DOI 10.37882/2223-2966.2020.10.07

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ, МЕТОД И АЛГОРИТМ ПЛАНИРОВАНИЯ ЗАГРУЗКИ ПРОЦЕССОРОВ В МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМАХ КРИТИЧЕСКОГО НАЗНАЧЕНИЯ

MATHEMATICAL MODEL, METHOD AND ALGORITHM FOR PLANNING PROCESSOR LOADING IN CRITICAL MULTIPROCESSOR SYSTEMS

D. Borzov
D. Titov
S. Egorov
Iu. Sokolova

Summary: Considered are multiprocessor systems for critical purposes (observation, tracking, aiming, etc.), the need to create a mathematical model, method and algorithm for scheduling processor load. The number of tasks, the fact of their fulfillment (non-fulfillment) and control of the criteria for queuing have been performed. An example of drawing up a processor load plan is considered, an analysis of the influence of the number of processed operators on the number of processors required for processing is carried out with a simultaneous analysis of the impact on the overall performance of the system. The conclusion is made about the expediency of using specialized hardware.

Keywords: planning, loading, multiprocessor systems, method, algorithm, processor, critical systems, mathematical model.

Борзов Дмитрий Борисович
д.т.н., профессор, Юго-Западный государственный университет, Курск, Россия
borzovdb@kursknet.ru
Титов Дмитрий Витальевич
д.т.н., доцент, Юго-Западный государственный университет, Курск, Россия
amaizing2004@inbox.ru
Егоров Сергей Иванович
д.т.н., профессор, Юго-Западный государственный университет, Курск, Россия
sie58@mail.ru
Соколова Юлия Васильевна
к.т.н., ведущий специалист, АО «НПО Лавочкина», Химки
jv.sokolova@mail.ru

Аннотация: Рассмотрены мультипроцессорные системы критического назначения (наблюдение, слежение, прицеливание и т.д.), необходимость создания математической модели, метода и алгоритма планирования загрузки процессоров. Выполнен учет номеров задач, факта их выполнения (невыполнения) и контроль критериев постановки в очередь. Рассмотрен пример составления плана загрузки процессоров, проведен анализ влияния количества обрабатываемых операторов на требуемое для обработки число процессоров с одновременным анализом влияния на общую производительность системы. Сделан вывод о целесообразности использования специализированных аппаратных средств.

Ключевые слова: планирование, загрузка, мультипроцессорные системы, метод, алгоритм, процессор, критические системы, математическая модель.

Введение

Данная статья посвящена мультипроцессорным системам критического назначения (наблюдение, слежение, прицеливание и т.д.). Назначение выполняется на различных уровнях, что позволяет достичь повышения производительности всей системы в целом, которая может достигаться как за счет загрузки процессоров участками программ, независимыми как по управлению и по данным, так и за счет масштабирования алгоритма путем составления плана загрузки процессора, запланированными предварительно участками программ [1-4].

В критических мультипроцессорных системах, при количестве обрабатываемых участков от тысячи и выше, при программной реализации снижается коэффициент готовности системы из-за повышения общего времени реакции. Кроме того, мультипроцессорная система одновременно должна анализировать возможность выделения независимого исполнения линейных, условных и циклических фрагментов для ускорения суммарного времени выполнения [5,6]. При этом в случае составлении плана загрузки процессоров необходим параллельный учет номеров задач, факта их выполнения (невыполнения) и контроль критериев постановки в очередь. Также, в число задач, обрабатываемых мультипроцессорной системой, могут входить задачи компиляции, маршрутизации, распределения, реконфигурирования систем после сбоя и т.п. Отсутствие суммарного учета этих факторов сможет снизить эффективность, скорость и производительность всей системы в целом [7,8].

При планировании загрузки процессоров, в случае

применения мультипроцессорных критических систем, также необходим учет возможного расширения количества процессоров, что требует дополнительных затрат, повышающих оперативность реакции как с программной стороны, так и со стороны аппаратной части всех узлов системы [7,8].

В статье предложена математическая модель, метод и алгоритм планирования загрузки процессоров в мультипроцессорных системах критического назначения, приведен анализ временной сложности предложенного алгоритма [9]. Представлен анализ влияния количества обрабатываемых операторов на требуемое для обработки число процессоров с одновременным анализом влияния на общую производительность системы.

Анализ подходов к задаче планирования

Одним из путей повышения производительности критических динамических систем может быть динамическое планирование загрузки процессоров, являющееся одним из видов планирования заданий. В этом случае используется краткосрочное планирование, подходящее для быстрого реагирования на текущие процессы. При этом соответствующие алгоритмы используют такие свойства, как предсказуемость, минимизация ресурсов, равномерное распределение ресурсов системы и масштабируемость [9,10].

