2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

doi 10.26089/NumMet.v24r108 УДК 519.7

# Построение расписания для многоядерного процессора с учетом взаимного влияния работ

**А. В. Еремеев**
Институт математики имени С. Л. Соболева СО РАН,
Омский филиал, Омск, Российская Федерация
ORCID: 0000-0001-5289-7874, e-mail: eremeev@ofim.oscsbras.ru

**М. Ю. Сахно**
Институт математики имени С. Л. Соболева СО РАН,
Омский филиал, Омск, Российская Федерация
ORCID: 0000-0003-1548-0804, e-mail: sosnovskayamy@gmail.com

**Аннотация:** В статье рассматривается задача планирования работ на многоядерном процессоре с учетом их замедления при совместном выполнении. Предложена постановка задачи и модель частично целочисленного линейного программирования, доказана NP-трудность задачи при числе ядер, ограниченном константой. Результаты планировщика Intel TBB и жадного алгоритма сравниваются с результатами, полученными в соответствии с предложенной моделью с помощью пакета CPLEX. Проведенный эксперимент показал преимущества предложенного подхода по времени завершения всех работ.

**Ключевые слова:** многоядерный процессор, построение расписаний, частично целочисленное линейное программирование.

**Благодарности:** Работа выполнена в рамках государственного задания ИМ СО РАН (проект FWNF–2022–0020).

**Для цитирования:** Еремеев А.В., Сахно М.Ю. Построение расписания для многоядерного процессора с учетом взаимного влияния работ // Вычислительные методы и программирование. 2023. **24**, № 1. 115–126. doi 10.26089/NumMet.v24r108.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

# Multi-core processor scheduling with respect to the mutual influence of jobs

**Anton V. Eremeev**
Sobolev Institute of Mathematics SB RAS,
Omsk Department, Omsk, Russia
ORCID: 0000-0001-5289-7874, e-mail: eremeev@ofim.oscsbras.ru

**Maria Yu. Sakhno**
Sobolev Institute of Mathematics SB RAS,
Omsk Department, Omsk, Russia
ORCID: 0000-0003-1548-0804, e-mail: sosnovskayamy@gmail.com

**Abstract:** The paper deals with the problem of multi-core processor scheduling with respect to the mutual influence of jobs during their joint execution. A problem formulation and a model of mixed integer linear programming are proposed, the problem is shown to be NP-hard with the number of cores bounded by a constant. The results of the Intel TBB scheduler and the greedy algorithm are compared with the results obtained in accordance with the proposed model using the CPLEX package. The conducted experiment showed the advantages of the proposed approach in terms of the completion time of all jobs.

**Keywords:** multi-core processor, scheduling, mixed integer linear programming.

**Acknowledgements:** The work was funded in accordance with the state task of the IM SB RAS, project FWNF–2022–0020.

**For citation:** A. V. Eremeev and M. Yu. Sakhno, "Multi-core processor scheduling with respect to the slowdown of jobs during their joint execution," Numerical Methods and Programming. **24** (1), 115–126 (2023). doi 10.26089/NumMet.v24r108.

**1. Введение.** В работе исследуются задачи составления расписаний, возникающие, например, при разработке программы для выполнения на многоядерном процессоре. В этом случае необходимо составить расписание выполнения подпрограмм (работ) на ядрах процессора, учитывая, что они могут замедлять друг друга при совместном выполнении. Таким образом, скорость выполнения работы может меняться в зависимости от загрузки других ядер процессора. Например, разным работам необходимо передавать разный объем данных по шине данных, поэтому в случае одновременного выполнения нескольких работ может возникнуть конкуренция за шину данных и выполнение каждой из работ в этом случае может занять больше времени, чем в случае однопоточного выполнения. Задача планирования работ на многоядерном процессоре актуальна для производителей процессоров и компаний, разрабатывающих многопоточное программное обеспечение, поскольку учет взаимного влияния (замедления) работ позволяет повысить скорость работы программного обеспечения.

В литературе существует ряд подходов к планированию назначения работ на ядра процессора с учетом переменной длительности их выполнения. Как правило, такие задачи решаются с помощью быстрых эвристик, которые работают в онлайн-режиме, т.е. работы поступают последовательно и в каждый момент времени рассматривается только ограниченное количество работ. Эвристики планирования работ, предложенные в [1, 2] и некоторых других статьях, основаны на том принципе, что работы должны размещаться на ядрах процессора комплементарным образом, чтобы работы с наиболее различными потребностями в использовании ресурсов выполнялись одновременно (в [1, 2] под такими ресурсами подразумевается использование пропускной способности шины данных и использование кеша на разных уровнях).

Метод планирования работ, предложенный в [3], основан на *коэффициентах деградации* при совместном выполнении, равных увеличению времени выполнения приложения, когда работы в нем совместно используют кеш, по сравнению с запуском работ в одиночку. В случае двухъядерных процессоров работы могут быть представлены в виде вершин, соединенных ребрами, а веса ребер задаются суммой взаимных

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

деградаций совместного выполнения двух работ. Затем, при некоторых упрощающих предположениях, оптимальное расписание может быть найдено путем решения задачи совершенного паросочетания минимального веса. В случае большего числа ядер процессора показано, что задача является NP-трудной, и предложено несколько приближенных эвристических алгоритмов. Хотя методология из [3] и соответствующие алгоритмы были бы слишком дорогими для использования онлайн, они приемлемы для оценки качества других подходов.

Авторы [4] предлагают новый алгоритм совместного планирования работ с учетом справедливости, основанный на некооперативной игре, для уменьшения промахов кеша L2. Время выполнения работы варьируется в зависимости от того, какие работы выполняются на других ядрах того же процессора, поскольку разные комбинации работ приводят к разным уровням конкуренции в кеше. В [5] определенный объем кеша на процессоре совместно используется его ядрами, а скорость выполнения работы на процессоре зависит от того, какие работы размещены на других ядрах этого процессора. При этом количество работ $n$ равно количеству ядер всех процессоров и все работы запускаются в одно и то же время. В [5] доказано, что эта задача является NP-трудной в общей постановке, но для частного случая (двухъядерный процессор без миграции работ между ядрами) задача разрешима за $O(n^{2.5} \log\ n)$. Кроме того, в [5] для общей постановки представлен ряд алгоритмов, включая жадный алгоритм, для поиска оптимальных или приближенных решений.

