ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2008, № 3, с. 133–142

СИСТЕМНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

УДК 681.3

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ИМИТАЦИИ ОТЖИГА SIMULATED ANNEALING ДЛЯ ПОСТРОЕНИЯ МНОГОПРОЦЕССОРНЫХ РАСПИСАНИЙ

© 2008 г. А. В. Калашников, В. А. Костенко

Москва, МГУ

Поступила в редакцию 09.11.07 г.

Рассмотрены алгоритмы имитации отжига для решения задач построения многопроцессорных расписаний, предложен подход к их распараллеливанию и приведены результаты сравнительного исследования классического последовательного, последовательного и параллельного алгоритмов имитации отжига, использующих разбиение исходного пространства решений на области.

Введение. Алгоритмы имитации отжига в процессе поиска оптимального решения с некоторой вероятностью допускают переход в состояние с более высоким значением целевой функции. Это свойство позволяет им выходить из локальных оптимумов. Принципы, положенные в основу работы алгоритмов, можно объяснить на следующей физической аналогии [1]. На рис. 1 изображен шарик в коробке, внутренняя поверхность которой соответствует ландшафту целевой функции. При сильном встряхивании коробки в горизонтальном направлении шарик может переместиться из произвольной точки в любую другую. При постепенном уменьшении силы встряхивания будет достигнуто условие, когда эта сила обеспечивает перемещение шарика из точки A в точку B, но недостаточна для того, чтобы шарик мог переместиться из B в A. При дальнейшем уменьшении силы встряхивания до нуля шарик остановится в точке B – точке глобального минимума. В алгоритмах имитации отжига аналогом силы встряхивания является вероятность перехода в состояние с более высоким значением целевой функции. В начале работы алгоритма эта вероятность должна быть достаточно велика, чтобы была возможность перехода от выбранного начального приближения к любому другому решению. В процессе работы алгоритма вероятность перехода уменьшается в соответствии с выбранным законом.

Для многих *NP*-трудных задач наилучшие решения были получены алгоритмами имитации отжига. Однако их недостатком является высокая вычислительная сложность. Это обусловлено тем, что для получения хорошего решения требуется очень медленное понижение вероятности перехода в состояние с более высоким значением целевой функции, которое приводит к большому числу итераций алгоритма. При проектировании вычислительных систем реального времени на каждом этапе проектирования от аванпроекта до создания опытного образца необходимо решение задачи построения статических многопроцессорных расписаний. Для оценки времени выполнения расписания используются имитационные модели. По мере детализации аппаратных и программных средств вычислительной системы сложность и соответственно время выполнения имитационных моделей увеличиваются. Кроме того, на начальных этапах часто поддерживается несколько версий разрабатываемой системы. В [2] показано, что алгоритмы имитации отжига могут применяться для построения расписаний выполнения прикладных программ на всех стадиях проектирования вычислительных систем реального времени. Однако для их практического внедрения актуальной остается задача уменьшения вычислительной сложности алгоритмов.

Одним из подходов, направленных на достижение этой цели, является разбиение пространства решений на области и поиск решения в каждой из них отдельно. Данный подход также позволяет строить параллельные алгоритмы с низким трафиком обмена между параллельными процессами. Построение алгоритма имитации отжига, основанного на этом подходе, требует решения следующих проблем:

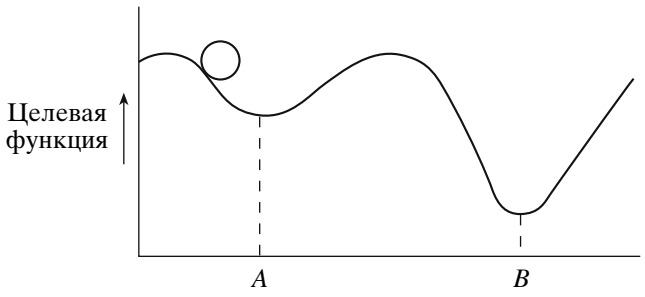

Рис. 1. Иллюстрация принципов работы алгоритмов имитации отжига

1) разбиение исходного пространства корректных решений на несколько непересекающихся областей, дающих в объединении все пространство;

2) выбор начального корректного решения в каждой из областей;

3) введение операций преобразования решения таким образом, чтобы они были замкнуты в каждой из областей;

4) выбор способа распределения областей по узлам вычислительной системы и схемы отсечения "неперспективных" областей в ходе работы алгоритма.

Для задач комбинаторной оптимизации решение этих проблем существенно зависит от конкретной задачи, так как структура пространства решений в каждом случае различна. В работе рассмотрены задачи построения многопроцессорных расписаний и приведены результаты сравнительного исследования классического последовательного, последовательного и параллельного алгоритмов имитации отжига, использующих разбиение исходного пространства решений на области.

1. Задача построения многопроцессорных расписаний. Общая задача построения статических многопроцессорных расписаний заключается в распределении и упорядочивании множества фиксированных работ (процессов) на заданное число ресурсов (процессоров) 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. При этом предполагаем, что выделение работ, подлежащих планированию, и параллелизм, допускаемый при выполнении программы, заданы (выявлены) предварительно.