Алгоритмы планирования, определенные на различных классах и уровнях задач, эффективны и используют варианты приоритетного планирования и различные механизмы очередей, но, при этом, не учитывают структуру алгоритмов задач, время их выполнения и сложность. С учетом объемов задач, решаемых в мультипроцессорных системах, программное решение этого вопроса неприемлемо. В связи с этим актуальной является задача разработки математической модели, метода и алгоритма планирования загрузки процессоров в мультипроцессорных критических системах.

Существуют разные подходы к задаче планирования и теории расписаний, среди которых можно выделить алгоритм «первым пришел – первым обслужен» (FCFS), называемый невытесняющее планирование. Данному подходу характерно наличие большого среднего времени отклика [5-8], что неприемлемо для критических систем.

Альтернативой может быть алгоритм Round Robin (RR), в котором заданию выделяется фиксированный квант времени (около 10 – 100 миллисекунд), что влияет на производительность, и, следовательно, не подходит при использовании в критических системах.

Возможен вариант решения задачи планирования, когда процессы уже находятся в состоянии готовности. В этом случае предложено выбирать задачи не из начала очереди, а с минимальной длительностью исполнения.

Если известно время выполнения следующих процессов, находящихся в состоянии готовности, то существует возможность выбора для исполнения не процесса из начала очереди, а процесса с минимальной длительностью исполнения или «кратчайшей работы первой» (SJF).

Такое краткосрочное планирование может быть так же вытесняющим, т.е. учитывающим появление новых процессов в очереди, готовых к исполнению во время работы выбранного процесса. При невытесняющем SJF-планировании процессору предоставляется все необходимое ему время, независимо от других событий в системе.

В SJF-алгоритмах планирования продолжительность времени исполнения задания представлять невозможно. Это относится как к краткосрочному планированию, так и к пакетному режиму.

Вариантом гарантированного планирования может быть случай, когда пользователь имеет в своем распоряжении $\sim\frac{1}{N}$ часть процессорного времени, где N – количество заданий в вычислительной системе [3-4]. При таком планировании невозможно предсказать конечный поток задач. Частным случаем такого планирования является приоритетное планирование, когда каждому процессу задается определенное числовое значение – приоритет. В этом случае возможны те же недостатки, что и для гарантированного планирования.

Также существуют многоуровневые очереди, когда процессоры могут быть легко рассортированы по разным группам. Здесь для каждой группы процессоров, находящихся в состоянии готовности, создается своя очередь [3-4], и им приписываются фиксированные приоритеты. Внутри этих очередей для планирования может быть применен любой из рассмотренных алгоритмов. Например, приоритет очереди системных процессов устанавливается выше, чем приоритет очередей пользовательских процессов. В этом случае пользовательский процесс не будет выбран, пока существует хоть один готовый системный процесс.

Вариантом многоуровневой очереди может быть многоуровневая очередь с обратной связью, когда процесс не постоянно приписан к определенной очереди, а может перемещаться из одной очереди в другую в зависимости от своего поведения [3-4].

Рассмотренные алгоритмы планирования загрузки процессоров не учитывают объемов передаваемых данных и соответствующих времен, необходимых для

передачи, что особенно важно в мультипроцессорных системах критического назначения [10-12]. Поэтому возникает объективная необходимость разработки математической модели, метода и алгоритма планирования загрузки процессоров.

Математическая модель и метод планирования загрузки процессоров

В задачах теории расписаний (ТР) время *p* задается в условных единицах. С точки зрения вычислительного процесса, например: p_1=2 минуты, p_2=3 минуты, r_1=0 – нулевая минута, r_2=1 – первая минута и т.д. Эти параметры являются идентичными, поэтому эта единица далее опускается.

Срок D_j выполнения задания Z_j (i = 1, 2,..., m) не задан (D_j = +∞, D_j = +∞, j = 1, 2, . . ., n). Обозначим t_j – момент окончания выполнения задания j, т.е. $t_j = S_j + p_j$, где S_j – момент начала выполнения задания j. Тогда необходимо построить допустимое расписание, минимизирующее функцию:

$$\sum_{j=1}^{n} t_j \to \min. \tag{1}$$

Введем ограничения:

– процессоры $П_p$ в очереди планирования загрузки одинаковы по производительности и тактовой частоте. Поэтому время обслуживания процессора $П_{pj}$ для требования j не зависит от процессора, на котором задание будет обслужено;

– при освобождении процессора $П_p$, на него назначается невыполненное задание, готовое при этом к исполнению и находящееся в очереди для исполнения;

– функции маршрутизации, выделения независимых (не взаимодействующих заданий) выполняются на хост–процессоре, либо устройстве, взаимодействующим с хост–процессором мультипроцессорной системы.