В приложениях для планирования производства также важны постановки задач с переменным временем обработки. Например, в [6] рассматривается задача планирования производства кокса, в которой работы влияют на время выполнения других работ из-за повышения температуры производственной единицы. Авторы [6] строят модель целочисленного программирования, чтобы свести время к минимуму, предлагают несколько эвристик, включая генетический алгоритм, и сравнивают их производительность.

В теории планирования аналогичные постановки задач могут быть найдены в области планирования с контролируемым временем обработки. Модели и методы планирования для случая, когда прерывания допустимы, рассматриваются в [7]. Однако в настоящей работе мы рассматриваем постановку задачи без прерываний. Поскольку работы замедляют друг друга в основном из-за конкуренции за возобновляемые ресурсы (пропускная способность шины данных, кеш), мы ссылаемся на [8] и [9] для обзоров задач с возобновляемыми ресурсами, где распределение ресурсов может меняться с течением времени. Случай дискретных ресурсов рассматривается в [9], а непрерывные ресурсы рассматриваются в [8]. Последняя статья содержит постановку задачи, аналогичную рассматриваемой в настоящей статье, однако в [8] предполагается, что количество ресурсов, выделяемых для каждой работы, ограничено, но непрерывно и определяется планировщиком в каждый момент времени. В нашем случае скорость выполнения работ полностью определяется набором совместно запланированных работ на других ядрах (или машинах в традиционной терминологии планирования). Для такого случая в [10] предложен жадный алгоритм, в котором замедление работ рассчитывается на основе информации об использовании ими шины данных.

В недавней работе [11] авторы сосредоточились на назначении общих непрерывных ресурсов *ядрам*, в то время как назначение работ ядрам и их порядок фиксированы. В этом основные отличия от рассматриваемой в данной статье задачи. Авторы [11] показывают, что даже для работ единичной длительности поиск оптимального решения является NP-трудным, если число ядер является частью входных данных. Однако существует алгоритм с полиномиальным временем для любого постоянного числа ядер и работ единичной длительности.

В настоящей статье предложена математическая постановка задачи размещения работ по ядрам процессора и доказана ее NP-трудность при числе ядер, ограниченном константой. Также предлагается модель частично целочисленного линейного программирования (ЧЦЛП), использующая концепцию точек событий (см., например, [12]). Проведено сравнение результатов решений модели, найденных решателем CPLEX, с результатами, показанными библиотекой Intel TBB. Также приведены результаты сравнения значений целевой функции, полученных жадным алгоритмом, с оптимальными значениями целевой функции, полученными пакетом CPLEX при использовании алгоритма расчета скоростей выполнения работ, предложенного в [10].

Основные идеи статьи были представлены авторами в трудах конференции OPTIMA'2020 [10]. В настоящей работе математическая модель записана более экономно по расходу памяти, число рассматриваемых конфигураций снижено за счет использования транзитивного замыкания графа частичного порядка и проведен новый вычислительный эксперимент с целью сравнения предложенного метода с библиотекой Intel TBB и оценки практической значимости предложенного подхода.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

Статья имеет следующую структуру. В разделе 2 предложена постановка задачи. NP-трудность задачи показана в разделе 3. Модель ЧЦЛП приведена в разделе 4. В разделе 5 описан метод генерации тестовых данных, а в разделе 6 представлены результаты вычислительного эксперимента. Раздел 7 содержит заключение.

**2. Постановка задачи.** Неформально рассматриваемая задача состоит в том, чтобы запланировать выполнение работ на ядрах процессора с учетом их замедления при совместном выполнении. Ограничение на порядок выполнения этих работ задается как частичный порядок на множестве работ. Такая постановка не подразумевает, что предшествующие работы как-либо влияют на последующие, например посредством кеша или температуры процессора. Цель состоит в том, чтобы свести к минимуму время завершения последней работы.

**Математическая постановка задачи.** Имеется множество работ $J = \{1, \ldots, n\}$ и $m$ ядер процессора. Работы выполняются непрерывно и не меняют ядро в процессе выполнения. На одном ядре не может выполняться более одной работы.

Для каждой работы $j \in J$ известно количество единиц времени $s_j$, необходимое ей для полного выполнения в идеальных условиях (т.е. при условии, что вместе с ней не выполняется других работ).

Введем определения. *Конфигурация* — это набор работ, выполняющихся одновременно на разных ядрах с учетом частичного порядка на множестве работ (конфигурация не может содержать пару работ, в которой одна из работ должна быть выполнена раньше другой в соответствии с частичным порядком с учетом транзитивности) и ограничений на количество ядер. Множество всех конфигураций обозначим $C$. Положим, что в нулевой конфигурации ни одна работа не выполняется. На множестве $C$ также задан частичный порядок, составленный на основе частичного порядка на множестве работ. То есть конфигурация $c_1$ предшествует конфигурации $c_2$ в том случае, если хотя бы одна из работ конфигурации $c_1$ должна выполниться раньше хотя бы одной работы из конфигурации $c_2$ . Заметим, что число элементов в множестве $C$ ограничено сверху $\sum\limits_{i=0}^{m} C_n^i$.

*Скоростью выполнения работы* $j$ в конфигурации $c \in C$ будем называть отношение времени выполнения работы $j$ в идеальных условиях ко времени полного выполнения работы $j$, если бы $j$ все это время выполнялась в конфигурации $c$. Скорость работы зависит от конфигурации, в которой она выполняется. На протяжении каждой конфигурации скорость выполнения всех работ считается постоянной.

Итак, для каждой конфигурации $c \in C$ известно, из каких работ она состоит. Для каждой работы $j$ в конфигурации $c$ известна скорость ее выполнения $v_{jc}$.

Необходимо составить такое расписание выполнения работ на ядрах процессора, что общее время завершения работ $C_{\max}$ минимально.

