $T_3(\pi)$ как сумму двух величин. Первая величина – это время $t_n(1, h^*-1)$ решения всех n задач в первой части системы (от начала решения входного фрагмента первой задачи на первом процессоре критического пути до начала решения фрагмента n-й задачи на h^*-м процессоре), вторая величина – это время $t_n(h, m)$ решения n-й задачи на второй части системы:

$$T_3(\pi)=t_n(1,h^*-1)+t_n(h^*,m^*)\ .$$

Очевидно, что время $t_n(1, h^*-1)$ будет равняться времени решения $(n-1)$-й задачи на процессорах с 1-го по h^*-й, которое определяется выражением (*) и имеет вид

$$t_n(1,\ h^*-1)=t_{n-1}(1,\ h^*)=\sum_{i=1}^{h^*}\tau^*_{i,1}+\sum_{j=2}^{n-1}\tau^*_{h^*,j}.$$

Время $t_n(h^*, m^*)$ решения n-й задачи на второй части системы определяется очевидным выражением

$$t_n(h^*,\ m^*)=\sum_{i=h^*}^{m^*}\tau^*_{i,n}.$$

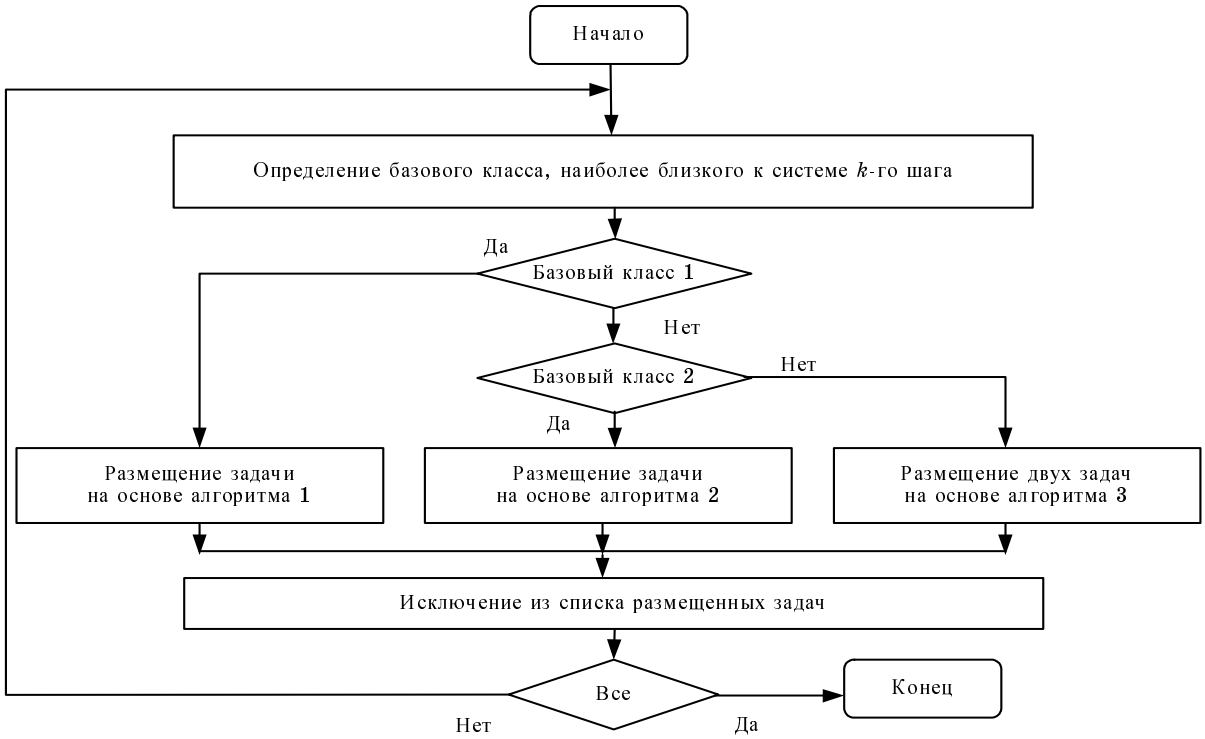
В результате для $T_3(\pi)$ получаем

$$T_3(\pi)=\sum_{i=1}^{h^*}\tau^*_{i,1}+\sum_{j=2}^{n-1}\tau^*_{h^*,j}+\sum_{i=h^*}^{m^*}\tau^*_{i,n}.$$