Модель прикладной программы. В качестве модели прикладной программы будем использовать частный случай инварианта поведения программы, предложенного в [3], котрый является одной историей поведения программы для конкретного набора входных данных. Программа задается набором взаимодействующих процессов. Точками взаимодействия каждый процесс разбивается на упорядоченный набор рабочих интервалов. Модель программы может быть представлена размеченным ациклическим ориентированным графом $H(PR) = \{P, \prec\}$, где P – множество вершин, соответствующих рабочим интервалам, $\prec$ – множество дуг графа, отражающее взаимодействие рабочих интервалов. Отношение $\prec$ представляется следующим образом: если $p_i \prec p_k$, то рабочий интервал p_i необходимо выполнить до начала рабочего интервала p_k. На $\prec$ накладываются условия ацикличности и транзитивности. Каждая вершина имеет свой уникальный номер и метки: принадлежности рабочего интервала к процессу и вычислительной сложности рабочего интервала. Вычислительная сложность рабочего интервала позволяет оценить время его выполнения на процессоре. Дуга определяется номерами смежных вершин и имеет метку, соответствующую объему данных обмена, который для каждой связи из $\prec$ позволяет оценить затраты времени на выполнение внешнего взаимодействия.

Расписание выполнения программы. Расписание HP сформировано, если для каждого рабочего интервала введены привязка и порядок $\prec_{HP}$. Привязка – всюду определенная на множестве рабочих интервалов функция, которая задает их распределение по процессорам. Порядок устанавливает ограничения на последовательность выполнения рабочих интервалов и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве рабочих интервалов, распределенных на один и тот же процессор, представляет отношение полного порядка. Различные варианты расписаний могут быть получены изменением привязки и порядка в установленных ограничениями пределах, которые задаются требованием не нарушения отношения частичного порядка в модели поведения прикладной программы и определениями привязки и порядка.

Расписание HP корректно, если выполнены следующие ограничения:

1) каждый рабочий интервал должен быть назначен на процессор;

2) любой рабочий интервал обслуживает лишь один процессор;

3) частичный порядок, заданный в H, сохранен в HP: $\prec \subset \prec^T_{HP}$, $\prec^T_{HP}$ – *транзитивное замыкание отношения* $\prec_{HP}$;

4) расписание HP должно быть беступиковым. Условием беступиковости при неограниченном размере буферов обмена является отсутствие контуров в графе HP: $\prec_{HP}$ – *ациклично*;

5) все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор.

Ограничения 1)–4) обеспечивают сохранение инварианта поведения программы и являются обязательными. Ограничение 5) запрещает возобновление работы процесса после прерывания на другом процессоре, т.е. определяет способ организации параллельных вычислений в вычислительной системе (ВС). В дальнейшем будем говорить, что расписание допустимо $HP \in HP^*_{1\text{–}5}$, если оно удовлетворяет ограничениям 1)–5). Нижний индекс в $HP^*_{1\text{–}5}$ указывает ограничения, налагаемые на расписание.

Временна́я диаграмма выполнения программы. Она задана, если для каждого рабочего интервала установлены процессор, которому он соответсвует, и времена его начала и завершения. Временна́я диаграмма строится для заданных расписания и архи-

тектуры ВС или в ходе работы алгоритма. Согласно терминологии, используемой в теории расписаний, временна́я диаграмма также является одним из способов представления расписания. Однако в дальнейшем будем применять термин "временна́я диаграмма". Для ее построения приемлемы модели ВС различной степени детализации, получающие соответственно различную точность времен начала и завершения выполнения рабочих интервалов.

Задачу построения расписаний будем рассматривать в следующем варианте постановки.

Дано: $H(PR) = (P, \prec)$ – модель программы, $f(HP, HW)$ – функция вычисления времени выполнения расписания HP на архитектуре HW (целевая функция).

Требуется определить: HP– расписание выполнения программы.

Должны выполняться условия: $T = f(HP, HW) \longrightarrow min$ – минимум времени выполнения, $HP \in \in HP^*_{1-5}$ – ограничения 1–5.

Функция вычисления времени выполнения расписания может быть задана в аналитическом виде или в виде имитационной модели.

2. Особенности задачи и анализ подходов к ее решению. Основными особенностями рассмотренной задачи построения расписания выступают:

1) задача является NP-трудной;

2) для большинства архитектур ВС целевая функция, с помощью которой оценивается качество полученного решения – вычислимая функция, т.е. задается в виде имитационной модели;

3) задача построения расписаний относится к классу задач комбинаторной оптимизации.

Выделим три укрупненных класса алгоритмов, различающихся по используемым методам построения:

1) основанные на нахождении максимального потока в транспортной сети [4–6];

2) конструктивные (например, на базе метода ветвей и границ, динамического программирования [7], жадных стратегий [4]).

3) итерационные (например, генетические алгоритмы, алгоритмы имитации отжига).

Алгоритмы из классов 1), 2) на каждом шаге достраивают временну́ю диаграмму, размещая в нее одну из работ. Они применяются для конкретной математической модели вычислительной системы. В соответствии с данной моделью для алгоритмов, основанных на нахождении максимального потока в транспортной сети, формируется транспортная сеть, а для конструктивных алгоритмов выбирается стратегия ограниченного перебора. Применение данных алгоритмов ограничивается возможностями построения транспортной сети и стратегий перебора для сложных моделей, которые учитывают дополнительные ограничения на корректность временно́й диаграммы, обусловленные особенностями конкретной вычислительной системы. Использование приближенных моделей способно привести к тому, что полученная временна́я диаграмма не сможет быть реализована в реальной вычислительной системе. Кроме того, применение алгоритмов из класса 1) ограничивается их вычислительной сложностью (для большинства задач они имеют псевдополиномиальную сложность [8, 9]), а качество решений, обеспечиваемых алгоритмами класса 2), очень сильно зависит от характеристик исходных данных.

