Оптимизация, системный анализ и исследование операций

© 2023 г. В.В. БАЛАШОВ, канд. физ.-мат. наук (hbd@cs.msu.ru),
В.А. КОСТЕНКО, канд. техн. наук (kostmsu@gmail.com),
И.А. ФЕДОРЕНКО (iliasfedorenko@mail.ru)
(Московский государственный университет им. М.В. Ломоносова),
Ц. ГАО, канд. физ.-мат. наук (gaojiexing@huawei.com)
(Московский исследовательский центр компании Хуавэй),
Ч.М. СУН, PhD (sunchumin@huawei.com),
Ц. СУН, PhD (j.sun@huawei.com)
(Гонконгский исследовательский центр компании Хуавэй)

АЛГОРИТМ ИМИТАЦИИ ОТЖИГА ДЛЯ ПОСТРОЕНИЯ СПИСОЧНЫХ РАСПИСАНИЙ С ОГРАНИЧЕНИЕМ НА КОЛИЧЕСТВО МЕЖПРОЦЕССОРНЫХ ПЕРЕДАЧ ДАННЫХ

Предложен алгоритм имитации отжига для построения многопроцессорных списочных расписаний минимальной длительности с дополнительным ограничением на количество передач между процессорами. Данное ограничение характерно для вычислительных систем с жесткими ограничениями на ресурсы межпроцессорной сети передачи данных. В целом задача минимизации длительности расписания возникает при разработке систем обработки данных в реальном масштабе времени, таких как бортовые и телекоммуникационные системы. Также задача актуальна для периферийных вычислений (edge computing). Экспериментальное исследование свойств алгоритма показало его высокую точность, стабильность и масштабируемость.

Ключевые слова: комбинаторная оптимизация, списочные расписания, алгоритм имитации отжига.

DOI: 10.31857/S0005231023080093, **EDN:** HDDNWC

1. Введение

Алгоритмы построения расписаний можно разделить на два больших класса.

Класс 1. Алгоритмы начинают процесс построения расписания с пустого расписания, не содержащего работ, а затем на каждом шаге добавляют работы в расписание. К алгоритмам этого класса относятся жадные алгоритмы [1, 2], алгоритмы, сочетающие жадные стратегии, и ограниченный перебор [3], алгоритмы, основанные на методе ветвей и границ [4, 5].

Класс 2. Алгоритмы работают с полным расписанием (содержащим все работы). На каждом шаге такие алгоритмы изменяют расписание, пытаясь его улучшить. К этому классу относятся алгоритмы имитации отжига [6, 7], генетические и эволюционные алгоритмы [8–11], муравьиные алгоритмы [12–14], алгоритмы случайного поиска [15].

Рассматриваемая в работе задача построения списочных расписаний отличается от классической [2, 16] наличием дополнительного ограничения на корректность расписаний. Задается ограничение на максимально возможное число передач данных между процессорами. Наличие дополнительного ограничения может приводить к тому, что алгоритмы первого класса заходят в тупик: есть еще не размещенные в расписание работы, при этом ни одна из них не может быть размещена без нарушения дополнительного ограничения на корректность расписания. Для алгоритмов второго класса дополнительное ограничение может приводить к невозможности перехода между двумя корректными расписаниями при помощи переноса работ между процессорами по одной: такой перенос может увеличить число межпроцессорных передач, приводя к нарушению ограничения.

Ограничение на максимально возможное число передач данных между процессорами позволяет только за счет планирования вычислений уменьшить нагрузку на сеть обмена данными. При этом граф потока данных не изменяется. Изменение графа потока данных означает разработку нового алгоритма решения прикладной задачи. Данная задача актуальна для систем с жесткими ограничениями на доступные ресурсы, такие как вычислительная мощность, объем оперативной памяти, пропускная способность сети. К таким системам относятся, в частности, встроенные и бортовые управляющие системы, системы цифровой обработки сигналов. Также задача актуальна для периферийных вычислений (edge computing), для выполнения которых нет возможности использовать высокопроизводительные вычислители.

В работе математически сформулирована постановка задачи построения списочных расписаний с дополнительным ограничением на количество передач данных между процессорами, рассмотрены подходы к решению близких задач, предложен алгоритм имитации отжига для этой задачи и приведены результаты экспериментального исследования точности и вычислительной сложности алгоритма.

2. Задача построения списочных расписаний с дополнительным ограничением

Общая задача построения списочных расписаний заключается в распределении фиксированного множества работ на процессоры и задания порядка их выполнения таким образом, чтобы оптимизировать желаемую меру эффективности расписания и выполнить заданные ограничения, которые обеспечивают корректность полученного решения. Прежде чем конкретизировать постановку задачи построения расписаний, определим модели исходных данных и самого расписания.

Модель прикладной программы. При определении модели прикладной программы предполагается, что работы, подлежащие планированию, и параллелизм, допускаемый при выполнении программы, заданы (выявлены) предварительно. Модель программы задается графом потока данных. Это ориентированный ациклический граф G с N вершинами и M ребрами. Каждой вершине графа соответствует работа из набора работ $P = \{p_i\}_{i=1}^{N}$. Каждому ребру графа соответствует однонаправленная передача данных между работами. Директивные сроки для работ не определены.