Перенося для удобства анализа последнее слагаемое из первой суммы и первое слагаемое из третьей суммы во вторую, получаем

$$T_3(\pi)=\sum_{i=1}^{h^*-1}\tau^*_{i,1}+\sum_{j=1}^{n}\tau^*_{h^*,j}+\sum_{i=h^*+1}^{m^*}\tau^*_{i,n}.$$

Значение второй суммы представляет собой суммарное время, затрачиваемое на решение соответствующих фрагментов всех задач на h^*-м процессоре. Очевидно, что для заданной очереди ее величина фиксирована и не зависит от выбора расписания. В отличие от второй суммы, первая и третья суммы зависят от выбора расписания. Причем оптимальным будет то расписание, которое характеризуется минимальным значением этой части выражения. Минимальность значения, в свою очередь, будет достигаться, если в расписании первой будет решаться задача, требующая наименьшего времени для первых (h^*-1) фрагментов, выполняемых на процессорах критического пути, а последней – задача, наиболее быстро решаемая на последних (m^*-h^*) процессорах критического пути. Поскольку одна и та же задача может удовлетворять обоим требованиям, ее надо попробовать разместить на каждой из этих позиций, а затем выбрать наилучшее из этих двух расписаний. Отсюда следует, что расписание, формируемое в алго-



■ ***Рис. 4.*** *Блок-схема субоптимального алгоритма построения расписаний*

ритме 3, является оптимальным для системы из класса 3.

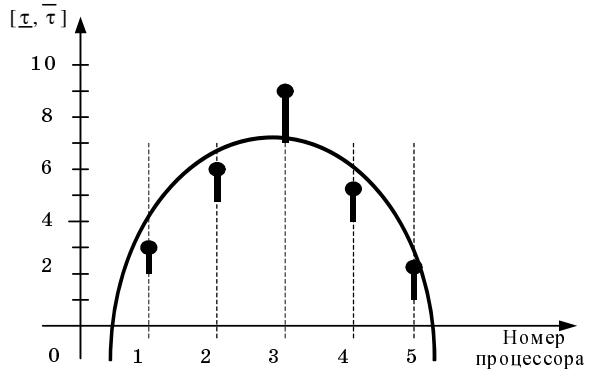
Отметим, что в данном алгоритме перебор расписаний также практически отсутствует. При построении расписания достаточно проанализировать лишь два возможных варианта, а для оптимальности расписания достаточно лишь правильного выбора первой и последней исполняемых задач. Упорядоченность же остальных задач не влияет на время исполнения расписания.

Субоптимальный алгоритм построения расписаний

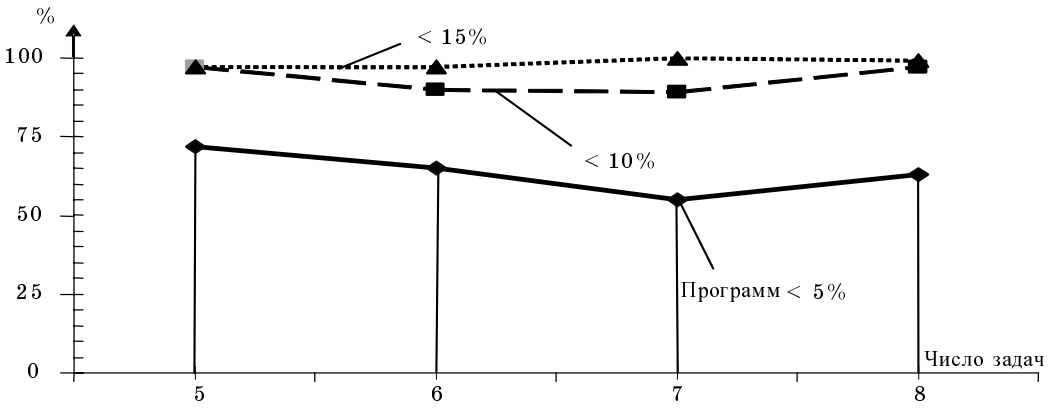
Конечно же, оптимальность приведенных выше алгоритмов имеет место лишь в случаях, когда рассматриваемая система полностью удовлетворяет требованиям какого-либо из базовых классов. На практике же чаще всего эти требования не выполняются, и тогда, в частности, утверждение о независимости времени выполнения расписания от варианта упорядочивания некрайних задач теряет силу. Напротив, представляется правдоподобным, что расхождение времен выполнения оптимального расписания и расписания с упорядочиванием лишь крайних задач будет тем меньше, чем меньше время выполнения расписания для всех некрайних задач. Таким образом, приходим к следующему рекурсивному алгоритму (рис. 4).