Пусть $N = \{1, 2, ..., n\}$ – множество данных ($i \in N$, $j \in N$, $(\mathrm{i} \neq \mathrm{j})$, $\tilde{n}_{ij} = |\tau_i - \tau_j|$),

где $\tilde{n}_{ij} \geq 0$ разница во времени исполнения операции n_i и n_j.

Непересекающиеся подмножества множества N отвечает условию:

$$\bigcup_{i=1}^{k} S_i = N, \tag{2}$$

где k – это требование, предъявляемое к задаче N.

Разобьем множество N на непересекающиеся подмножества $S_1, S_2, ..., \mathrm{S}_k, k \geq 2$ так, что:

$$S_i \bigcap S_j = \varnothing, i \neq j, i, j = 1, 2, ..., k, \tag{3}$$

где $n_{\min} \leq |\mathrm{S}_i| \leq \mathrm{n}_{\max}$, $i = 1, 2, ..., k.$

Число подмножеств k ограничено условиями:

$$k \times n_{\min} \leq n \leq k \times n_{\max} \tag{4}$$

Для всех разбиений $S = (\mathrm{S}_1, \mathrm{S}_2, ..., \mathrm{S}_k)$ сопоставим значение функции $\varphi(\mathrm{S}) = \varphi(\mathrm{S}_1, \mathrm{S}_2, ..., \mathrm{S}_k)$. Тогда, необходимо найти разбиение $S^0 = S_1^0, S_2^0, ..., S_k^0$, такое, что

$$\varphi(\mathrm{S}^0) = \min_s \varphi(\mathrm{s}) \tag{5}$$

Тогда, функция $g_P(\mathrm{S}_i) \in \mathrm{S}_i$ описывает различие между данными, а при фиксированном значении $\varphi_P(S)$ – p равно

$$\varphi_P(\mathrm{S}_1, \mathrm{S}_2, ..., \mathrm{S}_k) = \max_{l \leq j \leq k} g_P(\mathrm{S}_i). \tag{6}$$

Тогда, на основе формализованной постановки, математическая модель задачи планирования загрузки процессоров выглядит следующим образом:

$$\begin{gathered} \sum_{j=1}^{n} t_j \to \min, \\ \varphi(\mathrm{S}^0) = \min_s \varphi(\mathrm{s}). \\ \varphi_P(\mathrm{S}_1, \mathrm{S}_2, ..., \mathrm{S}_k) = \max_{l \leq j \leq k} g_P(\mathrm{S}_i). \end{gathered} \tag{7}$$

С учетом (1-7) и исходя из представленных выше теоретических положений, введем дополнительные матрицы: матрицу времени, матрицу порядка и матрицу очередности, в которых предполагается отслеживание процесса составления плана загрузки процессоров.

Матрица времени Time хранит задачи, планируемые к загрузке:

$$Time = \left\| Time_{ij} \right\|, \tag{8}$$

где $i = \overline{1, m}$, $j = \overline{1, n}$, $m = n$.

В матрице Time вершины столбцов обозначают процессоры $Пр_{ij}$ мультипроцессорной системы, а в строках отложены номера задач *1, 2,..., m*, планируемых к составлению плана загрузки. На пересечении строки и столбца проставляется предполагаемое время выполнения задачи *n*.

Факт выполнения или невыполнения задачи n_m в множестве задач $\{N_{nm}\}$ заносится в матрицу порядка Exec:

$$Exec = \left\| Exec_{ij} \right\|, \tag{9}$$

где $i = \overline{1, m}$, $j = \overline{1, n}$, $m = n$.

В матрице порядка Exec вершины столбцов обозначают процессоры $Пр_{ij}$ мультипроцессорной системы, а в строках отложены задачи, подлежащие выполнению в

порядке возрастания своих номеров. Единица ставится в случае, если задача n выполнена. Таким образом, в конце работы алгоритма в каждой строке матрицы должен быть набор из значений единица, как факт выполнения всех задач в строке. С помощью матрицы Exec можно отслеживать факт выполнения задачи, назначения новой и выполнения текущей.

Очередь выполнения задач хранит матрица очередности Q

$$Q = \left\| Q_{ij} \right\|, \quad (10)$$

где $i = \overline{1,m}$, $j = \overline{1.n}$, $m = n$. В Q отображается очередь выполнения задач Z_i ($i = \overline{1,m}$). Столбцы матрицы обозначают процессоры $Пр_{ij}$ мультипроцессорной системы, а в строках – задачи, подлежащие выполнению в порядке возрастания номеров соответственно.