Итерационные алгоритмы позволяют работать с различными формами представления расписаний, а не только с временно́й диаграммой. Общую их схему можно представить следующим образом.

Шаг 1. Задание начального приближения расписания.

Шаг 2. Построение временно́й диаграммы или вычисление значения целевой функции.

Шаг 3. Анализ временно́й диаграммы или полученного значения целевой функции.

Шаг 4. Преобразование расписания.

Шаг 5. Если критерий останова не достигнут, то возврат к шагу 2.

В этих алгоритмах процедуры построения временно́й диаграммы и преобразования расписания разделены, что допускает использование сколь угодно точных моделей (вплоть до эмуляторов) и не накладывает таких жестких ограничений на классы допустимых архитектур вычислительных систем как для алгоритмов первых двух классов.

Алгоритмы, основанные на имитации отжига [1, 2, 10], позволяют получать решения хорошего качества, но требуют значительного времени работы для задач большого размера и при использовании точных моделей вычислительных систем. Одним из подходов к уменьшению времени работы алгоритмов имитации отжига является их распараллеливание.

3. Проблемы построения параллельного алгоритма имитации отжига для построения многопроцессорных расписаний. Алгоритм имитации отжига для решения задачи построения расписания можно представить следующей схемой.

Шаг 1. Задать начальное корректное расписание HP^0 и считать его текущим вариантом расписания ($HP = HP^0$).

Шаг 2. Установить начальную температуру T_0, приняв ее текущей ($T = T_0$).

Шаг 3. Применить операции преобразования расписания к текущему расписанию HP и получить новый корректный вариант расписания HP'.

Шаг 4. Найти изменение целевой функции $\Delta f = f(HP', HW) - f(HP, HW)$:

если $\Delta f \le 0$ (расписание улучшилось), то новый вариант расписания считать текущим ($HP = HP'$);

если $\Delta f > 0$ (расписание ухудшилось), то принять с вероятностью $p = e^{\frac{-\Delta f}{T}}$ в качестве текущего расписания новый вариант расписания HP'.

Ш а г 5. Повторить заданное число раз шаги 3 и 4 без изменения текущей температуры.

Ш а г 6. Понизить текущую температуру в соответствии с выбранным законом понижения.

Ш а г 7. Если не выполнен критерий останова, то перейти к шагу 3.

Преобразование расписания осуществляется с помощью функционально полной системы операций преобразования расписаний $O = \{O_1, O_2\}$ (в дальнейшем будем называть эту систему операций базовой системой операций) [11]:

$O_1(p_i, R_m, R_k, c)$ – операция изменения привязки рабочего интервала. Переносит рабочий интервал p_i с процессора R_m на процессор R_k (порядковый номер рабочего интервала становится равным c);

$O_2(p_i, R_m, c)$ – операция корректировки порядка выполнения рабочих интервалов на одном процессоре, модифицирующая порядковый номер рабочего интервала p_i на процессоре R_m (порядковый номер рабочего интервала становится равным c).

В [11] показано, что для базовой системы операций справедливо следующее свойство: если HP и HP' – произвольные корректные варианты расписания ($HP, HP \in HP^*_{1-4}$), то существует конечная цепочка операций $\{O_i\}_{i=1}^{K}$, $O_i \in \{O_1, O_2\}$, переводящая расписание HP в HP', такая, что все K промежуточных расписаний являются корректными и $K \le 2N$, где N – количество рабочих интервалов.

Для представления расписания используется ярусная форма максимальной высоты [12], т.е. каждый рабочий интервал, кроме привязки и порядка, характеризуется также и номером яруса, на котором он находится. Ярусная форма максимальной высоты обладает следующими свойствами: на каждом ярусе располагается только один рабочий интервал и для любого рабочего интервала p все его предшественники сосредоточены на более высоких ярусах, чем p. Операции преобразования расписания O_1 и O_2 применяются к ярусной форме максимальной высоты, причем операция O_1 сохраняет ярус рабочего интервала при изменении его привязки, при этом после операции O_1 получившееся расписание всегда корректно. Если на расписание накладывается условие 5), то операцию O_1 необходимо повторить для каждого рабочего интервала процесса. Сложность O_1 равна $O(1)$, операции O_2 – $O(N)$, где N – количество рабочих интервалов.

В настоящее время известно несколько подходов к распараллеливанию алгоритма имитации отжига [13, 14]:

параллельный независимый запуск алгоритма имитации отжига на нескольких узлах с различными начальными приближениями. Результат – лучшее решение из найденных на всех узлах. В [15] показано, что скорость поиска решений и их качество практически не отличаются от классического последовательного алгоритма;

параллельный запуск алгоритма имитации отжига с периодическим обменом информацией о полученных решениях, при этом узлы вычислительной системы производят рестарт имитации отжига из найденных решений с наименьшим значением целевой функции. Такой способ универсален и приемлем для любых задач оптимизации, но требует больших затрат на обмен данными [15];

разбиение пространства решений на области и поиск решения в каждой области отдельно [16, 17]. Такой способ может эффективно применяться на широком классе архитектур ВС, но требует разработки способа разбиения на области для каждой решаемой задачи. Не всякая задача комбинаторной оптимизации может быть разбита на такие подобласти, как показано в [18].