Модель вычислительной системы. Вычислительная система HW представляет собой S одинаковых по функциональным возможностям и производительности процессоров: $SP = \{sp_i\}_{i=1}^{S}$, каждый из которых может одновременно выполнять одну работу. Времена выполнения работ p_j на любом из процессоров определяются вектором времен выполнения: $C = \{c_j\}$, $j = 1, \ldots, N$. Работы непрерываемы, каждая работа должна быть целиком выполнена на одном процессоре.

Связь между процессорами sp_i и sp_j задается согласно матрице связности процессоров $D = \{d_{ij}\}$, $i = 1, \ldots, S$, $j = 1, \ldots, S$, причем $d_{ij} = 0$ при $i = j$. Значение d_{ij} задает задержку на передачу данных от sp_i к sp_j, т.е. если работы p_k и p_l находятся на процессорах sp_i и sp_j соответственно и в графе потока данных присутствует ребро (p_k, p_l), то завершение p_k и старт p_l должны быть разделены по времени не менее чем на d_{ij}.

Размеры данных и конфликты доступа к среде передачи данных не рассматриваются. На процессоре возможно выполнение работы, даже если с этого процессора передаются и/или на этот процессор принимаются данные.

Модель расписания. Расписание HP выполнения прикладной программы определено для заданного набора работ и набора процессоров, если заданы: 1) привязка работ к процессорам; 2) порядок выполнения работ на каждом процессоре.

Будем рассматривать расписание в списочной форме. Списочная форма расписания HP представляет собой ориентированный граф G_{HP}, вершины которого соответствуют работам. В состав G_{HP} входят:

— для каждого процессора — простая цепь из привязанных к этому процессору работ, задающая порядок выполнения работ на этом процессоре;

— для каждой межпроцессорной передачи данных — секущее ребро между цепью процессора-отправителя и цепью процессора-получателя, соединяющее две работы, участвующие в передаче данных.

Отметим, что в G_{HP} отсутствуют ребра, соответствующие передачам данных между работами, выполняющимися на одном процессоре и не являющимися соседними в цепи этого процессора.

Расписание HP корректно, если выполнены следующие ограничения.

1) Каждая работа назначена на процессор.

2) Любую работу обслуживает лишь один процессор, без прерывания.

3) Процессор в каждый момент времени выполняет не более одной работы.

4) Частичный порядок, заданный графом потока данных G, сохранен в HP: $G \subset G^*_{HP}$, где G^*_{HP} – транзитивное замыкание отношения G_{HP}.

5) Расписание HP должно быть беступиковым (необходимо для построения временной диаграммы расписания). Достаточным условием беступиковости при неограниченном размере буферов обмена данными является отсутствие контуров в графе G_{HP}.

6) Ограничение на долю межпроцессорных передач данных: $CR \leqslant CR_U$; $CR = \frac{M_p}{M}$, где M_p – количество передач данных между работами, находящимися на разных процессорах; CR_U – исходно заданное число.

Ограничения 1–5 являются основными. Они обеспечивают корректность расписания и однозначность построения временной диаграммы по списочной форме расписания. Ограничение 6 является дополнительным. В дальнейшем будем обозначать $HP \in HP^*_{1-5}$ или $HP \in HP^*_{1-6}$, если расписание HP удовлетворяет ограничениям 1–5 или 1–6 соответственно. Нижний индекс в HP^*_{1-5} и HP^*_{1-6} указывает ограничения, налагаемые на расписание.

Временная диаграмма выполнения программы. Временная диаграмма определена для расписания и модели вычислительной системы, если каждая работа назначена на процессор (из расписания), определены времена начала и завершения каждой работы. Время начала работы определяется как минимальное возможное с учетом передачи данных от работ-предшественников и порядка выполнения работ на процессорах.

Задачу построения расписаний будем рассматривать в следующем варианте постановки.

Дано: G – модель прикладной программы, HW – модель вычислительной системы, CR_U – ограничение на число межпроцессорных передач.

Требуется: построить расписание HP выполнения программы.

Минимизируемый критерий: длительность расписания, т.е. время завершения последней работы в расписании. Времена старта и завершения работ определяются по временной диаграмме расписания.

Ограничение корректности: $HP \in HP^*_{1-6}$.

3. Алгоритмы решения близких задач

При выборе классов алгоритмов для решения поставленной задачи учитывается масштабируемость алгоритмов, так как в настоящей работе графы потока данных для оценки качества результатов алгоритма содержат несколько тысяч (до 10 000) работ.