Тогда, с учетом представленных предположений предлагается метод конвейерного планирования загрузки процессоров в мультипроцессорных системах, основанный на (1-7), а также с использованием (8-10). Фрагменты программ предлагается назначать конвейерно независимо на процессоры мультипроцессорной системы.

Введем дополнительные ограничения:

1. Введем антирефлексивное, антисимметричное и транзитивное отношение следования операций $\nu \subseteq Z \times Z$, определяющее допустимый порядок реализации операций.
2. Потребуем, чтобы выполнялось следующее условие: $\forall z_a \in Z(p_i), z_b \in Z(p_i), a \neq b : z_a \vee z_b \vee z_b \vee z_a$ (любая пара операций, назначенных на процессор p_i, выполняется в заданном порядке *v*).
3. Операции выполняются в указанной последовательности.
4. Общая длительность всех операций вычисляется по формуле $T = \sum_{j=1}^{n} t_{ij}$.

Предлагаемый метод состоит из следующих шагов:

1. Получить множество подпрограмм, требующих составления плана загрузки на процессоры $Пр_1, Пр_2, \ldots, Пр_\alpha, \ldots, Пр_m (\alpha = \overline{1,m})$.
2. Выполнить алгоритм составления плана загрузки процессоров.
3. Получить тэг *Exec*, показывающий состояние готовности мультипроцессорной системы к получению нового задания.
4. Анализ битов доступности тэга *Exec*.
5. Если бит доступности поля *Work* равен единице, то прочитать номер свободного процессора и п.1, иначе п. 4. +

В результате анализа подходов планирования загрузки планирования процессоров была показана необходимость разработки математической модели, метода и алгоритма планирования загрузки процессоров в мультипроцессорных системах критического назначения.

Алгоритм планирования загрузки процессоров

Алгоритм планирования загрузки процессоров осуществляется следующей последовательностью шагов.

1. Задать $W = \left\| w_{ij} \right\|$ – матрицу подпрограмм, где $i = \overline{1, Пр}$ – это количество подпрограмм мультипроцессорной системы, а $j = \overline{1, Пр}$ – количество процессоров.
2. Поиск в строке $\min_{j}^{Пр} w_{ij}$ и вычитание его из всех ее элементов.
3. Определение в столбце $\min_{i}^{Пр} w_{ij}$ и вычисление всех ее элементов.
4. Найти в строке минимальный элемент и зафиксировать его. Если есть еще нули в строке, то вычеркнуть их.
5. П. 4 выполнить для всех строк W_{ij}.
6. Поиск в столбце нуля с последующей его фиксацией. Если есть еще нули в столбце, то вычеркнуть их.
7. П. 6 выполнить для всех W_{ij}.
8. Если в W_{ij} нет нулей, проводим минимальное количество горизонтальных и вертикальных пересечений через отмеченные нули, иначе п. 2-7.
9. Среди незачеркнутых прямыми чисел найти min и вычесть его из этих чисел. Прибавить min к числам, стоящим на пересечении прямых.
10. Повторять п. 2-13 пока в W_{ij} не будет один ноль.
11. Если $W_{ij} = 0$, то подпрограмма *i* назначается на процессор *j*.

В приведенном алгоритме на первом шаге задается исходная матрица подпрограмм. Здесь строки задают множество подпрограмм, для которых необходимо найти расписание загрузки, а столбцы – множество процессоров соответственно.

На втором шаге выполняется поиск минимального элемента в каждой строке исходной матрицы, после чего на третьем шаге происходит вычитание найденного элемента из всех элементов соответствующей строки. Третий шаг выполняет аналогичные действия со столбцами матрицы. Далее находим в строке минимальное значение и фиксируем его. В случае наличия в строке аналогичных нулей, необходимо их вычеркнуть.

Если на данном шаге в матрице присутствует количество нолей равное начальному количеству подпрограмм, то поиск завершен. В этом случае каждый ноль в матрице означает: строка означает номер подпрограммы, а столбец – это процессор, на которая она назначается.

Если количество нулей не равно числу подпрограмм, то необходимо перейти к шагу 8. На данном этапе необходимо провести минимальное число прямых, проходящих через все нули таблицы. На девятом шаге в полученной таблице выполняется поиск минимального числа, не проходящего ни через одну прямую. Далее вычитаем найденный минимум из всех чисел, через которые не проходит ни одна прямая, и добавляем его ко всем элементам, лежащим на пересечении двух прямых. Если после выполнения п. 9 не появилось числа нулей, равного числу подпрограмм, то повторять п. 10 и 11, пока не выполнится данное требование.

Для предложного алгоритма было проведено моделирование с помощью программы на языке C++. Его целью было определение времени решения задач при увеличении количества процессоров мультипроцессорной системы и определение необходимого и достаточного количества процессоров для обработки определенного количества информации.