В данной работе для распараллеливания алгоритма имитации отжига будет использоваться подход, основанный на разбиении пространства решений на области. Для этого множество всех возможных решений HP^*_{1-5} задачи построения расписаний представляется совокупностью областей HP_1, HP_2, …, HP_k. Такое разбиение должно удовлетворять следующим условиям:

1) $HP_1 \cup HP_2 \cup \ldots \cup HP_k = HP^*_{1-5}$;

2) $HP_i \cap HP_j = \varnothing, \forall\, i, j : i \ne j$;

3) введенные в HP^*_{1-5} операции преобразования должны быть замкнутыми на областях HP_1, HP_2, …, HP_k и сохранять на них свойство, описанное выше.

Определенное таким образом разбиение пространства всех решений на области позволяет искать решение в каждой из них независимо от других. Используя априорные нижние оценки времени выполнения расписания в каждой области, можно отсекать те, которые заведомо не содержат оптимального решения. Поиск решения в любой области можно осуществлять на разных узлах вычислительной системы. При этом в связи с возможным отсечением областей, в том числе в процессе поиска решений, распределение областей по узлам вычислительной системы должно обеспечивать однородную загрузку всех узлов в ходе всего времени работы алгоритма.

Таким образом, построение параллельного алгоритма требует решения следующих подзадач:

Рис 2. Принцип разбиения пространства расписаний на области

1) разбиение исходного пространства корректных расписаний на несколько непересекающихся областей, дающих в объединении все пространство;

2) выбор начального корректного расписания в каждой из областей;

3) модификация операций преобразования расписаний таким образом, чтобы модифицированные операции были замкнуты в каждой из областей и сохраняли все свойства базовых операций;

4) выбор способа распределения областей по узлам вычислительной системы и схемы отсечения областей.

Необходимо заметить, что решение четырех подзадач существенно зависит от задачи комбинаторной оптимизации, так как структура пространства решений в каждом случае различна.

4. Параллельный алгоритм имитации отжига. Разбиение исходного пространства решений на области осуществляется с целью разделения задачи на подзадачи. После разбиения всего пространства решений на непересекающиеся области, дающие в объединении все пространство, можно проводить независимый поиск решений в областях, а затем выбрать из них наилучшее. Указанное разбиение достигается введением дополнительной разметки на граф модели поведения прикладной программы H для каждой области, которая расширяет ограничения на поведение программы. В качестве исходной области берется все пространство расписаний, разбиваемое на три непересекающиеся подобласти. Для этого фиксируются два произвольных рабочих интервала, не связанных транзитивным отношением порядка:

в первой области эти рабочие интервалы распределены на разные процессоры;

во второй рабочие интервалы обязаны выполняться на одном процессоре, причем второй рабочий интервал выполняется после первого;

в третьей области рабочие интервалы реализуются на одном процессоре, причем первый рабочий интервал следует после второго.

На рис. 2 схематично изображен принцип разбиения пространства расписаний на области. На первом этапе выбираются два рабочих интервала (2 и 4), не связанные отношением порядка, затем пространство решений разделяется на три области, как показано на рисунке. Каждой из областей HP_i, образующих пространство HP^*_{1-5}, можно поставить в соответствие граф $H_i = (P, \prec \cup \prec', J, K)$; P – множество вершин, соответствующих рабочим интервалам. Дугам $\prec \cup \prec'$ отвечают связи, определяющие взаимодействия между рабочими интервалами. Все связи $\prec$, присутствующие в H, сохраняются в H_i. Отношение $\prec'$ задает дуги, устанавливающие дополнительные ограничения на порядок выполнения рабочих интервалов в каждой области. Отношение $\prec \cup \prec'$ транзитивно и ациклично. Отношения J и K соответствуют ограничениям на привязку рабочих интервалов:

два рабочих интервала p_i и p_j связаны отношением J, $(p_i, p_j) \in J$, если они должны выполняться на одном процессоре;

два рабочих интервала p_i и p_j связаны отношением K, $(p_i, p_j) \in K$, когда они распределены на разные процессоры.

Отношения J и K симметричны и $J \cap K = \varnothing$, а J транзитивно. Все метки, присутствующие в графе H, сохраняются в графе H_i. Расписание HP является корректным в области HP_i, если выполнены условия:

1) каждый рабочий интервал должен быть назначен на процессор;

2) любой рабочий интервал соответствует лишь одному процессору;

3) частичный порядок $\prec \cup \prec'$, заданный в HP_i сохранен в HP;

4) расписание HP должно быть беступиковым. Условием беступиковости является отсутствие циклов в графе HP при неограниченном размере буферов обмена;

5) отношение J должно быть сохранено в HP: любые два рабочих интервала, связанные отношением J, должны быть назначены на один и тот же процессор;

6) отношение K сохраняется в HP: любые два рабочих интервала, связанные отношением K, должны быть назначены на разные процессоры.