**3. Труднорешаемость задачи.** Поставленная задача является NP-трудной при числе ядер $m \geqslant 2$, ограниченном константой. Частным случаем ее распознавательной версии для предложенной постановки является NP-полная задача МНОГОПРОЦЕССОРНОЕ РАСПИСАНИЕ [13]. Приведем постановку этой задачи.

*Заданы множество работ $T$, число процессоров $w \in \mathbb{Z}^+$, длительность $l(t) \in \mathbb{Z}^+$ для каждой $t \in T$ и общий директивный срок $D \in \mathbb{Z}^+$. Существует ли $w$-процессорное расписание для работ из $T$, которое удовлетворяет общему директивному сроку $D$?*

В [13] также доказано, что задача МНОГОПРОЦЕССОРНОЕ РАСПИСАНИЕ остается NP-полной при $w = 2$.

**Утверждение 1.** *Задача планирования работ на многоядерном процессоре с учетом их взаимного влияния является NP-трудной при числе ядер $m \geqslant 2$, ограниченном константой*

**Доказательство.** Задача МНОГОПРОЦЕССОРНОЕ РАСПИСАНИЕ является частным случаем распознавательной версии задачи планирования работ на многоядерном процессоре с учетом их взаимного влияния в случае, если (i) работы друг друга попарно не замедляют, (ii) отсутствует отношение частичного порядка и (iii) множество работ $T$ равно множеству работ $J$, число процессоров — количеству ядер, а длительность работы соответствует необходимому количеству единиц времени для выполнения работы. Также положим количество ядер равным 2. В этом случае количество конфигураций есть $1 + n + \dfrac{n(n-1)}{2}$, где первое слагаемое соответствует конфигурации, не содержащей ни одной работы, второе слагаемое — количество конфигураций, содержащих ровно по одной работе, а третье слагаемое — количество конфигураций, состоящих из двух работ. Следовательно, размер входа рассматриваемой за-

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

дачи ограничен полиномом от размера входа задачи МНОГОПРОЦЕССОРНОЕ РАСПИСАНИЕ. Заметим, что это верно и для конечного числа процессоров $w \geqslant 2$. Действительно, в этом случае количество конфигураций равно $\sum\limits_{i=0}^{w} C_{|T|}^{i}$, что также полиномиально ограничено от $w$ и $|T|$. Таким образом, задача планирования работ на многоядерном процессоре с учетом их взаимного влияния NP-трудна при числе ядер $m \geqslant 2$, ограниченном константой, что и требовалось доказать.

Стоит отметить, что, как правило, расписания строятся для процессора с известной конфигурацией, поэтому $m$ ограничено константой.

**4. Математическая модель.** Рассмотрим математическую модель. Определим понятие точки событий, аналогичное введенному в [12]. В настоящей работе точка событий — это номер интервала времени, в котором выполняется одна конфигурация.

Введем следующие параметры:

1. $q_{jc} = 1$ тогда и только тогда, когда работа $j$ выполняется в конфигурации $c$, 0 иначе.
2. $a_{c_1c_2} = 1$ тогда и только тогда, когда конфигурация $c_2$ должна выполняться после конфигурации $c_1$, 0 иначе.
3. $T_{\max}$ — оценка сверху продолжительности выполнения конфигурации в точке событий.

Обозначим через $K = \{0, 1, 2, \ldots, e\}$ множество точек событий, где $e$ является максимальной точкой событий, и введем переменные:

1. $t_{kc}$ — продолжительность выполнения конфигурации $c$ в точке событий $k$.
2. $x_{kc} = 1$ тогда и только тогда, когда в точке событий $k$ выполняется конфигурация $c$, 0 иначе. Для единообразия математической модели положим, что нулевая конфигурация выполняется в нулевой точке событий: $x_{00} = 1$ и $x_{0c} = 0$, $c \in C$.
3. $y_{jk} = 1$ тогда и только тогда, когда работа $j$ начала выполняться в точке событий $k$, 0 иначе.

Тогда математическая модель может быть записана следующим образом:

$$C_{\max} = \sum_{k \in K} \sum_{c \in C} t_{kc} \to \min, \tag{1}$$

$$t_{kc} \geqslant 0, \qquad k \in K,\ c \in C, \tag{2}$$

$$t_{kc} \leqslant x_{kc} T_{\max}, \qquad k \in K,\ c \in C, \tag{3}$$

$$\sum_{c \in C} x_{kc} = 1, \qquad k \in K, \tag{4}$$

$$\sum_{k \in K} \sum_{c \in C} t_{kc} v_{jc} = s_j, \qquad j \in J, \tag{5}$$

$$\sum_{c \in C} x_{kc} q_{jc} - \sum_{c \in C} x_{k-1,c} q_{jc} \leqslant y_{jk}, \qquad j \in J,\ k \in K, \tag{6}$$

$$\sum_{k \in K} y_{jk} = 1, \qquad j \in J, \tag{7}$$

$$\sum_{k \in K} (k+1) x_{kc_1} \leqslant (1 - \sum_{k \in K} x_{kc_1}) e + \sum_{k \in K} k x_{kc_2}, \quad c_1, c_2 \in C,\ a_{c_1c_2} > 0, \tag{8}$$

$$x_{kc} \in \{0, 1\},\ y_{jk} \in \{0, 1\}, \qquad j \in J,\ k \in K,\ c \in C. \tag{9}$$

Целевая функция (1) задает критерий минимизации момента окончания выполнения всех конфигураций. Неравенство (2) гарантирует неотрицательность продолжительности выполнения конфигурации $c$ в точке событий $k$, а неравенство (3) — что продолжительность выполнения конфигурации $c$ будет равна нулю только в случае, если эта конфигурация не выполняется в точке событий $k$, иначе она будет не больше $T_{\max}$. Равенство (4) выражает, что в каждой точке событий выполняется одна и только одна конфигурация, а равенство (5) — что каждая работа должна выполниться полностью. Неравенство (6) и

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

равенство (7) гарантируют непрерывность выполнения работ. Неравенство (8) задает частичный порядок между конфигурациями. Условие (9) описывает область определения переменных $x_{kc}$ и $y_{jk}$.