Моделирование выполнялось при исходных данных *Пр* = {1…20} и *P* = {1…20}. Длительности операций для моделирования выбраны в диапазоне {1…16}, которые присваиваются произвольно. Результат моделирования представлен на рисунке 1.

График, представленный на рисунке 1, показывает уменьшение времени так, что $D = \sum_{i=1}^{n} t_i$. Из анализа графика (рис. 1) следует:

- уменьшение времени загрузки для первого и второго процессора (с 3000 мс до 1500 мс); снижение времени загрузки между вторым и четвертым процессором (с 1500 мс до 1000 мс);
- снижение времени на интервале между четвертым и десятым процессором;
- с десятого процессора падение времени не происходит, что не влияет на общую производительность системы. Этот показатель соответствует закону Амдала [1-4], утверждавшему, что объем решаемой задачи с адаптированным числом процессоров в системе, выделенным для её решения, остается неизменным.

Доля операций, которые должны выполнятся последовательно, обозначим f, где 0 ≤ f ≤ 1, а доля, приходящаяся для распараллеливания, обозначатся 1 – f. В исключительных случаях f=0, если задачи полностью распараллелены и f=1, если данные полностью последовательны. Распараллеливаемая часть программы равномерно распределяется по всем процессорам.

В результате получаем:

$$I_p = f \times I_s + \frac{(1-f)\times I_s}{n}. \quad (11)$$

Из (11) получаем формулировку Амдала, выражающую ускорение, которое может быть достигнуто для n процессоров

$$S = \frac{T_s}{T_p} = \frac{n}{1+(n-1)\times f}. \quad (12)$$

Формула (12) показывает, что если число процессоров стремится к бесконечности, то получаем:

$$\lim_{n\to\infty} s = \frac{1}{f}. \quad (13)$$

Из (13) следует, что если в программе 10% последовательных операций (f=0,1), то при любом количестве

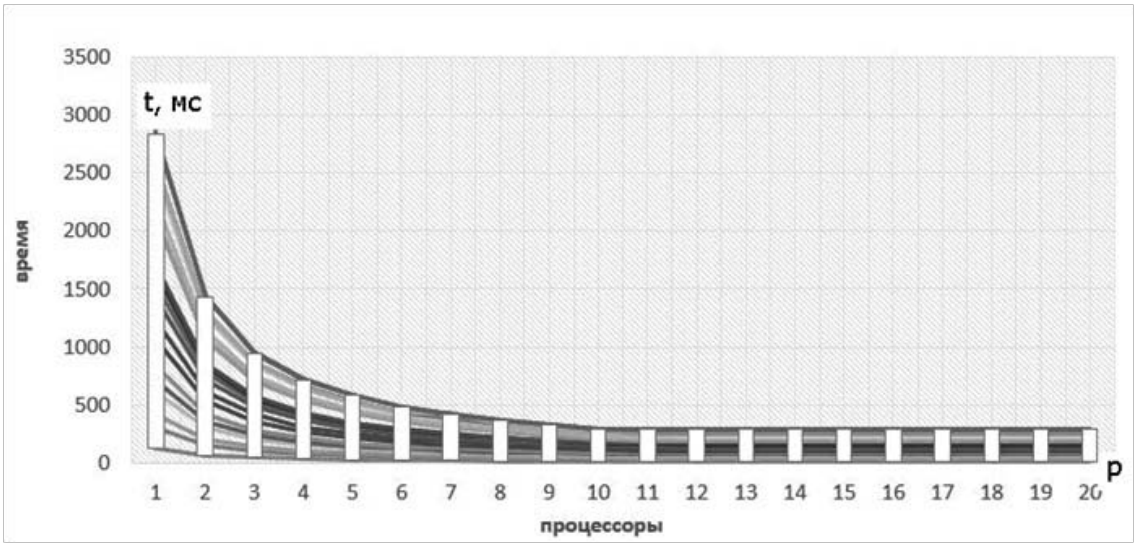

Рис. 1. График загрузки процессоров (доверительный интервал +/- 0,14)

процессоров ускорение более чем в 10 раз невозможно.

Влияния элементов одной ветви на количество процессоров показано на рисунке 2.

График, представленный на рисунке 2, демонстрирует необходимое и достаточное количество процессоров для обработки определенного количества информации. Так, при изменении количества команд с 10 до 500, увеличение количества процессоров возрастает в 6 раз.

Был проведен анализ эффективности алгоритма, основанного на методах теории расписаний, предложенного в данной статье. Результаты, полученные в результате моделирования, сравнивались с методом полного перебора, а также с результатом для примера задач большой размерности. Точность полученного результата определятся в соответствие с выражением

$$\varepsilon = \frac{t_{ex} - t_{opt}}{t_{opt}} \qquad (14)$$

где t_{opt} – минимальное время выполнения, полученное методом полного перебора, t_{ex} - время выполнения исследуемого алгоритма.

