ПРОГРАММИРОВАНИЕ, 2013, № 5, с. 53–71

УДК 681.3

АЛГОРИТМЫ ПОСТРОЕНИЯ РАСПИСАНИЙ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ, ДОПУСКАЮЩИЕ ИСПОЛЬЗОВАНИЕ ИМИТАЦИОННЫХ МОДЕЛЕЙ

© 2013 г. В.А. Костенко

Факультет вычислительной математики и кибернетики МГУ им. М.В. Ломоносова
119991 ГСП-1, Ленинские горы, МГУ, д. 1, стр. 52
E-mail: kost@cs.msu.su

Поступила в редакцию 02.02.2013

Одним из важнейших требований к функционированию вычислительных систем реального времени (ВСРВ) является соблюдение директивных сроков выполнения прикладных программ. При нарушении директивных сроков, ВСРВ теряет свою работоспособность. В связи с этим возникает проблема обеспечения требуемой точности оценки времени выполнения прикладных программ. В работе предлагается подход к построению итерационных алгоритмов построения расписаний, в которых для оценки времени выполнения расписания прикладных программ могут использоваться имитационные модели с различной степенью детализации, что позволяет обеспечить требуемую точность оценки времени выполнения программ. Предложенный в работе подход может быть использован для построения итерационных алгоритмов следующих классов: генетические и эволюционные алгоритмы, алгоритмы имитации отжига, алгоритмы случайного поиска, локально-оптимальные алгоритмы.

1. ВВЕДЕНИЕ

Большинство современных бортовых вычислительных систем реального времени (ВСРВ) являются распределёнными. В работе [1] был предложен подход к разработке распределенных программ для ВСРВ в среде имитационного моделирования методом пошаговой детализации компонентов программы и модели аппаратных средств. Среда имитационного моделирования должна позволять описывать программы и аппаратные средства с различной степенью детализации и выполнять оценку наихудшего времени выполнения программ. Кроме этого на каждом уровне детализации должны решаться задачи проверки правильности свойств, как всей программы, так и ее отдельных компонент и построения расписания выполнения программы.

В данной работе предлагается подход к разработке итерационных алгоритмов построения расписаний, в которых для оценки времени выполнения расписания программ могут использоваться имитационные модели с различной степенью детализации. Предложенный в работе подход может быть использован для построения итерационных алгоритмов следующих классов: генетические и эволюционные алгоритмы, алгоритмы имитации отжига, алгоритмы случайного поиска, локально-оптимальные алгоритмы.

В разделе 2 определены модель прикладной программы, модель расписания, условия его корректности и временная диаграмма выполнения расписания. В третьем разделе рассмотрена общая схема итерационных алгоритмов построения расписаний; сформулированы задачи, которые необходимо решить для построения конкретного алгоритма, и рассмотрены наиболее широко применяемые подходы к построению итерационных алгоритмов: схемы случайного поиска (ненаправленный случайный поиск, направленный случайный поиск без самообучения,

направленный случайный поиск с самообучением), схема имитации отжига, генетические и эволюционные алгоритмы. В разделе 4 для алгоритмов использующих лишь унарные операции преобразования расписания введены способ представления расписаний и система операций преобразования расписаний, для которой доказаны ее полнота и замкнутость в пространстве корректных расписаний; доказано, что для двух любых корректных расписаний существует линейная последовательность операций, переводящая одно расписание в другое, такая, что все промежуточные расписания корректны и введена метрика в пространстве корректных расписаний. В разделе 5 приведены стратегии применения операций преобразования расписания (какую операцию применять, к какой работе, на какую позицию в расписании перемещать работу), которые могут быть использованы для широкого класса архитектур ВСРВ. В разделе 6 рассмотрен способ представления расписаний, позволяющий использовать известные операции мутации и скрещивания для целочисленного кодирования решений; доказано, что любое корректное расписание может быть задано предложенным способом. В разделе 7 приведен краткий аналитический обзор известных методов построения параллельных итерационных алгоритмов, в разделе 8 предложен и обоснован метод распараллеливания алгоритмов построения расписаний, которые используют лишь унарные операции преобразования расписаний.

2. МОДЕЛЬ ПРИКЛАДНОЙ ПРОГРАММЫ И РАСПИСАНИЯ

Модель прикладной программы. В качестве модели прикладной программы будем использовать частный случай инварианта поведения программы предложенного в работах [2, 3]. Следуя этим работам, обозначим поведение программы $Bh(PR)$ и определим $Bh(PR)$ следующим образом: $Bh(PR) =< S, \{R—(PR)\}, \{R \to (PR)\} >$, где S – множество всех возможных шагов процессов в допустимом диапазоне входных данных программы, $\{R \to (PR)\}$ – отношения, определяющие частичный порядок на множестве шагов каждого процесса, $\{R—(PR)\}$ – отношения взаимодействия между процессами.