Отметим, что в расписаниях, построенных предложенной моделью, может возникать необходимость начать какую-либо работу с некоторой задержкой после начала или окончания другой работы. Например, как показано на рис. 1 сверху, работа $j_2$ должна начинаться через некоторое время после начала работы $j_1$. Если же начать выполнение работ $j_1$ и $j_2$ одновременно, то может оказаться, что они сильно замедляют друг друга, как показано на рис. 1 снизу. В этом случае общее время завершения работ больше, чем в случае, если бы задержка была реализована. Таким образом, возможность задержки начала работ в некоторых случаях позволяет найти лучшие решения.

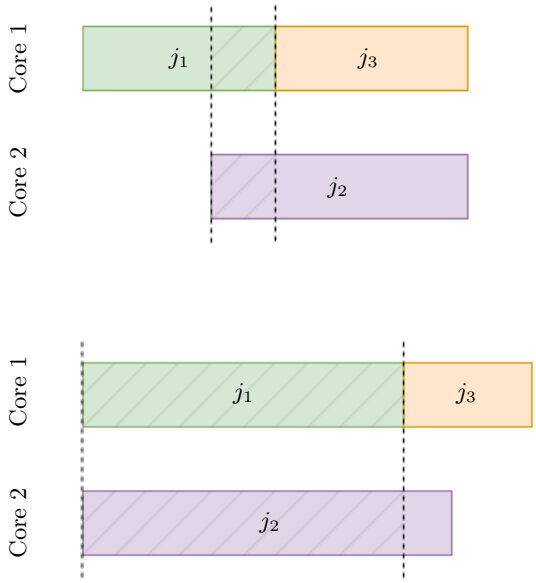


Рис. 1. Выполнение работ с задержкой (сверху) и без задержки (снизу)

Fig. 1. Execution of jobs with delay (at the top) and without delay (at the bottom)

**Утверждение 2.** *Для нахождения оптимального расписания выполнения $n$ работ необходимо не более чем $2n + 1$ точек событий.*

**Доказательство.** Заметим, что каждая точка событий соответствует смене конфигураций, а смена происходит только тогда, когда началась или закончилась какая-либо работа (или несколько работ). Предположим, что в каждой точке событий начинается или заканчивается только одна работа, тогда легко увидеть, что в таком случае необходимо $2n$ точек событий. Учтем также, что нам необходима нулевая точка событий — точка, в которой ни одна конфигурация не выполняется. Отсюда получаем, что в случае, когда никакие две работы не начинаются и не заканчиваются одновременно, количество точек событий равно $2n + 1$. В любом другом случае потребуется меньшее количество точек событий, что и требовалось доказать.

Ввиду утверждения 2 далее полагаем $e = 2n$.

Приведенная выше математическая модель была записана в системе моделирования GAMS, и для ее решения применялся пакет CPLEX.

**5. Методы генерации данных.** Все вычисления, описанные в разделах 5 и 6, проводились на ЭВМ Intel Core i7-8565U CPU 1.80 ГГц, оперативная память 16 ГБ. Операционная система Windows 10 версии 1909. Количество потоков, используемых для вычислений, не превосходило количества физических ядер процессора, поэтому влияние других процессов и самой операционной системы можно считать незначительным. Технологии Turbo Boost и Hyper-threading были отключены, чтобы исключить замедление работы процессора из-за перегрева и другие факторы на время выполнения работ. Перечисленные технологии позволяют ускорить выполнение программ в некоторых случаях: Turbo Boost повышает производительность процессора, автоматически увеличивая базовую частоту ядер процессора, если мощность, потребляемый ток и температура не превышают максимальных значений; Hyper-threading обеспечивает более эффективное использование ресурсов процессора, позволяя выполнять несколько потоков на каждом ядре.

Все описанные в этом разделе программы были реализованы на языке C++. Для проведения вычислительного эксперимента в качестве работ выступает многократное повторение одной из следующих процедур, взятых из библиотеки Intel MKL (Math Kernel Library):

- копирование вектора;
- суммирование абсолютных значений элементов вектора;
- вычисление суммы одного вектора с результатом произведения другого вектора на число;
- QR-разложение матрицы.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

Многократное повторение каждой из процедур вызвано тем, что при расчете скоростей работ из одной конфигурации нам нужно останавливать выполнение всех работ, если хотя бы одна работа из этой конфигурации завершилась, при этом необходимо знать, какая доля выполнена от каждой из не завершившихся полностью работ. Эта доля рассчитывается как отношение выполненных повторов процедуры к необходимому количеству (в нашем случае это 100). Выбор таких процедур обусловлен тем, что они поразному замедляют друг друга при совместном выполнении, при этом не имеет значения, какие факторы на это влияют (кеш, шина данных, размещение в ОЗУ).

Для генератора необходимо указать следующие входные данные:

1. Количество работ, расписание для которых нужно составить. Используемые значения: 4, 6, 7, 8, 10. При решении задач для большего количества работ пакет CPLEX не удовлетворял ограничениям по памяти и/или времени.
2. Частичный порядок, который нужно сгенерировать для работ. Используемые значения: тривиальный частичный порядок (т.е. без связей между работами), построенный случайным образом (с вероятностью 0.5 определяется, должна ли работа $j_1$ выполняться после работы $j_2$; чтобы избежать циклов, просматриваются только пары работ $(j_1, j_2) : j_1 > j_2$), бинарное дерево и один-ко-многим-к-одному (сначала одна задача, после нее много задач, после них вновь одна).
3. Количество ядер. Используемые значения: 2, 3, 4.

С целью сокращения числа рассматриваемых конфигураций строилось транзитивное замыкание графа частичного порядка и в качестве допустимых конфигураций рассматривались только те, в которых работы не связаны дугами в транзитивном замыкании.