Первоначально проводилась случайная генерация графов заданий и длительностей задач. При такой генерации время выполнения заданий составляло 3–250. Для каждой точки интервала производилось по 500 экспериментов. В итоге, доверительный интервал составил 0,6% при доверительной вероятности 0,92.

При случайном формировании задания происходило формирование направленного графа при заданном числе вершин. Граф содержит несколько путей между всеми вершинами. Для каждой задачи планирования задавались различные длительности, равномерно распределенные в некотором интервале. Результат моделирования представлен на рисунке 3, на котором видно уменьшение точности по сравнению с алгоритмом полного перебора для пяти случайно сгенерированных примеров, включающих по 10 выборок с 5, 10, 15, 25, и 50 процессорами.

Из анализа графика, представленного на рисунке 3, видно, что среднее ухудшение предложенного алгоритма не превышает 7,7%, а существующий алгоритм показывает проигрыш в среднем 11,6 %.

Проведен анализ времени выполнения, предложенного и существующего алгоритма в зависимости от количества заданий (Рис. 4).

При построении графиков учитывалось, что вычислительная сложность предложенного алгоритма существенно меньше существующего алгоритма. При числе заданий более 250, время, затраченное на планирование, вырастает более чем в 100 раз, что наглядно демонстрируется (Рис. 4). В результате анализа зависимостей, представленных на рисунках 3 и 4, можно сделать вывод, что предложенный алгоритм позволяет поддерживать необходимую точность и быстродействие. В критических системах данного быстродействия недостаточно и для компенсации временных потерь более сложную и наибольшую по временным затратам часть алгоритма целесообразно перенести на специализированное устройство.

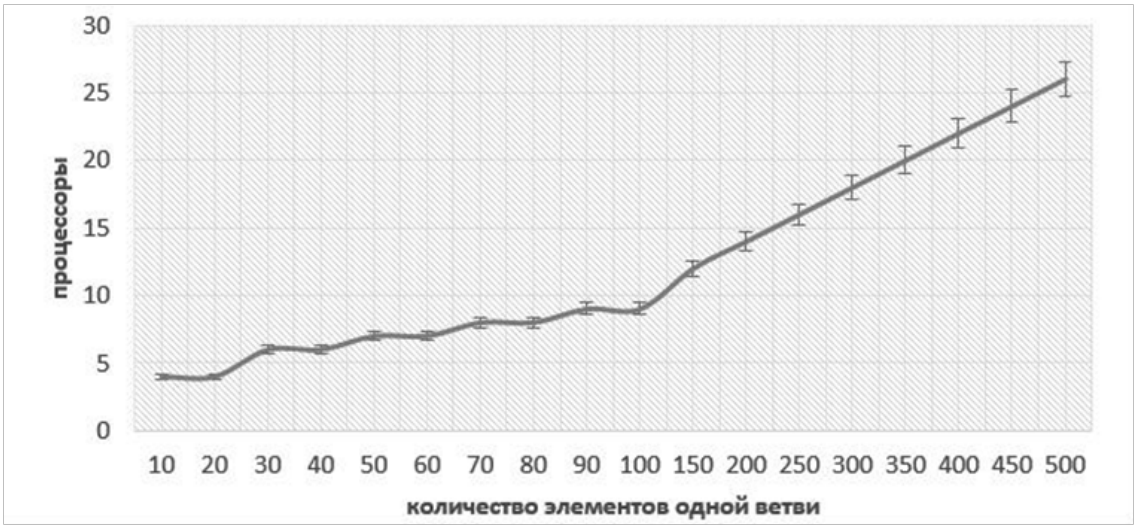

Рис. 2. График влияния элементов одной ветви на количество процессоров (доверительный интервал +/- 0,14)