Шаги процесса определяются последовательностью взаимодействий с другими процессами. Назовем рабочим интервалом процесса внутренние действия процесса между двумя последовательными взаимодействиями с другими процессами. Каждый рабочий интервал процесса по существу является реализацией соответствующего шага процесса.

Для задачи построения расписания будем использовать одну из историй поведения программы $H(PR) \in Bh(PR)$ (поведение программы для конкретного набора входных данных). Для $H(PR)$ отношение $\{R \to (PR)\}$ является отношением полного порядка, а множество S сужается до множества шагов, которые делают процессы для конкретного набора входных данных.

$H(PR)$ можно представить ациклическим ориентированным размеченным графом. Вершинам $P = \{p_i\}_{i=1}^{N_1} \cup \{p_i\}_{i=1}^{N_2} \cup \cdots \cup \{p_i\}_{i=1}^{N_K}$ этого графа соответствуют рабочие интервалы процессов, дугам $\prec = \{\prec_{ik} = (p_i,\ p_k)\}_{(i,k)\in(1\ldots N)}$ – связи, определяющие взаимодействия между рабочими интервалами из множества P (определяются объединением отношений $\{R—(PR)\}$, $\{R \to (PR)\}$). Где $\{p_i\}_{i=1}^{N_j}$ – множество рабочих интервалов j-го процесса, N_j – число рабочих интервалов в j-ом процессе, K – число процессов в программе PR, $N = N_1 + N_2 + \cdots + N_K$ – мощность множества P. Чередование рабочих интервалов различных процессов, назначенных на один и тот же процессор, допустимо, если не нарушается частичный порядок, заданный отношением $\prec$. Отношение $\prec_{ik}$ представляется следующим образом: если $p_i \prec_{ik}\ p_k$, то рабочий интервал p_i, необходимо выполнить до начала выполнения 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 $\{w_i\}_{i=1}^{N}$. Вычислительная сложность рабочего интервала позволяет оценить время выполнения рабочего интервала на процессоре. Дуга определяется номерами смежных вершин и имеет метку, соответствующую объему данных обмена $\{v_{ik}\}_{(i,k)\in(1\ldots N)}$. Объем данных обмена для каждой связи из позволяет оценить затраты времени на выполнение внешнего взаимодействия.

Расписание выполнения программы определе-

но, если заданы: 1) множества процессоров и рабочих интервалов, 2) привязка, 3) порядок. Привязка – всюду определенная на множестве рабочих интервалов функция, которая задает распределение рабочих интервалов по процессорам. Порядок задает ограничения на последовательность выполнения рабочих интервалов и является отношением частичного порядка на множестве рабочих интервалов, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве рабочих интервалов, распределенных на один и тот же процессор, является отношением полного порядка.

Модель расписания выполнения программы определим графом $HP = (\{SP_i\}_{i=(1...M)}, \prec_{HP})$. Где $\{SP_i\}_{i=(1...M)}$ – набор простых цепей (ветвей параллельной программы). Они образуются рабочими интервалами процессов, распределенными на один и тот же процессор (M – число процессоров в ВСРВ). Отношение частичного порядка $\prec_{HP}$ на множестве P для HP определим как объединение отношений: $\prec_{HP}=\prec_c \cup \prec_1 \cup \cdots \cup \prec_M$, $\prec_i$ – отношение полного порядка на SP_i, которое определяется порядковыми номерами рабочих интервалов в SP_i; $\prec_c$ – набор секущих ребер, которые определяются связями рабочих интервалов, распределенных на разные процессоры. Если рабочие интервалы p_i и p_j распределены на разные процессоры и в $\prec$ существует связь $\prec_{ij}$, то она определяет секущее ребро в графе HP. На отношение $\prec_{HP}$ накладываются условия ацикличности и транзитивности.

Условия корректности расписания. Расписание HP является корректным (сохраняет инвариант поведения программы), если выполнены следующие ограничения:

1. Каждый рабочий интервал должен быть назначен на процессор (в SP_i).

2. Каждый рабочий интервал должен быть назначен лишь на один процессор (в один SP_i).

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, т.е. в графе HP нет контуров: $\prec_{HP}$ – *ациклично.*

5. Все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор (в один и тот же SP_i).

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 ВСРВ, и не всегда является обязательным. В дальнейшем будем говорить, что расписание корректно $HP \in HP^*_{1-5}$, если оно удовлетворяет ограничениям 1–5, здесь HP^*_{1-5} – множество всех корректных расписаний для заданной модели программы $H(PR)$. Нижний индекс в HP^*_{1-5} указывает ограничения, налагаемые на расписание.