Операции преобразования расписания внутри области. Алгоритмы выполнения операций O_1 и O_2 должны обеспечивать их замкнутость внутри области.

Введем следующие обозначения: $I_1 = \{p_i \in P | (p_i, p) \in J\}$ – множество рабочих интервалов, которые должны выполняться на одном процессоре с рабочим интервалом p; $I_2 = \bigcup\limits_{p_i \in I_1} \{p_j \in P | (p_j, p_i) \in K\}$ – множество рабочих интервалов, для которых запрещено выполнение на одном процессоре с рабочим интервалом p; SP – множество всех процессоров; SP^p_{HP} – процессор, на который распределен рабочий интервал p в расписании HP. Алгоритмы выполнения операции O_1 включают следующие шаги.

Ш а г 1. Случайным образом выбирается рабочий интервал p.

Ш а г 2. Строится множество процессоров SP', на которые возможно перенести рабочий интервал p: $SP' = SP \setminus \bigcup\limits_{p_k \in I_2} SP^{p_k}_{HP}$.

Ш а г 3. Случайным образом выбирается процессор из SP' и на него переносятся все рабочие интервалы из множества I_1.

Алгоритм выполнения операции O_2 объединяет нижеперечисленные действия.

Ш а г 1. Случайным образом выбирается рабочий интервал p.

Ш а г 2. Определяется допустимый диапазон ярусов $[L_{low} + 1, L_{high} - 1]$. Этот диапазон выбирается таким образом, что рабочий интервал p может быть перенесен на любой ярус из найденного диапазона.

Ш а г 3. Ярус из допустимого диапазона выбирается случайным образом и на него переносится рабочий интервал p.

Сложность алгоритмов применения операций O_1 и O_2 равна $O(N)$. Так как операции преобразования в области совпадают с операциями преобразования на всем пространстве решений, но применяются к графам с дополнительными дугами, то сохраняется свойство из разд. 3.

Основной характеристикой подпространства является минимальное теоретически возможное время выполнения расписания в этом подпространстве. Этот минимальный теоретический предел определяется критическим путем в графе $G_i = (P, \prec \cup \prec')$, соответствующем i-й области разбиения. Независимо от числа процессоров в вычислительной системе время выполнения расписания в области не может быть меньше критического пути в соответствующем графе. Для вычисления критического пути в случае несвязного графа используется следующий метод: вводятся две фиктивные вершины с нулевым временем выполнения и создаются дуги, направленные от первой фиктивной вершины ко всем вершинам исходного графа без предшественников, и дуги, связывающие вершины исходного графа без потомков со второй фиктивной вершиной. В результате получается связный граф с двумя фиктивными вершинами (входной и выходной), для которого критический путь считается обычным образом.

Алгоритм разбиения исходного пространства решений на области. Введем следующие обозначения: P^p_{pred} и $\prec^p_{pred}$ – множества вершин и дуг графа H, соответствующие рабочим интервалам, которые предшествуют рабочему интервалу p, в том числе и транзитивно, P^p_{anc} и $\prec^p_{anc}$ – множества вершин и дуг графа, отвечающие рабочим интервалам, которые следуют за рабочим интервалом p, в том числе и транзитивно, $P^p_{\prec}$ – множество рабочих интервалов, не связанных с рабочим интервалом p отношением порядка $\prec \cup \prec'$, в том числе и транзитивно. Для разбиения пространства на подобласти используется алгоритм, состоящий из следующих шагов.

Ш а г 1. Задать количество областей разбиения.

Ш а г 2. В список графов поместить граф, соответствующий всему пространству расписаний: $H = (P, \prec, \varnothing, \varnothing)$.

Ш а г 3. Выбрать первый граф $\tilde{H} = (P, \tilde{\prec}, \tilde{J}, \tilde{K})$ из списка и построить три графа для трех непересекающихся областей следующим образом.

Ш а г 3.1. $\hat{P} = P$, где $\hat{P}$ – временно́е множество рабочих интервалов. В начале каждой итерации оно совпадает с множеством P всех рабочих интервалов.

Ш а г 3.2. Выбрать из $\hat{P}$ рабочий интервал p_i, для которого критический пусть в графе $(P^{p_i}_{pred}, \prec^{p_j}_{pred}, \tilde{J}, \tilde{K})$ минимален и удалить p_i из множества $\hat{P}$. Если $\hat{P}$ пусто, закончить разбиение, сообщив о невозможности его продолжения.

Ш а г 3.3. Для выбранного рабочего интервала p_i построить множество $P^{p_i}_{\prec}$. Если оно пусто, перейти к шагу 3.2 и рассмотреть следующий рабочий интервал из $\hat{P}$. Если $P^{p_i}_{\prec}$ не пусто, то использовать из него

рабочий интервал p_j, соответствующий минимальному критическому пути в графе $(P^{p_j}_{anc}, \prec^{p_j}_{anc}, \tilde{J}, \tilde{K})$.