Рассматриваемая задача построения списочных расписаний представляет собой один из вариантов задачи построения многопроцессорных расписаний и является NP-трудной [17]. В связи с этим вряд ли возможно разработать полиномиальный алгоритм, находящий точное решение поставленной зада-

чи. Существующие решения, основанные на таких методах, как метод ветвей и границ и динамическое программирование, имеют плохую масштабируемость. Например, алгоритм на основе метода ветвей и границ, предложенный в [18], был применен к наборам данных с числом процессоров до 16 и графами, содержащими до 100 вершин, и продемонстрировал экспоненциальный рост времени работы.

Попытки применения методов целочисленного линейного программирования к задаче построения многопроцессорного расписания [19] привели к алгоритму, время выполнения которого на примере, содержащем 8 процессоров и граф с 30 вершинами, составляло несколько часов. При увеличении размера входных данных время выполнения существенно увеличивалось. В [20] указано, что такие алгоритмы могут быть использованы для входных данных, содержащих до 50 вершин в графе потока данных.

Для задачи построения многопроцессорного расписания было предложено множество жадных алгоритмов, многие из которых основаны на последовательной схеме планирования [10, 21, 22]. В такой схеме работы линейно упорядочиваются в соответствии с некоторым критерием, с учетом частичного порядка, заданного графом потока данных. В полученном порядке работы выбираются для назначения на процессоры. Алгоритм, сочетающий жадные стратегии и ограниченный перебор, предложен в [3]. Качество работы жадной стратегии сильно зависит от класса исходных данных. Однако жадные алгоритмы можно использовать для генерации начального приближения для алгоритмов, перечисленных ниже.

Для решения задачи построения многопроцессорного расписания также предлагались генетические и эволюционные алгоритмы [23]. Основная проблема алгоритмов этого класса в их ограниченной масштабируемости. Например, в [24] наиболее быстрый алгоритм обрабатывал задачу с 1000 работ порядка 1,5 ч на процессоре с частотой 2 ГГц. Одна из причин такого большого времени выполнения заключается в том, что генетические и эволюционные алгоритмы работают с популяциями, содержащими десятки и даже сотни решений.

В отличие от генетических и эволюционных алгоритмов алгоритм имитации отжига работает с единственным решением. В [6] предложен алгоритм имитации отжига для задачи построения многопроцессорного списочного расписания без задержек на передачу данных. Этот алгоритм хорошо масштабируем и может быть адаптирован к рассматриваемой задаче посредством учета времени передачи данных при вычислении длительности расписания. Для решения рассматриваемой задачи в качестве основы был выбран именно этот алгоритм.

Муравьиные алгоритмы также используются для построения многопроцессорных расписаний. В [25] эксперименты были ограничены наборами данных с 11 процессорами и 120 вершинами. Авторы работы [25] делают вывод: чтобы гарантировать сходимость муравьиного алгоритма, необходимо точно

настроить его многочисленные параметры, что серьезно усложняет применение его к задаче с разными размерами графов.

По результатам обзора не было найдено работ, в которых описаны алгоритмы, поддерживающие ограничение по числу межпроцессорных передач. Поэтому выбранный алгоритм нуждается в доработке в части процедуры выбора начального приближения и операций, применяемых к расписанию.

4. Алгоритм имитации отжига

Введем следующие обозначения и понятия:

— $T = \dfrac{T_0}{\log(1 + (i + 1))}$ – схема изменения температуры, где T_0 – начальная температура, i – номер текущей итерации (схема Больцмана);

— $F(HP)$ – функция оценки качества расписания HP;

— HP_{best} – лучшее найденное корректное расписание, причем $HP_{best} \in \in HP^*_{1-6}$;

— критерий останова — выполнение заданного числа итераций без изменения лучшего корректного решения.

Алгоритм имитации отжига построен по следующей схеме:

1) Инициализация. Задать начальное корректное расписание $HP_0 \in \in HP^*_{1-6}$ и считать его текущим: $HP = HP_0$. Вычислить длительность полученного расписания HP. Сохранить начальное расписание как лучшее найденное на данный момент: $HP_{best} = HP$.

2) Применить операцию преобразования расписания к HP и получить новое расписание HP'. Вычислить длительность расписания HP'. Если длительность расписания HP' является меньшей, чем у ранее сохраненного HP_{best}, и $HP' \in HP^*_{1-6}$, то сохранить его как лучшее найденное: $HP_{best} = = HP'$.

3) Найти изменение функции оценки качества расписания $\Delta F = F(HP') - -F(HP)$:

(i) Если $\Delta F \leqslant 0$ (т.е. нашлось расписание лучшее, чем текущее), то новое расписание становится текущим: $HP = HP'$.

(ii) Если $\Delta F > 0$, то с вероятностью $e^{-\frac{\Delta F}{T}}$ принять новое расписание в качестве текущего.

4) Повторить заданное число раз шаги 2 и 3 без изменения текущей температуры.

5) Если критерий останова выполнен, то завершить работу алгоритма.

6) Изменить текущее значение температуры T в соответствии с выбранной схемой и перейти к шагу 2.

Инициализация. Обычно начальное расписание строится с помощью различных жадных схем. Однако с их помощью тяжело построить расписание, которое бы удовлетворяло ограничению на количество межпроцессорных пе-

редач, поскольку после построения частичного расписания со сбалансированным распределением работ по процессорам жадный алгоритм заходит в тупик и не может добавить в расписание новые работы без нарушения этого ограничения.