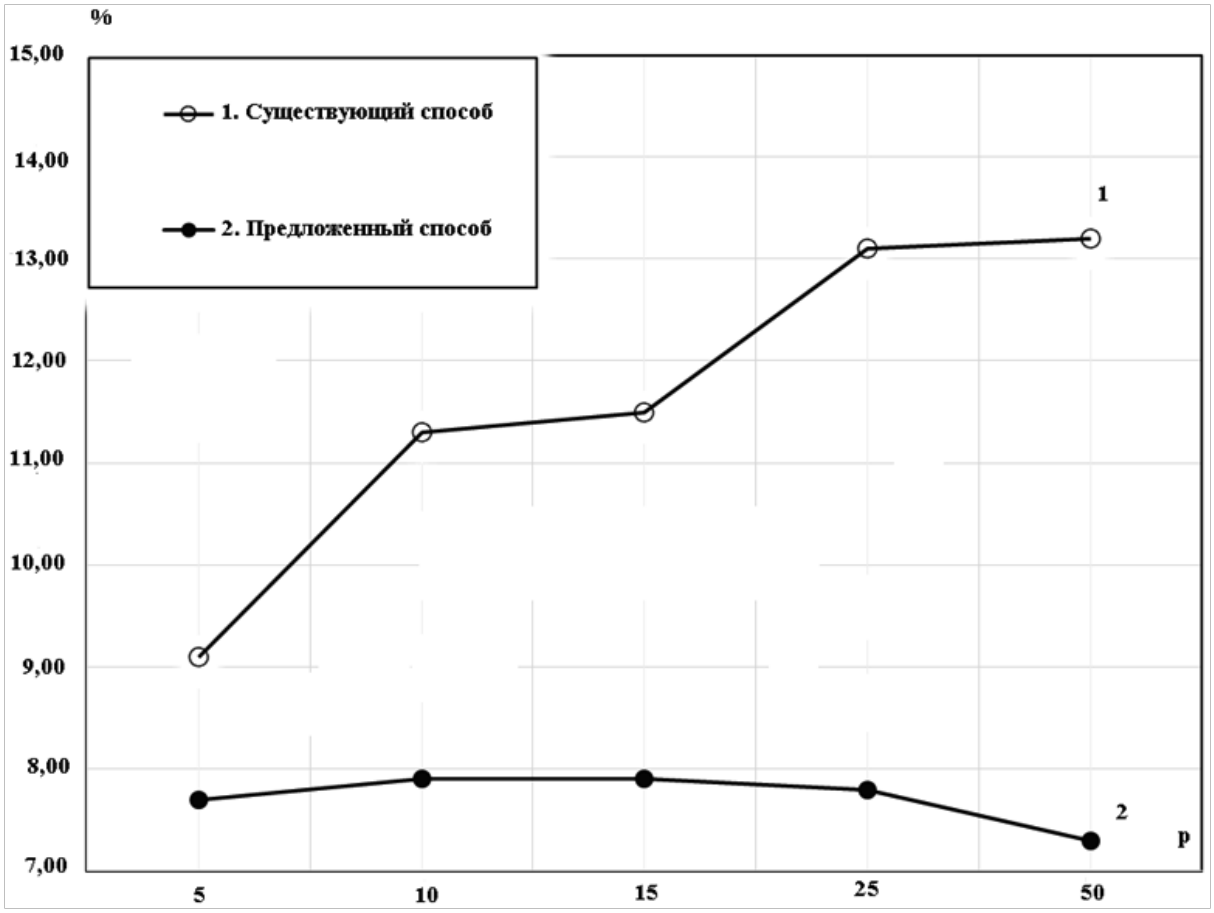

Рис. 3. График проигрыша алгоритмов на множестве случайных примеров (доверительный интервал +/- 0,6)