Для возможности генерировать данные автоматизированно в генераторе можно задать следующие значения:

1. Количество вариантов с заданными параметрами, которые нужно сгенерировать. Для рассматриваемой задачи это значение равно 16.
2. Ограничения на размеры входных данных для процедур. Для векторов использовались значения от 10 до 70 млн элементов, для матриц — от 1000 до 1300. Такие размерности обусловлены тем, что работы не должны длиться слишком мало (меньше одной секунды), поскольку это может привести к большим погрешностям измерений, или слишком долго (больше 10 с), так как иначе измерения будут занимать слишком много времени. К тому же при таких размерностях входных данных их объем многократно превышает размер кеша, что соответствует ситуации, когда предшествующие процедуры практически не влияют на последующие посредством кеша.
3. Число повторных запусков работы или набора работ (т.е. конфигурации) для получения усредненных значений времени выполнения. Для рассматриваемой задачи число повторных запусков равнялось 5.
4. Максимально возможное число компонент связности графа предшествования, определяющее частичный порядок в моделях типа бинарное дерево и один-ко-многим-к-одному. Фактическое количество компонент связности выбирается в генераторе случайным образом.

Для каждой работы генератор вычисляет время выполнения в идеальных условиях, т.е. когда скорость этой работы максимальна. Далее перебираются все возможные конфигурации и выполняются те из них, которые не запрещены частичным порядком. Метод расчета скоростей работ в одной конфигурации заключается в следующем. Создаются потоки в количестве, равном количеству работ в конфигурации, и каждому потоку отдается одна работа на выполнение. Потоки запускаются одновременно, выполняя переданные работы в цикле. После того как один из них закончил выполнение, останавливаются все другие (с помощью некешируемого флага). После этого рассчитываются скорости выполнения для каждой из работ по следующей формуле: $standardTime \times completionShare / configurationTime$, где $standardTime$ — время выполнения работы в идеальных условиях, $configurationTime$ — время выполнения текущей конфигурации, $completionShare$ — отношение количества фактически выполненных повторов процедуры в этой работе к заданному количеству повторов.

Стоит отметить, что для удобства тестирования в качестве количества единиц времени, необходимого каждой работе для полного выполнения, было взято время выполнения в идеальных условиях в миллисекундах. Благодаря этому ответ получается так же в миллисекундах и найденное оптимальное решение легко сравнить с реальным выполнением работ согласно построенному расписанию.

Таким образом, всего было построено 960 тестовых примеров[1].

[1]Примеры доступны по ссылке https://github.com/mysosnovskaya/multicore-processor-scheduling-test-data.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

**6. Вычислительный эксперимент.** Было произведено сравнение времени выполнения работ согласно расписаниям, построенным с помощью пакета CPLEX в соответствии с моделью ЧЦЛП из раздела 4, и библиотекой Intel TBB [14]. Библиотека Intel TBB используется в современном многопоточном программном обеспечении. Она предоставляет онлайн-планировщик, который не учитывает замедления работ при совместном выполнении. Для справедливого сравнения выполнение работ согласно расписаниям, полученным с помощью модели ЧЦЛП, осуществлялось средствами библиотеки Intel TBB: полученные расписания преобразовывались в граф предшествования работ, который однозначно задавал порядок выполнения. При необходимости начать выполнение одной работы с некоторой задержкой после окончания другой работы между ними добавлялась *фиктивная* работа, которая приостанавливала текущий поток на время, равное времени задержки. Также значения целевых функций, полученных с помощью жадного алгоритма [10], были сопоставлены со значениями целевых функций, полученных с помощью пакета CPLEX, если в модели ЧЦЛП используются скорости выполнения работ, рассчитанные по алгоритму из [10].

Введем следующие обозначения для результатов экспериментов: $C^{\mathrm{opt}}_{\max}$ — оптимальное значение целевой функции, найденное пакетом CPLEX, $\tilde{C}^{\mathrm{opt}}_{\max}$ — реальное время завершения работ, выполненных согласно оптимальным расписаниям, $\tilde{C}^{\mathrm{tbb}}_{\max}$ — время завершения работ библиотекой Intel TBB, $C^{\mathrm{gr}}_{\max}$ — значение целевой функции, найденное жадным алгоритмом.

На рис. 2 приведена гистограмма относительного отклонения $d$ времени выполнения работ, рассчитанного согласно модели ЧЦЛП, от реального времени выполнения работ, где $d = \dfrac{\tilde{C}^{\mathrm{opt}}_{\max} - C^{\mathrm{opt}}_{\max}}{C^{\mathrm{opt}}_{\max}} \cdot 100\%$.

Отклонение большинства из построенных расписаний (98%) попадает в отрезок от $-11\%$ до 4%. Это связано с тем, что при проведении вычислительного эксперимента операционная система может запускать фоновые процессы, которые также могут замедлять выполнение работ.

В табл. 1 представлено среднее время работы пакета CPLEX для разных типов частичного порядка и разного количества работ. Время работы пакета CPLEX было ограничено 20 мин. Те примеры, для которых решение за отведенное время не было найдено, не учитывались в результатах таблиц 1, 2 и 3. Быстрее всего пакет CPLEX находит решения для работ со случайным частичным порядком, поскольку этот тип частичного порядка, как правило, имеет большее количество связей, чем другие, а медленнее всего — для работ, на множестве которых отношение частичного порядка не задано.

Преимущество решений, полученных с помощью модели ЧЦЛП, над Intel TBB обозначается через $\Delta$ и вычисляется как среднее величины $\dfrac{\tilde{C}^{\mathrm{opt}}_{\max} - \tilde{C}^{\mathrm{tbb}}_{\max}}{\tilde{C}^{\mathrm{opt}}_{\max}} \cdot 100\%$. Через $r = \dfrac{\tilde{C}^{\mathrm{opt}}_{\max}}{\tilde{C}^{\mathrm{tbb}}_{\max}}$ обозначим отношение времени выполнения работ согласно расписаниям, полученным с помощью модели ЧЦЛП, ко времени выполнения работ библиотекой Intel TBB.