Ш а г 3.4. Построить графы H_1, H_2, H_3 для областей HP_1, HP_2, HP_3:

$H_1 = (P, \tilde{\prec} \cup (p_i, p_j), \tilde{J} \cup (p_i, p_j) \cup (p_j, p_i), \tilde{K})$,

$H_2 = (P, \tilde{\prec} \cup (p_j, p_i), \tilde{J} \cup (p_i, p_j) \cup (p_j, p_i), \tilde{K})$,

$H_3 = (P, \tilde{\prec}, \tilde{J}, \tilde{K} \cup (p_i, p_j) \cup (p_j, p_i))$.

Ш а г 4. Удалить граф $\tilde{H}$ из списка и добавить три построенные графа H_1, H_2, H_3 в его конец. Если количество графов в списке меньше заданного количества областей разбиения, перейти к шагу 3.

Данный алгоритм позволяет так осуществлять разбиение на области, что критические пути в графах областей разбиения будут существенно отличаться между собой за счет выбора на шагах 3.2 и 3.3 рабочих интервалов, отстоящих как можно "дальше" друг от друга: первый рабочий интервал имеет малое количество предшественников, второй является предшественником малого количества рабочих интервалов. Это позволяет отбросить большое количество областей (без запуска в них алгоритма имитации отжига), в которых критический путь больше времени выполнения расписаний, полученных при запуске алгоритма имитации отжига в других областях.

Распределение областей по узлам вычислительной системы осуществляется так, чтобы обеспечить максимально равномерную загрузку процессоров в течение всего времени работы параллельного алгоритма имитации отжига. В процессе работы алгоритма области, графы которых имеют критические пути большие, чем время выполнения наилучшего расписания, полученного к данному моменту, исключаются из дальнейшего рассмотрения. Таким образом, необходимо так распределить области по узлам вычислительной системы, чтобы в ходе работы алгоритма не возникали ситуации, когда на одном из узлов количество рассматриваемых областей существенно больше, чем на другом. Пусть n – количество областей разбиения, а m – количество узлов вычислительной системы, осуществляющих поиск решений в областях ($n > m$). Области упорядочиваются по критическому пути в графах. Тогда в i-й узел ($1 \le i \le m$) будут распределены области с номерами $j : j \bmod m = i$ $(1 \le j \le n)$ (рис. 3).

Для запуска алгоритма имитации отжига необходимо задать: начальное приближение расписания, начальную температуру и режим ее понижения, а также выбрать критерий останова. В качестве начальных приближений в областях разбиения используются произвольные корректные расписания, принадлежащие этим областям. Для построения начального приближения применяется жадный алгоритм. Начальная температура T_0 выбирается пропорционально средней длине рабочих интервалов, а

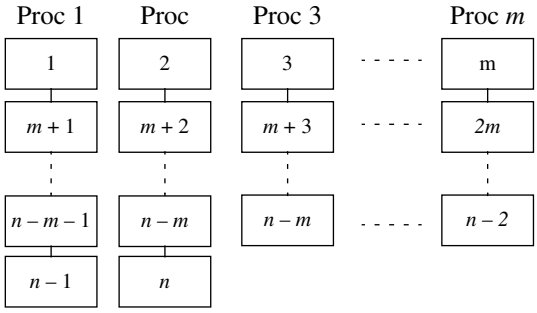

Рис. 3. Распределение областей по узлам вычислительной системы

режим понижения температуры задается законом Больцмана $T = \dfrac{T_0}{\ln(1+i)}$, i – номер текущей итерации алгоритма. На каждом шаге работы алгоритма последовательно применяются операции O_2, затем O_1.

Для параллельной работы алгоритма и отсечения областей используется следующая стратегия:

1) каждый узел осуществляет поиск решений в областях, начиная просматривать области с наименьшим критическим путем;

2) в каждой области выполняется фиксированное число итераций. После этого каждый узел исключает из дальнейшего рассмотрения области с критическим путем, большим, чем время выполнения наилучшего расписания, полученного этим узлом;

3) если было отсечено заданное количество областей, то узел инициирует обмен, передавая время наилучшего расписания остальным узлам, которые в свою очередь выполняют отсечение областей с учетом переданного им времени;

4) узел производит отсечение области, если в течение заданного числа итераций в данной области не произошло уменьшения целевой функции (времени выполнения расписания). При этом отсечение областей остальными узлами не инициируется;

5) узел заканчивает поиск, если в течение заданного числа итераций не произошло уменьшения целевой функции (время выполнения расписания) или, если все области оказались отсечены, или превышено допустимое число итераций. В качестве итогового расписания выбирается наилучшее среди расписаний, полученных в областях разбиения в результате работы всех узлов.

5. Результаты исследований. Рассматривался классический последовательный алгоритм имитации отжига, последовательный алгоритм, использующий разбиение на области, и параллельный алгоритм.