Поэтому первоначально был использован простой вариант построения начального расписания, гарантирующий выполнение этого ограничения, — назначение всех работ на один процессор. Таким образом, $CR = 0$, так как межпроцессорные передачи отсутствуют. Порядок работ на процессоре определяется топологической сортировкой графа G. То есть между работами установлен такой порядок, чтобы любое ребро графа G, соответствующее передаче данных, вело от работы с меньшим номером к работе с большим номером. Построенное этим способом начальное приближение по результатам экспериментов не приводило к решению, близкому к оптимуму, при числе процессоров $S \geqslant 10$. Вместо этого происходило сбалансированное распределение работ на меньшее число процессоров, после чего алгоритм имитации отжига не мог улучшить данное решение.

Для решения проблемы сбалансированного распределения работ по процессорам при построении начального расписания было использовано программное средство METIS [26], включающее в себя реализацию алгоритма разбиения графов на заданное число подграфов, минимизирующего число ребер между полученными подграфами. Если подать на вход алгоритму граф потока данных G, выбрать число разбиений, равное числу процессоров, и сопоставить каждый полученный подграф процессору, то получится распределение работ по процессорам с минимизацией числа межпроцессорных передач.

Различия в размерах (количестве вершин) подграфов управляются параметром ufactor [27], ограничивающим отношение максимального размера подграфа к его среднему размеру. Увеличивая этот параметр, всегда возможно достичь необходимого отношения числа ребер между подграфами к общему числу ребер, т.е. удовлетворить заданному ограничению на количество межпроцессорных передач за счет меньшей сбалансированности распределения работ по процессорам.

После построения начального распределения работ по процессорам при помощи METIS порядок выполнения работ на процессорах определяется по следующей схеме. Для каждой работы по алгоритму ALAP [28] определяется наиболее позднее время ее завершения. Это время может быть вычислено, поскольку для работ определено их распределение по процессорам, а значит, известны и задержки на передачу данных между работами. Работы упорядочиваются по возрастанию наиболее позднего времени завершения. В соответствии с полученным порядком происходит выбор работ для включения в расписание. Если для очередной работы обнаружено, что она может быть поставлена раньше какой-то из уже включенных в расписание работ на том же процессоре без нарушения условий HP^*_{1-6}, она будет помещена перед этой

работой, в противном случае работа ставится в конец расписания на процессоре.

Операции преобразования расписания. Для представления расписания используется ярусная форма наибольшей высоты [6] с указанием привязки работ к процессорам. Эта форма в ходе выполнения алгоритма переводится во временную диаграмму для расчета времени работы программы с заданным расписанием (асимптотическая сложность перевода такого представления во временную диаграмму — $O(M)$). Ярусная форма наибольшей высоты характеризуется тем, что на каждом ярусе находится ровно одна работа и для любой работы p_i все работы-предшественники расположены на более высоких ярусах, чем p_i.

Для изменения расписания использовалась функционально полная система операций преобразования расписаний $\{O_1, O_2\}$, описанная в [29]. При представлении расписания в ярусной форме максимальной высоты операции выглядят следующим образом:

1) $O_1(p_i, sp_m \to sp_k)$ — операция изменения привязки работы. Переносит работу p_i с процессора sp_m на процессор sp_k на тот же ярус;

2) $O_2(p_i, sp_m, c)$ — операция изменения порядка работ на процессоре. Перемещает работу p_i на ярус c, смещая остальные работы в сторону изначальной позиции работы p_i на процессоре.

Сложность операции O_1 равна $O(1)$, а операции O_2 — $O(N)$.

Функция оценки качества расписания. Состоит из двух слагаемых: первое отвечает за длительность полученного расписания, второе — за превышение ограничения CR:

$$F(HP) = K\frac{Time(HP)}{\sum\limits_{1}^{N} C_i + N \max\limits_{i,j=1\ldots S} D_{ij}} + (1-K)CR_{above},$$

где $Time(HP)$ – длительность полученного расписания, $CR_{above} = CR - CR_U$, если $CR > CR_U$, иначе $CR_{above} = 0$.

Знаменатель дроби в первом слагаемом отвечает за нормировку длительности расписания. Это выражение является верхней оценкой длительности расписания.

В [29] сформулирована и доказана следующая

Теорема 1. Если $HP_1 \in HP^*_{1-5}$ и $HP_2 \in HP^*_{1-5}$ – произвольные корректные варианты расписания, то существует конечная цепочка операций $\{O_i\}_{i=1}^{K}$, $O_i \in \{O_1, O_2\}$, переводящая расписание HP_1 в HP_2, такая что все K промежуточных расписаний являются корректными ($HP_t \in HP^*_{1-5}$) и $K \leqslant 4N$, где N – количество работ.