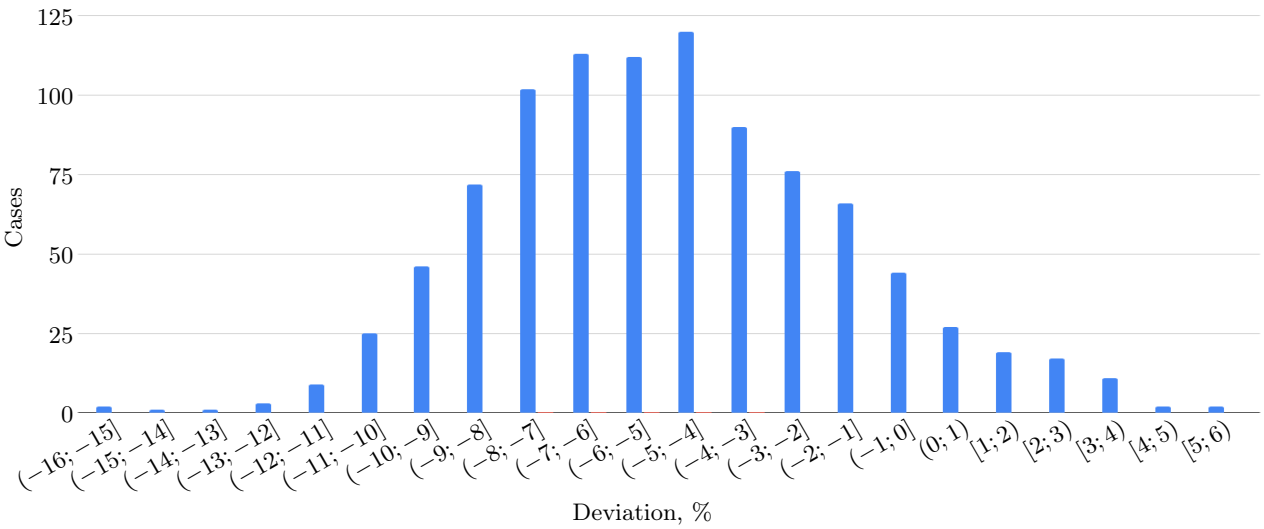


Рис. 2. Гистограмма относительного отклонения времени выполнения работ, рассчитанного согласно модели ЧЦЛП, от реального времени выполнения

Fig. 2. Histogram of relative deviation of the minimum completion time from the real completion time

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

Таблица 1. Среднее время работы пакета CPLEX

Table 1. Average time of CPLEX package

| | 4 jobs | 6 jobs | **7 jobs** | **8 jobs** | **10 jobs** |
|---|---|---|---|---|---|
| Trivial order | 0.4 s | 4.3 min | 13 min | 16 min | 15.5 min |
| One to many to one | 0.2 s | 3 s | 26 s | 6.3 min | 14.8 min |
| Random order | 0.2 s | 3.6 s | 18 s | 32 s | 3.6 min |
| Bitree order | 0.2 s | 4 s | 1.5 min | 7.2 min | 16 min |

Таблица 2. Среднее преимущество $\Delta(\%)$ решений, найденных пакетом CPLEX, по сравнению с TBB

Table 2. Average advantage $\Delta(\%)$ of solutions, found by CPLEX package, comparing to Intel TBB

| | **4 jobs** | **6 jobs** | **7 jobs** | **8 jobs** | **10 jobs** |
|---|---|---|---|---|---|
| Trivial order | 6 | 13.1 | 9.2 | 11.9 | 11.5 |
| One to many to one | 1.1 | 9.4 | 6.9 | 11.1 | 11.5 |
| Random order | 2.9 | 4 | 3.5 | 4.6 | 4 |
| Bitree order | 3.7 | 4.9 | 4.8 | 9.4 | 10 |

Результаты тестирования, приведенные в табл. 2 и на рис. 3, показали, что в среднем по каждой серии примеров общее время завершения всех работ меньше, если их выполнять в соответствии с расписаниями, полученными с использованием модели ЧЦЛП. Среднее преимущество по всем сериям составляет 7.2%. Максимальное среднее преимущество по серии составляет 13.1%. Максимальное преимущество по отдельно взятому примеру — 44.9%. Наименьшее преимущество наблюдается для серий примеров, которые были сгенерированы с использованием случайного частичного порядка. Это объясняется тем, что данный тип частичного порядка обладает наибольшим количеством связей между работами и, соответственно, в

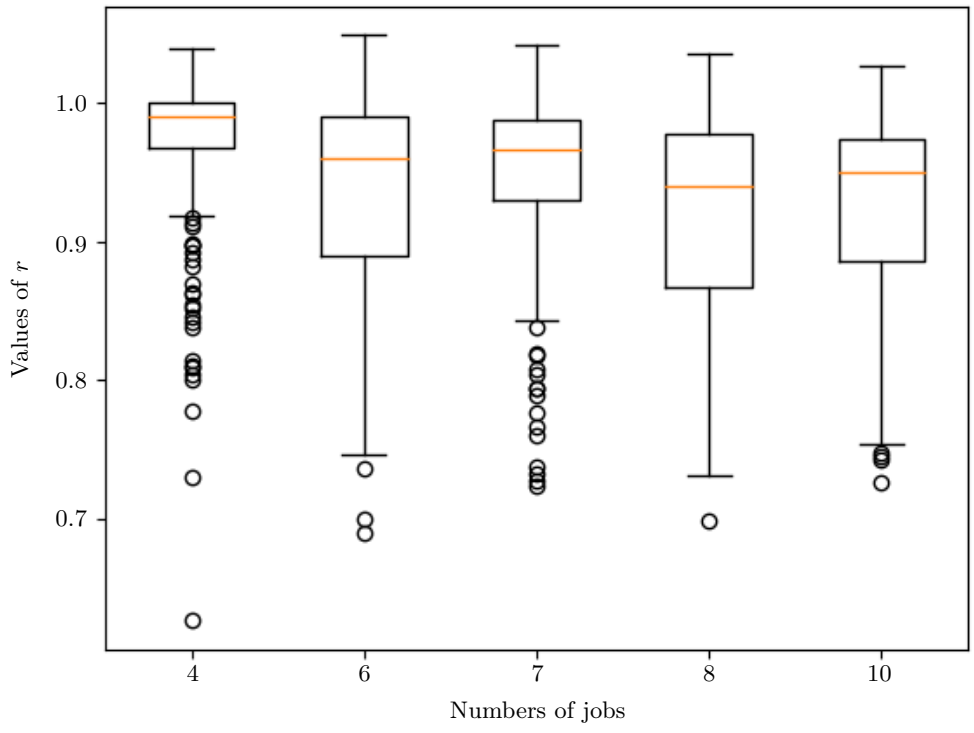


Рис. 3. Диаграмма размаха значений $r$ для разного количества работ

Fig. 3. A box plot of the range of $r$ values for a different number of jobs

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

Таблица 3. Среднее преимущество $\theta(\%)$ по целевой функции для решений, найденных пакетом CPLEX, по сравнению с решениями жадного алгоритма