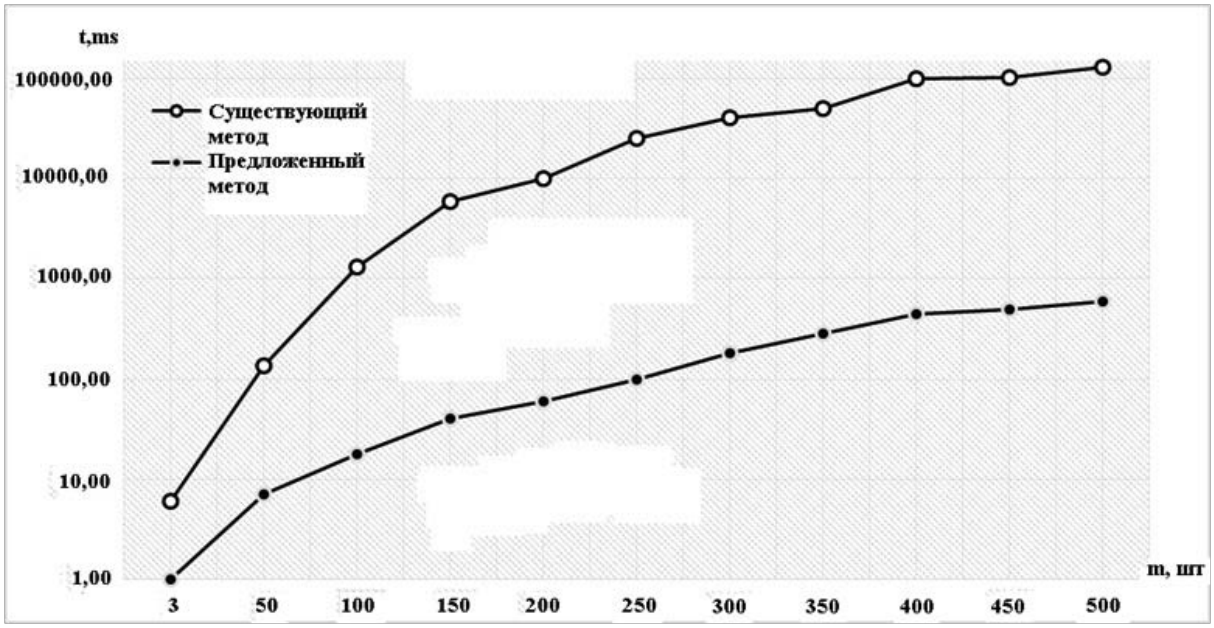

Рис. 4. Время выполнения, предложенного и существующего алгоритмов
в зависимости от количества заданий (доверительный интервал +/- 0,6)

Заключение

В работе была предложена математическая модель, метод и алгоритм планирования загрузки процессоров в мультипроцессорных системах критического назначения, ориентированного на аппаратную реализацию. Выполнено программное моделирование, в результате которого установлена зависимость влияния загрузки заданий от количества процессоров мультипроцессорной системы и зависимость выбора необходимого и достаточного их количества для обработки определенного количества информации. В результате было установлено, что получаемые показатели соответствуют закону Амдала, то есть возможен выбор адаптированного числа процессоров для определенного количества информации. В дальнейших исследованиях планируется проектирование структурной и функциональной схемы соответствующего устройства.

ЛИТЕРАТУРА

1. Основы архитектуры, устройство и функционирование вычислительных систем: Учебник / В.В. Степина. М.: КУРС: ИНФРА-М, 2018. 288 с.
2. Архитектура ЭВМ и вычислительные системы: учебник / В.В. Степина. – М.: КУРС: ИНФРА-М, 2017. 384 с.
3. Цилькер, Б.Я. Организация ЭВМ и систем: Учебник для вузов. СПб.: Питер, 2007. 668с.: ил- С. 481-492.
4. Воеводин Вл.В. Решение больших задач в распределенных вычислительных средах. // Автоматика и Телемеханика. 2007, N5, С. 32-45.
5. Zhang, L.; Wong, T.N. Solving integrated process planning and scheduling problem with constructive meta-heuristics. Inf. Sci. 2016, P. 340-341, 1-16.
6. Zhang, S.; Wong, T.N. Integrated process planning and scheduling: An enhanced ant colony optimization heuristic with parameter tuning. J. Intell. Manuf. 2014, 29, P. 1-17.
7. Морев Н.В. Сравнение алгоритмов планирования распределения задач для однородных распределенных вычислительных систем [Текст] / Н.В. Морев // Информационные технологии: Научно-технический и научно-производственный журнал. 2010. N 5. С. 43-46.
8. Antamoshkin A.N., Kazakovtsev L.A. Random Search Algorithm for the p-Median Problem // Informatica. 2013. V. 37(3). P. 267-278.
9. Борзов Д.Б., Басов Р.Г., Титов В.С. Аппаратные средства составления плана загрузки процессоров в мультипроцессорных системах критического назначения / Известия вузов. Приборостроение»., Том 62, №6. 2019. С. 517-523.
10. Борзов Д.Б., Ткачев П.Ю. Метод распараллеливания циклов со счетчиком. Известия Юго-западного государственного университета. Том 2 / №2., 2015. С. 104-108.
11. Борзов Д.Б., Дюбрюкс С.А., Титов В.С., Прилуцкий С.В.. Математическая модель выявления независимых параллельных участков последовательных программ. Нейрокомпьютеры: разработка, применение. 2009, №12. С. 37-41.
12. Борзов Д.Б., Титов В.С., Басов Р.Г. Метод и алгоритм планирования загрузки процессоров в мультипроцессорных системах критического назначения / Телекоммуникации. Ежемесячный научно-технический, информационно–аналитический и учебно-методический журнал. 2020, №1. С. 41-48.
13. Borzov D.B., Ilya I. Masyukov, Evgeny A. Titenko. Methods of Critical Systems Reconfiguration. International Russian Automation Conference (RusAutoCon).– 2018. С. 1-5.

© Борзов Дмитрий Борисович (borzovdb@kursknet.ru), Титов Дмитрий Витальевич (amaizing2004@inbox.ru),
Егоров Сергей Иванович (sie58@mail.ru), Соколова Юлия Васильевна (jv.sokolova@mail.ru).

Журнал «Современная наука: актуальные проблемы теории и практики»