Введенные выше операции изменения расписания могут преобразовывать его так, что получившееся расписание уже не будет удовлетворять ограничению на количество межпроцессорных передач (HP^*_{1-6}). Возможность рас-

сматривать расписания с невыполненным ограничением необходима, так как в общем случае множество решений не является связным относительно операций, не нарушающих ограничение $CR \leqslant CR_U$. Добавление штрафа за его нарушение предотвращает слишком большое отклонение алгоритма от множества корректных решений. Из всего вышесказанного для данного алгоритма справедливо следующее

Утверждение 1. Используемая в алгоритме система операций преобразования расписаний позволяет перейти от произвольного начального приближения $HP_1 \in HP^*_{1-6}$ к оптимальному расписанию $HP_2 \in HP^*_{1-6}$ за $K \leqslant 4N$ шагов применения операций, причем для каждого из промежуточных расписаний HP_t верно, что $HP_t \in HP^*_{1-5}$.

5. Экспериментальное исследование свойств алгоритма

Для исследования использовались различные наборы входных данных с известным оптимумом, удовлетворяющие дополнительному ограничению с $CR_U = 0{,}4$. Число процессоров S варьировалось от 2 до 64; число работ N — от 10 до 1000 с шагом 100 и от 1000 до 10 000 с шагом 1000; плотность графа, выражающаяся в отношении числа ребер к числу вершин (M/N) в графе G, для всех наборов входных данных примерно равнялась 5; времена выполнения работ варьировались от 1 до 10; время передачи данных между процессорами — от 1 до 3, в зависимости от пары процессоров. В качестве критерия останова алгоритма было установлено 10 000 итераций без изменения лучшего корректного решения. Наборы с отношением $N/S < 10$ не рассматривались.

Набор входных данных, для которого известен оптимум (т.е. длительность оптимального расписания), формируется по следующей схеме. Предполагается, что каждый процессор непрерывно (без простоев) занят от момента 0 до заданного момента L, который и будет длительностью оптимального расписания. Для каждого процессора интервал его занятости $[0; L]$ разделяется на части случайной длительности, выбираемой между заданными минимальной и максимальной длительностью (в рассматриваемом случае между 1 и 10). Эти части являются работами, размещенными на данный процессор, и для каждой из них по построению определено время старта. Затем случайным образом добавляются передачи данных между работами (т.е. пары <работа-отправитель, работа-получатель>) так, чтобы время старта работы-получателя было не меньше, чем время завершения работы-отправителя плюс длительность передачи данных между конкретной парой процессоров; при добавлении передач данных контролируется выполнение условия $CR \leqslant CR_U$. Затем по расписанию восстанавливается граф потока данных; для этого имеются все необходимые данные: длительность работ, наличие и длительность передач данных между работами. Значение L выбирается так, чтобы с учетом минимальной и максимальной допустимой длительности работ было возможно деление совокупности интервалов занятости всех процессоров на суммарное количество частей, равное требуемому коли-

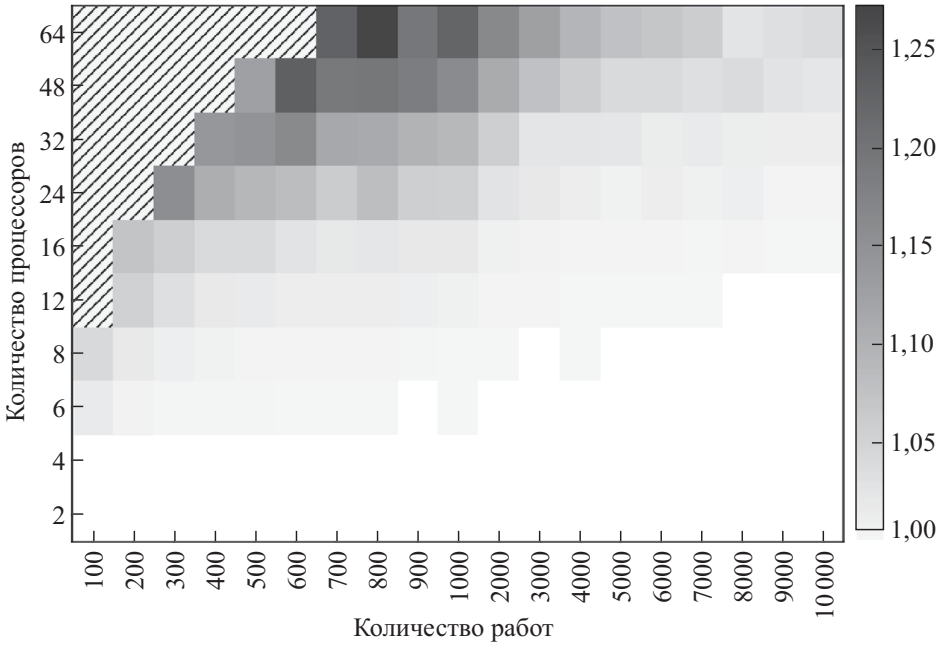

Рис. 1. Тепловая карта отношения длительности полученного расписания к длительности оптимального расписания.