Для исследования привлекались графы модели прикладной программы со следующими характери-

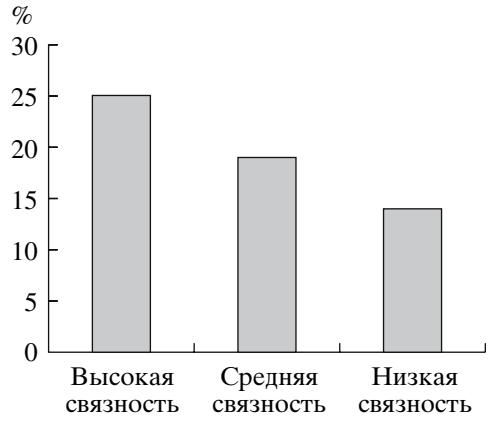

Рис. 4. Отсечение областей на одном узле без обмена с другими узлами

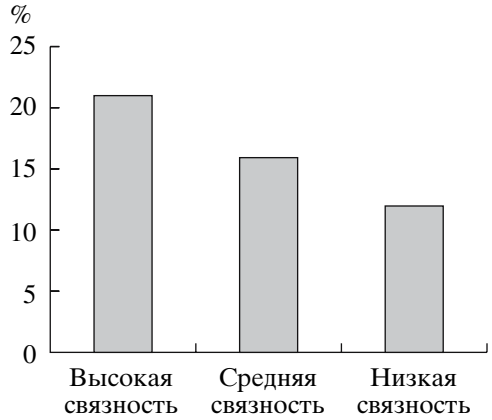

Рис. 5. Отсечение областей в результате обмена с другими узлами

стиками: количество рабочих интервалов от 100 до 250, число процессов от 40 до 90.

По степени связности были выделены три класса графов:

малое количество связей – $K/N \le 1.2$;

среднее количество связей – $1.2 < K/N < 1.6$;

большое количество связей – $K/N \ge 1.6$.

Здесь K – число дуг в графе, N – число вершин.

Было проведено сравнительное исследование времени, затрачиваемого на поиск решения одного и того же качества классическим последовательным алгоритмом, последовательным алгоритмом, реализующим разбиение на области, и параллельным алгоритмом. Были получены следующие результаты:

на одном узле, без обмена данными с другими узлами возможно отсечение до 25% областей (рис. 4), еще до 20% от оставшихся областей исключается после обмена данных между узлами (рис. 5). На рис. 6 показано количество отсеченных областей, в зависимости от количества итераций алгоритмов в этих областях;

последовательный алгоритм, использующий разбиение на области по сравнению с классическим делает в 2–3 раза меньше итераций для графов с высокой связностью. Для графов со средней и низкой связностью этот показатель существенно ниже;

в среднем параллельный алгоритм, запущенный на четырех процессорах, получал решение в 2.21 раза быстрее, чем последовательный, применяющий разбиение на области; на восьми процессорах параллельный алгоритм осуществлял поиск решения в 3.34 раза быстрее, чем последовательный. Предел увеличения быстродействия параллельного алгоритма связан с тем, что около 20% операций (разбиение на области) выполняются на одном узле последовательно. Однако следует заметить, что операция разбиения на области также может быть распараллелена, что позволит достичь лучшего отношения времен работы последовательного и параллельного алгоритмов. В среднем увеличение скорости выполнения параллельного алгоритма по сравнению с последовательным не зависит от параметров входного графа;

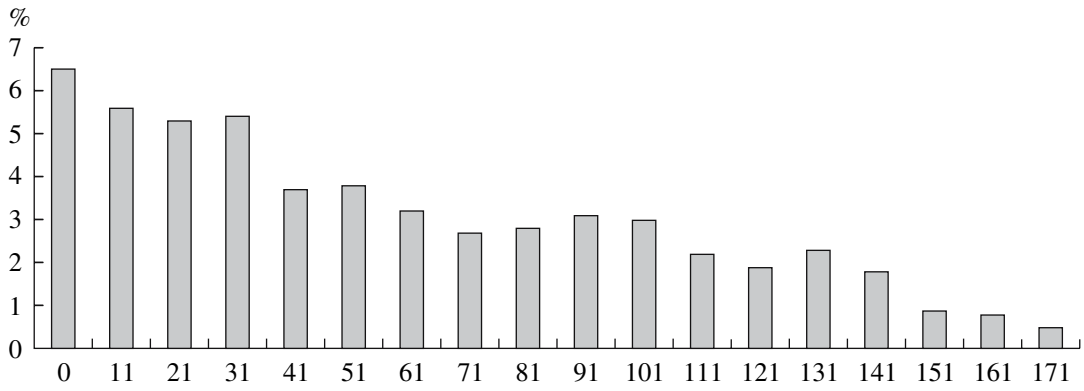

Рис 6. Зависимость количества отсеченных областей от количества итераций алгоритмов, запущенных в этих областях; $I = 1000$

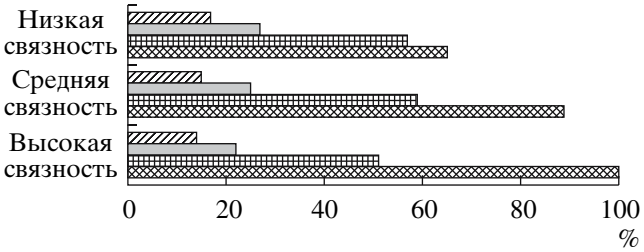

Рис. 7. Сравнение времени работы классического алгоритма с параллельным алгоритмом: ▨ – классический; □ – с разбиением на области; ⊞ – параллельный 4; ⊠ – параллельный 8