Table 3. Average advantage $\theta(\%)$ of objective function of solutions, found by CPLEX package, comparing to the greedy algorithm solutions

| | **4 jobs** | **6 jobs** | **7 jobs** | **8 jobs** | **10 jobs** |
|---|---|---|---|---|---|
| Trivial order | 1.2 | 1.8 | 0.6 | 0.8 | 1 |
| One to many to one | 1.7 | 6.5 | 5.2 | 4.9 | 3.5 |
| Random order | 4.2 | 4.1 | 3.3 | 4.2 | 4.1 |
| Bitree order | 5.2 | 4.6 | 4.6 | 3.6 | 1.1 |

большем количестве случаев библиотека Intel TBB находит оптимальные или близкие к ним расписания. И наоборот, наибольшее преимущество показывают серии примеров с тривиальным частичным порядком. На некоторых примерах библиотека Intel TBB оказывается быстрее, однако количество таких примеров не превосходит 4% от общего количества и может быть связано с погрешностью вычисления входных данных задачи и тестирования полученных решений.

Заметим, что расчет скоростей выполнения работ в каждой конфигурации является трудоемким процессом. В [10] предложен жадный алгоритм, который составляет расписания для заранее известного множества работ. Для этого алгоритм рассчитывает скорость выполнения работ, основываясь на информации о потреблении ими шины данных. В табл. 3 приведены результаты сравнения значений целевой функции, полученных жадным алгоритмом, с оптимальными значениями целевой функции, полученными пакетом CPLEX при подстановке в ЧЦЛП скоростей выполнения работ, рассчитанных как в жадном алгоритме. Здесь $\theta$ — среднее значение величины $\dfrac{C^{\mathrm{opt}}_{\max} - C^{\mathrm{gr}}_{\max}}{C^{\mathrm{opt}}_{\max}} \cdot 100\%$.

Полученные результаты показали, что отклонение от оптимального значения у жадного алгоритма достигает 6.5%, в то время как у библиотеки Intel TBB — 13.1%. Стоит отметить, что для 10 работ значения, приведенные в таблице, не вполне характеризуют возможности модели ЧЦЛП, так как более чем в половине случаев оптимальные решения не были найдены. Среднее преимущество 6.5% достигается благодаря тому, что для всех примеров серии пакет CPLEX нашел оптимальные решения. В то время как при тривиальном частичном порядке даже, например, при 6 работах пакетом CPLEX были найдены оптимальные решения только в 41 из 48 примерах.

**7. Заключение.** Проведен анализ задачи составления расписаний для работ на многоядерном процессоре с учетом их замедления при совместном выполнении. Предложена формальная постановка задачи и построена математическая модель частично целочисленного линейного программирования. Модель была записана в системе моделирования GAMS, и оптимальное решение вычислялось с использованием пакета CPLEX. Было произведено сравнение времени выполнения работ согласно расписаниям, построенным с помощью модели ЧЦЛП, и библиотекой Intel TBB, широко используемой в современном программном обеспечении. Результаты эксперимента показали, что решения, найденные согласно предложенной модели, на некоторых сериях имели преимущество до 13% в среднем, что позволяет сделать вывод, что при учете взаимного влияния работ общее время завершения проекта может быть существенно сокращено. Также приведены результаты сравнения значений целевой функции, полученных жадным алгоритмом, с оптимальными значениями целевой функции, полученными пакетом CPLEX. Максимальное среднее преимущество по серии составило 6.5%.

Предложенный в данной работе подход может быть применен для оптимизации программ с заранее известной структурой, которые предполагают многократный запуск с одинаковым размером входных данных, но, возможно, различными значениями этих данных. Если длительность подпрограмм мало зависит от входной информации, то предложенным методом можно единожды найти оптимальное расписание и при каждом запуске выполнять подпрограммы согласно этому расписанию. Также предложенный подход может использоваться для оценки имеющихся эвристических алгоритмов с целью принятия решения о необходимости совершенствования этих алгоритмов.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

### Список литературы


1. *Merkel A., Stoess J., Bellosa F.* Resource-conscious scheduling for energy efficiency on multicore processors // Proc. 5th European Conference on Computer Systems (EuroSys'10). April 13–16, 2010, Paris, France. New York: ACM Press, 153–166. doi 10.1145/1755913.1755930.
2. *Zhuravlev S., Blagodurov S., Fedorova A.* Addressing shared resource contention in multicore processors via scheduling // Comput. Archit. News **38** (1), 129–142. doi 10.1145/1735970.1736036.
3. *Jiang Y., Shen X., Chen J., Tripathi R.* Analysis and approximation of optimal co-scheduling on chip multiprocessors // Proc. 17th International Conference on Parallel Architectures and Compilation Techniques (PACT 08). October 25–29, 2008, Toronto, Canada. New York: ACM Press, 220–229. doi 10.1145/1454115.1454146.
4. *Xiao Z., Chen L., Wang B., Du J., Li K.* Novel fairness-aware co-scheduling for shared cache contention game on chip multiprocessors // Information Sciences. 2020. **526**. 68–85. doi 10.1016/j.ins.2020.03.078.
5. *Tian K., Jiang Y., Shen X., Mao W.* Makespan minimization for job co-scheduling on chip multiprocessors // Technical Report WM-CS-2010-08. Williamsburg: College of William & Mary, 2010. https://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.221.3153. Cited February 8, 2023.
6. *Liu M., Chu F., He J., Yang D., Chu C.* Coke production scheduling problem: a parallel machine scheduling with batch preprocessings and location-dependent processing times // Computers and Operations Research. 2019. **104**. 37–48. doi 10.1016/j.cor.2018.12.002.
7. *Shioura A., Shakhlevich N.V., Strusevich V.A.* Preemptive models of scheduling with controllable processing times and of scheduling with imprecise computation: a review of solution approaches // European Journal of Operational Research. 2018. **266**, N 3. 795–818. doi 10.1016/j.ejor.2017.08.034.
8. *Jozefowska J., Weglarz J.* Scheduling with resource constraints — continuous resources // Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Boca Raton: CRC Press, 2004. 24-1–24-15.
9. *Blazewicz J., Brauner N., Finke G.* Scheduling with discrete resource constraints // Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Boca Raton: CRC Press, 2004. 23-1–23-18.
10. *Eremeev A.V., Malakhov A.A., Sakhno M.A., Sosnovskaya M.Y.* Multi-core processor scheduling with respect to data bus bandwidth // Communications in Computer and Information Science. Vol. 1340. Cham: Springer, 2020. 55–69. doi 10.1007/978-3-030-65739-0_5.
11. *Althaus E., Brinkmann A., Kling P., et al.* Scheduling shared continuous resources on many-cores // Journal of Scheduling. 2018. **21**, N 1. 77–92. doi 10.1007/s10951-017-0518-0.
12. *Ierapetritou M.G., Floudas C.A.* Effective continuous-time formulation for short-term scheduling: I. Multipurpose batch processes // Industrial and Engineering Chemistry Research. 1998. **37**, N 11. 4341–4359. doi 10.1021/ie970927g
13. *Garey M.R., Johnson D.S.* Computers and intractability. A guide to the theory of NP-completeness. San Francisco: W.H. Freeman Press, 1979.
14. Библиотека Intel TBB. https://github.com/oneapi-src/oneTBB. (Дата обращения: 9 февраля 2023).