честву работ. Передачи данных между работами добавляются до достижения требуемой плотности графа потока данных.

Утверждение 2. Расписание, по которому согласно описанной схеме «восстанавливается» граф потока данных, является оптимальным, потому что оно не содержит простоев и для каждого процессора интервал занятости один и тот же: $[0; L]$.

Поскольку для каждого использованного в экспериментах набора входных данных известна длительность оптимального расписания, для оценки качества алгоритма было выбрано отношение длительности полученного им расписания к длительности оптимального расписания. Это отношение всегда больше 1 и чем оно ниже, тем лучше результат работы алгоритма. На рис. 1 изображена тепловая карта, отображающая это отношение. По горизонтальной оси располагается число работ, по вертикальной оси — число процессоров. Отношение отображается градациями серого, соответствующая шкала изображена справа от графика. Алгоритм запускался на каждом наборе входных данных 5 раз. Результат усреднялся.

На рис. 2 показана та же информация в виде графиков. Каждой линии соответствуют наборы входных данных для фиксированного числа процессоров. По горизонтальной оси указано число работ, по вертикальной оси — отношение длительности полученного расписания к длительности оптимального расписания.

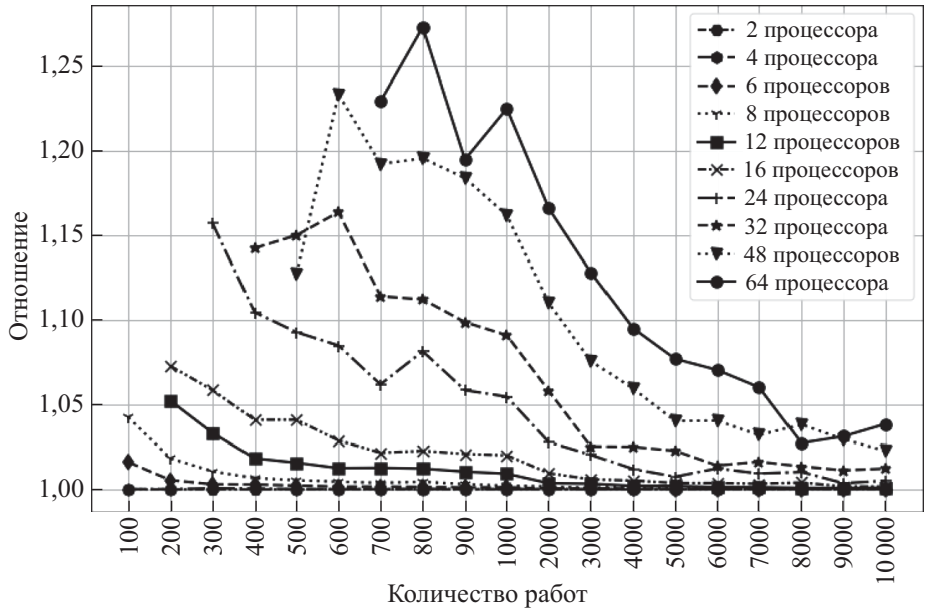

Рис. 2. Графики отношения длительности полученного расписания к длительности оптимального расписания.

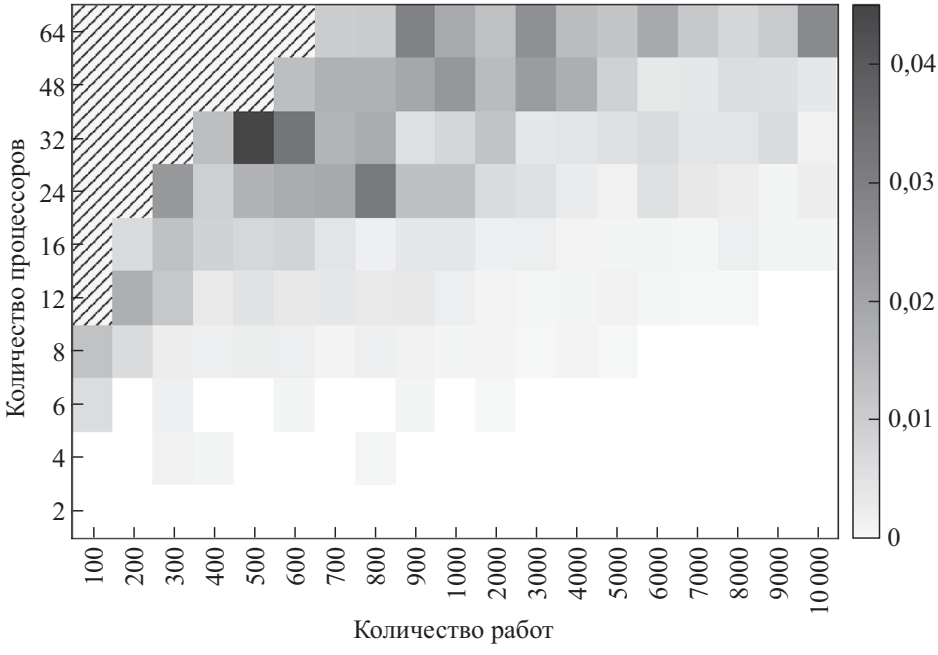

Рис. 3. Тепловая карта среднеквадратического отклонения отношения длительности полученного расписания к длительности оптимального расписания.