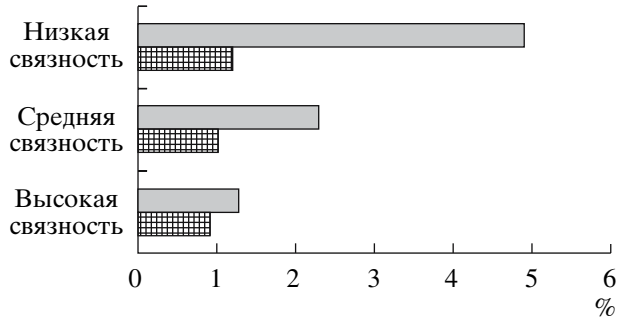

Рис. 8. Улучшение качества решений при использовании разбиения на области и распараллеливания по сравнению с решениями, полученными классическим алгоритмом: ▦ – с разбиением на области; ▥ – параллельный 4

время работы алгоритмов, использующих разбиение на области, увеличивается с уменьшением связности графа (рис. 7). Это обусловлено тем, что с уменьшением связности графа, уменьшается процент областей, отсекаемых алгоритмом в ходе работы. На графах с высокой связностью критический путь близок к времени выполнения расписания, получаемого алгоритмом и большинство областей с высокими критическими путями исключается из рассмотрения.

На рис. 7 приведены обобщенные результаты исследования. За 100% взято наибольшее зафиксированное время работы алгоритмов.

Также было проведено сравнительное исследование качества решений при одинаковом времени работы алгоритмов. Получены следующие результаты. Параллельный алгоритм, работающий на четырех процессорах, в среднем получает расписания с временем выполнения на 1.8% меньшим, чем последовательный, и на 2.7% меньшим, чем классический. Наибольшее уменьшение времени выполнения построенного расписания наблюдается на графах с низкой связностью и составляет в среднем 3.8% по сравнению с последовательным и 5.1% по сравнению с классическим. Это обусловлено тем, что на графах с низкой степенью связности время выполнения оптимального расписания существенно больше критического пути в графе и за малое время классический и последовательный алгоритмы не успевают довести расписание до "хорошего" локального минимума. На рис. 8 приведены обобщенные результаты исследования.

Заключение. Предложенные в работе последовательный и параллельный алгоритмы имитации отжига, использующие разбиение пространства решений на области, показали высокую эффективность при решении задач построения многопроцессорных расписаний. Последовательный алгоритм позволяет уменьшить время решения задачи до 3 раз по сравнению с классическим алгоритмом имитации отжига при сохранении, и во многих случаях – даже при улучшении качества получаемых решений. Параллельный алгоритм может быть эффективно реализован на локальной вычислительной сети, поскольку не требует высокого трафика обмена между узлами сети.

СПИСОК ЛИТЕРАТУРЫ

1. *Уоссермен Ф.* Нейрокомпьютерная техника: Теория и практика // М.: Мир. 1992. 240с.
2. *Костенко В.А, Калашников А.В.* Исследование различных модификаций алгоритмов имитации отжига для решения задачи построения многопроцессорных расписаний // Тр. Седьмой Междунар. научн. конф. "Дискретные модели в теории управляющих систем". М.: МАКС Пресс, 2006.
3. *Смелянский Р.Л.* Модель функционирования распределенных вычислительных систем // Вестн. МГУ. Сер 15, Вычисл. математика и кибернетика. 1990. № 3.
4. *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. М.: Наука., 1984.
5. *Liu C.L., Layland J.W.* Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment // J. ACM. 1973. V. 20. № 1. P. 46–61.
6. *Marte B.* Preemptive scheduling with release times, deadlines and due times // J. ACM. 1982. V. 29, № 3. P. 812–829.
7. *Беллман Р.* Динамическое программирование. М.: Изд-во иностр. лит., 1960.
8. Burns A. Scheduling Hard Real-Time Systems: A Review/ / Software Engineering J. 1991. V. 6. № 3. P. 116–128.
9. *Stankovic J.A.* Implications of Classical Scheduling Results for Real-Time Systems // IEEE Computer Society Press. 1995.
10. *Kirkpatrick S., Gelatt C. D., Vecchi M. P.*Optimization by Simulated Annealing // Science. 1983. № 4598.
11. *Костенко В.А.* Задача построения расписания при совместном проектировании аппаратных и программных средств // Программирование. 2002. № 3.
12. *Калашников А.В.* Алгоритм вычисления расстояния между расписаниями // Тр. Второй Всероссийск. научн. конф. "Методы и средства обработки

информации" (МСО'2005). М.: МАКС Пресс, 2005. С. 546–552.

13. *Greening D.R.* Parallel simulated annealing techniques // Emergent computation. 1991. P. 293–306.
14. *Gomez G.C.* A general interface for distributed and sequential simulated annealing // Qualifier II Research. Purdue University, 1994.
15. *Czech Z.J.* Parallel Simulated Annealing for the Delivery Problem // Parallel and Distributed Processing. 2001. Volume of Ninth Euromicro Workshop. P. 219–226.
16. *Janaki Ram D., Sreenivas T.H., Ganapathy Subramaniam K.* Parallel Simulated Annealing Algorithms // J. parallel and distributed computing. 1996. № 37. P. 207–212.
17. *Allwright J., Carpenter D.* A distributed implementation of simulated anneling for travelling salesman problem // Parallel Computing. 1989. № 3. P. 335–338.
18. *Barbosa V., Gafni E.* A distributed implementation of simulated annealing // J. Parallel and Distributed Computing. 1989. P. 411–433.