*Поступила в редакцию*
*28 июня 2022 г.*

*Принята к публикации*
*26 января 2023 г.*

### Информация об авторах

*Антон Валентинович Еремеев* — д.ф.-м.н., главн. научн. сотр.; Институт математики имени С. Л. Соболева СО РАН, Омский филиал, ул. Певцова, 13, 644043, Омск, Российская Федерация.

*Мария Юрьевна Сахно* — аспирант; Институт математики имени С. Л. Соболева СО РАН, Омский филиал, ул. Певцова, 13, 644043, Омск, Российская Федерация.

### References


1. A. Merkel, J. Stoess, and F. Bellosa, "Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors," in *Proc. 5th European Conf. on Computer Systems, Paris, France, April 13–16, 2010* (ACM Press, New York, 2010), pp. 153–166. doi 10.1145/1755913.1755930.

2023, **24** (1), 115–126. doi 10.26089/NumMet.v24r108

2. S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," Comput. Archit. News **38** (1), 129–142. doi 10.1145/1735970.1736036.
3. Y. Jiang, X. Shen, J. Chen, and R. Tripathi, "Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors," in *Proc. 17th Int. Conf. on Parallel Architectures and Compilation Techniques, Toronto, Canada, October 25–29, 2008* (ACM Press, New York, 2008), pp. 220–229. doi 10.1145/1454115.1454146.
4. Z. Xiao, L. Chen, B. Wang, et al., "Novel Fairness-aware Co-Scheduling for Shared Cache Contention Game on Chip Multiprocessors," Inf. Sci. **526**, 68–85 (2020). doi 10.1016/j.ins.2020.03.078.
5. K. Tian, Y. Jiang, X. Shen, and W. Mao, *Makespan Minimization for Job Co-Scheduling on Chip Multiprocessors*, Technical Report WM-CS-2010-08 (College of William & Mary, Williamsburg, 2010). https://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.221.3153. Cited February 8, 2023.
6. M. Liu, F. Chu, J. He, et al., "Coke Production Scheduling Problem: A Parallel Machine Scheduling with Batch Preprocessings and Location-Dependent Processing Times," Comput. Oper. Res. **104**, 37–48 (2019). doi 10.1016/j.cor.2018.12.002.
7. A. Shioura, N. V. Shakhlevich, and V. A. Strusevich, "Preemptive Models of Scheduling with Controllable Processing Times and of Scheduling with Imprecise Computation: A Review of Solution Approaches," Eur. J. Oper. Res. **266** (3), 795–818 (2018). doi 10.1016/j.ejor.2017.08.034.
8. J. Jozefowska and J. Weglarz, "Scheduling with Resource Constraints — Continuous Resources," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (CRC Press, Boca Raton, 2004), pp. 24-1–24-15.
9. J. Blazewicz, N. Brauner, and G. Finke, "Scheduling with Discrete Resource Constraints," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (CRC Press, Boca Raton, 2004), pp. 23-1–23-18.
10. A. V. Eremeev, A. A. Malakhov, M. A. Sakhno, and M. Y. Sosnovskaya, "Multi-core Processor Scheduling with Respect to Data Bus Bandwidth," in *Communications in Computer and Information Science* (Springer, Cham, 2020), Vol. 1340, pp. 55–69. doi 10.1007/978-3-030-65739-0_5.
11. E. Althaus, A. Brinkmann, P. Kling, et al., "Scheduling Shared Continuous Resources on Many-cores," J. Sched. **21** (1), 77–92 (2018). doi 10.1007/s10951-017-0518-0.
12. M. G. Ierapetritou and C. A. Floudas, "Effective Continuous-Time Formulation for Short-Term Scheduling: I. Multipurpose Batch Processes," Ind. Eng. Chem. Res. **37** (11), 4341–4359 (1998). doi 10.1021/ie970927g
13. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman, San Francisco, 1979).
14. Intel TBB Library. https://github.com/oneapi-src/oneTBB. Cited February 9, 2023.

*Received*
*June 28, 2022*

*Accepted for publication*
*January 26, 2023*

### Information about the authors

*Anton V. Eremeev* — Dr. Sci., Chief Scientist; Sobolev Institute of Mathematics SB RAS, Omsk Department, Pevtsova ulitsa, 13, 644043, Omsk, Russia.

*Maria Yu. Sakhno* — PhD Student; Sobolev Institute of Mathematics SB RAS, Omsk Department, Pevtsova ulitsa, 13, 644043, Omsk, Russia.