По результатам экспериментального исследования предлагаемого алгоритма отношение длительности построенного расписания к оптимуму почти всегда меньше 1,1, однако на отдельных наборах данных превышает это значение. Такое высокое отношение наблюдается на наборах данных с низким числом работ и высоким числом процессоров. Это можно объяснить тем, что при такой конфигурации среднее число работ на процессоре невелико и при переносе одиночной работы между процессорами длительность расписания сильно возрастает; вследствие этого измененное расписание с высокой вероятностью не принимается, что приводит к малому количеству переносов работ между процессорами, особенно при сниженной температуре. При увеличении числа работ на наборах данных с большим числом процессоров качество полученного решения улучшается.

Поскольку предлагаемый алгоритм является рандомизированным, необходимо исследовать его стабильность. На тепловой карте (рис. 3) показано среднеквадратическое отклонение отношения длительности полученного расписания к длительности оптимального расписания.

По тепловой карте на рис. 3 можно сделать вывод о высокой стабильности алгоритма. Для большинства наборов входных данных среднеквадратическое отклонение не поднимается выше 0,02. Более высокие (выше 0,03) значения отклонения на большом числе процессоров и малом числе задач возникают по тем же причинам, что и относительно невысокое качество решения в этой же ситуации.

Эксперименты выполнялись на компьютере с процессором Intel Core i5-8250U, 1,6 ГГц. Ни на одном наборе данных время выполнения алгоритма не превышало 6 мин. Следует отметить, что алгоритм демонстрирует приемлемое время выполнения и высокую точность в том числе на наборах входных данных, существенно превосходящих по масштабу те наборы, на которых проводилось исследование в статьях, рассмотренных в обзоре.

6. Заключение

Разработан алгоритм имитации отжига для построения многопроцессорных списочных расписаний минимальной длительности с дополнительным ограничением на количество передач между процессорами. Проблема выбора начального распределения работ по процессорам, удовлетворяющего дополнительному ограничению и являющегося сбалансированным (что существенно для длительности расписания), решена с использованием алгоритма из библиотеки METIS.

Проведенное экспериментальное исследование показало высокую точность и стабильность разработанного алгоритма, в особенности на наборах данных с достаточно большим (от 3000 до 10 000) количеством работ. Важным преимуществом алгоритма по сравнению с алгоритмами, рассмотренными в обзоре, является его высокая масштабируемость.

Разработанный алгоритм может применяться для построения расписаний в многопроцессорных вычислительных системах с ограниченными ресурсами межпроцессорной сети, например в бортовых и телекоммуникационных системах.

СПИСОК ЛИТЕРАТУРЫ

1. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
2. Теория расписаний и вычислительные машины / Под ред. Э.Г. Коффмана. М.: Наука, 1984.
3. *Костенко В.А.* Алгоритмы комбинаторной оптимизации, сочетающие жадные стратегии и ограниченный перебор // Известия РАН. Теория и системы управления. 2017. № 2. С. 48–56.

   Kostenko V.A. Combinatorial Optimization Algorithms Combining Greedy Strategies with a Limited Search Procedure // J. Comput. Syst. Sci. Int. 2017. V. 56. No. 2. P. 218–226. https://doi.org/10.1134/S1064230717020137
4. *Lawler E.L., Wood D.E.* Branch-and-Bound Methods: A Survey // Oper. Res. 1966. V. 14. No. 4. P. 699–719. https://doi.org/10.1287/opre.14.4.699
5. *Fujita S.* A Branch-and-Bound Algorithm for Solving the Multiprocessor Scheduling Problem with Improved Lower Bounding Techniques // IEEE Transact. Comput. 2011. https://doi.org/10.1016/j.procs.2016.07.216
6. *Калашников А.В., Костенко В.А.* Параллельный алгоритм имитации отжига для построения многопроцессорных расписаний // Известия РАН. Теория и системы управления. 2008. № 3. С. 133–142.

   Kalashnikov A.V., Kostenko V.A. A Parallel Algorithm of Simulated Annealing for Multiprocessor Scheduling // J. Comput. Syst. Sci. Int. 2008. V. 47. No 3. P. 455–463. https://doi.org/10.1134/S1064230708030155
7. *Зорин Д.А., Костенко В.А.* Алгоритм имитации отжига для решения задач построения многопроцессорных расписаний // АиТ. 2014. № 10. С. 97–110.