На первом шаге алгоритма для системы, характеризующейся множеством исполняемых задач мощностью n, на основе эвристического классификационного правила определяется базовый класс систем, к которому наиболее близка рассматриваемая система. Далее определяется одна из крайних (первая или последняя) задач будущего расписания (классы 1 и 2) либо обе крайние задачи (класс 3). Затем эти действия повторяются для $(n-1)$ задач (классы 1 и 2) или для $(n-2)$ задач (класс 3) и т. д. до тех пор, пока все задачи не будут размещены в расписании.

Классификация осуществляется путем выделения у рассматриваемой системы определяющего



■ ***Рис. 5.*** *Иллюстрация правила классификации систем*



■ ***Рис. 6.*** *Эффективность субоптимального алгоритма*

признака – наличия либо убывающей последовательности, либо возрастающей последовательности, либо состыкованных убывающей и возрастающей последовательностей. Значение определяющего признака вычисляется с использованием аппроксимации параболой верхних границ интервальной зависимости времени выполнения всех задач от номера процессора (рис. 5). На рисунке интервалы времен показаны отрезками жирных линий, а их верхние границы – черными кружками. При этом, если вершина параболы оказывается слева от интервала номеров процессоров, то система относится к классу 1, если справа,– то к классу 2, если внутри интервала,– то к классу 3.

Оценка эффективности приближенного алгоритма

Оценка эффективности предложенного алгоритма осуществлялась на основе компьютерного моделирования и в целях упрощения производилась авторами по отношению к классу конвейерных систем, составляющих важный подкласс иерархических систем. В основу использованного подхода было положено случайное генерирование примеров. При этом фиксировалось число процессоров ($m = 10$), образующих конвейер, а число задач варьировалось в интервале от 5 до 8. Для каждого значения n (число задач) из этого интервала генерировалось 100 примеров. Причем времена решения задач на каждом из процессоров формировались как случайные величины, равномерно распределенные на заданном интервале. Для каждого получаемого примера строилось расписание на основе предложенного алгоритма и путем полного перебора (оптимальное расписание). Эти расписания сравнивались между собой по относительной разности времен выполнения. Результаты моделирования приведены на рис. 6.

Результаты показывают, что, например, при $n = 5$ предлагаемый алгоритм примерно в 75% случаев строил расписания, которые уступали по длительности оптимальному расписанию не более 5%, а практически для всех примеров были построены расписания, уступающие оптимальному не более 10 %. Самым неблагополучным оказался случай при $n = 7$, когда лишь для 59% примеров были построены расписания, уступающие оптимальному не более 5%, а для 90% примеров были построены расписания, уступающие оптимальному не более 10%.

Заключение

В настоящей статье рассмотрен субоптимальный рекурсивный алгоритм составления расписаний в иерархических вычислительных системах, практически не требующий перебора вариантов и вследствие этого обладающий низкой временной сложностью. В его основе лежат несколько беспереборных оптимальных алгоритмов построения расписаний, каждый из которых ориентирован на некоторый свой в общем случае достаточно узкий базовый класс систем. Проведенное исследование эффективности алгоритма показало, что при его использовании в 90% случаях строится расписание, уступающее по длительности выполнения оптимальному расписанию не более 10%.

Литература

1. Теория расписаний и вычислительные машины: Пер. с англ. / Под ред. Э. Г. Коффмана. М.: Наука, 1984. 334 с.
2. **Левин В. И.** Структурно-логические методы исследования сложных систем с применением ЭВМ. М.: Наука, 1987. 304 с.
3. **Топорков В. В.** Модели распределенных вычислений. М.: Физматлит, 2004. 320 с.
4. **Колесов Н. В., Толмачева М. В.** Составление расписаний решения задач в конвейерных вычислительных системах // Информационно-управляющие системы. 2005. № 5. С. 16–21.