   Zorin D.A., Kostenko V.A. Simulated annealing algorithm in problems of multiprocessor scheduling // Autom. Remote Control. 2014. V. 75. No. 10. P. 1790–1801. https://doi.org/10.1134/S0005117914100063
8. *Holland J.N.* Adaptation in Natural and Artificial Systems / Ann Arbor, Michigan: Univ. of Michigan Press, 1975.
9. *Скобцов Ю.А.* Основы эволюционных вычислений. Донецк: ДонНТУ, 2008.
10. *Akbari M., Rashidi H.* An Efficient Algorithm for Compile-Time task scheduling problem on heterogeneous computing systems // Int. J. Academ. Res. 2015. V. 7. No. 1. P. 192–202. https://doi.org/10.7813/2075-4124.2015/7-1/A.45
11. *Rzadca K., Seredynski F.* Heterogeneous Multiprocessor Scheduling with Differential Evolution // IEEE Congress on Evolutionary Computation, 2005. V. 3. P. 2840–2847. https://doi.org/10.1109/CEC.2005.1555051
12. *Dorigo M.* Optimization, Learning and Natural Algorithms // Ph.D. Thesis. Dipartimentodi Elettronica. Milano: Politechnico Di Milano, 1992.
13. *Штовба С.Д.* Муравьиные алгоритмы: теория и применение // Программирование. 2005. № 4. С. 1–15.

Shtovba S.D. Ant Algorithms: Theory and Applications // Program. Comput. Software. 2005. V. 31. No. 4. P. 167–178. https://doi.org/10.1007/s11086-005-0029-1

14. *Myszkowski P.B., Skowronski M.E., Olech L.P. et al.* Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem // Soft Comput. 2015. V. 19. No. 12. P. 3599–3619. https://doi.org/10.1007/s00500-014-1455-x
15. *Растригин Л.А.* Статистические методы поиска. М.: Наука, 1968.
16. *Шахбазян К.В., Тушкина Т.А.* Обзор методов составления расписаний для многопроцессорных систем // Зап. научн. сем. ЛОМИ. 1975. Т. 54. С. 229–258.
17. *Garey M.R., Johnson D.S.* Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., 1979.
18. *Rahman M.* Branch and Bound Algorithm for Multiprocessor Scheduling // M.S. Thesis, Dept. Comput. Eng., Dalarna Univ., Sweden, 2009.
19. *Venugopalan S., Sinnen O.* Optimal Linear Programming Solutions for Multiprocessor Scheduling with Communication Delays // Proc. ICA3PP 2012: Algorithms and Architectures for Parallel Processing, 2012. P. 129–138. https://doi.org/10.1016/j.jpdc.2016.03.003
20. *Mallach S.* Improved Mixed-Integer Programming Models for the Multiprocessor Scheduling Problem with Communication Delays // J. Combinat. Optim. 2018. V. 36. P. 871–895. https://doi.org/10.1007/s10878-017-0199-9
21. *Hwang R., Gen M., Katayama H.* A Comparison of Multiprocessor Task Scheduling Algorithms with Communication Costs // Comput. Oper. Res. 2008. V. 35. P. 976–993. https://doi.org/10.1016/j.cor.2006.05.013
22. *Красовский Д.В.* Алгоритмы решения задачи составления оптимального расписания без прерываний // Дисс. . . . канд. физ.-мат. наук, 05.13.18 Москва, 2007, 109 с.
23. *da Silva E.C., Gabriel P.H.R.* Genetic Algorithms and Multiprocessor Task Scheduling: A Systematic Literature Review // Proc. ENIAC 2019. P. 250–261. https://doi.org/10.5753/eniac.2019.9288
24. *Sheikh H.F., Ahmad I., Fan D.* An Evolutionary Technique for Performance-Energy-Temperature Optimized Scheduling of Parallel Tasks on Multi-Core Processors // IEEE Trans. Parallel Distributed Syst., 2016. V. 27. No. 3. P. 668–681. https://doi.org/10.1109/TPDS.2015.2421352
25. *Lo S.-T., Chen R.-M., Huang Y.-M., Wu C.-L.* Multiprocessor System Scheduling with Precedence and Resource Constraints Using an Enhanced Ant Colony System // Expert Syst. Appl. 2008. V. 34. No. 3. P. 2071–2081. https://doi.org/10.1016/j.eswa.2007.02.022
26. METIS — Serial Graph Partitioning and Fill-reducing Matrix Ordering: [Электронный ресурс]. URL: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview (Дата обращения: 21.11.2022).
27. METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 5.1.0: [Электронный ресурс]. URL: http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf (Дата обращения: 21.11.2022).

28. *Wu M.-Y., Gajski D.D.* Hypertool: A programming aid for message-passing systems // IEEE Trans. Parallel Distributed Syst. 1990. V. 1. P. 330–343. https://doi.org/10.1109/71.80160

29. *Костенко В.А.* Задача построения расписания при совместном проектировании аппаратных и программных средств // Программирование. 2002. № 3. С. 64–80.

Kostenko V.A. The Problem of Schedule Construction in the Joint Design of Hardware and Software // Program. Comput. Software. 2002. V. 28. No 3. P. 162–173. https://doi.org/10.1023/A:1015636230903

Статья представлена к публикации членом редколлегии А.А. Лазаревым.

Поступила в редакцию 19.12.2022

После доработки 13.04.2023

Принята к публикации 09.06.2023