Временная диаграмма выполнения расписания HP на архитектуре ВСРВ HW задана, если для каждого рабочего интервала задан номер процессора, на котором он выполняется, и время начала выполнения и завершения выполнения рабочего интервала. Временная диаграмма может быть получена совместной интерпретацией моделей HW и $HP : TR = Mod(HP, HW)$. Получение приемлемых по точности оценок времени начала и завершения выполнения рабочих интервалов во многих случаях возможно лишь с использованием имитационных моделей ВСРВ, т.е. функция $TR = Mod(HP, HW)$ задается имитационной моделью.

3. ИТЕРАЦИОННЫЕ АЛГОРИТМЫ

Общую схему итерационных алгоритмов, в которых для построения временной диаграммы выполнения расписания используются имитационные модели можно представить следующим образом:

Шаг 1. Задать начальное корректное расписание $HP^0 \in HP^*_{1-5}$ *и считать его текущим вариантом расписания* ($HP = HP^0$).
Шаг 2. Построить временную диаграмму выполнения расписания $TR = Mod(HP, HW)$.
Шаг 3. Применить операции преобразования расписания к текущему расписанию HP *и получить новый корректный вариант расписания* $HP' \in HP^*_{1-5}$.
Шаг 4. Построить временную диаграмму

выполнения нового варианта расписания $TR' = Mod(HP',\ HW)$.
Шаг 5. Вычислить критерии качества расписаний HP *и* HP': $f(TR),\ f(TR')$.
Шаг 6. Используя значение критериев качества расписаний $f(TR),\ f(TR')$ *принять решение, какое расписание считать текущим.*
Шаг 7. Если критерий останова выполнен, то завершение работы алгоритма, если нет, то переход к шагу 3.

Итерационные алгоритмы на каждой итерации изменяют текущее полное расписание (включающее все работы из исходно заданного набора) пытаясь улучшить его. После шага 3 получается новое корректное расписание, для которого может быть построена временная диаграмма выполнения на заданной архитектуре ВСРВ. Для построения временной диаграммы выполнения расписания можно использовать как аналитическую функцию $Mod(HP,\ HW)$, так и имитационную модель.

Для построения конкретного алгоритма требуется решить следующие задачи:

1. Разработать способ представления расписания и операций преобразования текущего расписания на шаге 3.

2. Разработать стратегию применения операций преобразования текущего расписания на шаге 3 – какую операцию применять, к какому рабочему интервалу, на какую позицию в расписании перемещать рабочий интервал.

3. Разработать оператор выбора текущего решения на шаге 6.

4. Выбрать критерий останова алгоритма, используемый на шаге 7.

Способ представления расписания и операции преобразования текущего расписания должны удовлетворять условиям замкнутости (после применения любой операции к корректному расписанию получается корректное расписание) и полноты (для двух любых корректных расписаний HP и HP' существует конечная последовательность операций, приводящая HP к HP', такая, что все промежуточные расписания корректны). Условия замкнутости и полноты системы операций преобразования решений являются достаточными условиями того, что алгоритм за конечное число итераций может перейти от изначально заданного решения к оптимальному решению.

Оператор выбора текущего решения на шаге 6 определяется в соответствии с выбранным подходом к построению алгоритма. Наиболее широко применяемыми подходами к построению итерационных алгоритмов являются следующие: алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением) [4], алгоритмы имитации отжига [5], генетические и эволюционные алгоритмы [6].

Алгоритмы случайного поиска и имитации отжига на каждой итерации работают с одним расписанием HP, генетические и эволюционные алгоритмы – с некоторым множеством расписаний $\{HP\}$ (популяцией).

Основой алгоритмов случайного поиска служит итерационный процесс:

$$X^{k+1} = X^k + \alpha_k \cdot \frac{\xi}{\|\xi\|},\ k = 0, 1, \ldots,$$

где α_k – величина шага, $\xi = (\xi_1, \ldots,\ \xi_n)$ – некоторая реализация n-мерного случайного вектора ξ.

Ненаправленный случайный поиск (метод Монте-Карло) заключается в многократном случайном выборе допустимых вариантов решений и запоминании наилучшего из них:
$X^{k+1} = \xi,\ \xi \in S,\ g_i(\xi) \le 0,\ k = 0, 1, \ldots,$
$i = 1, \ldots, m.$

Все алгоритмы направленного случайного поиска без самообучения делают шаг от текущего значения X^k оптимизируемых параметров. Известны следующие алгоритмы направленного случайного поиска без самообучения [4]: алгоритм с парной пробой, алгоритм с возвратом при неудачном шаге, алгоритм с пересчетом при неудачном шаге, алгоритм с линейной экстраполяцией, алгоритм наилучшей пробы, алгоритм статистического градиента.

Алгоритмы случайного направленного поиска с самообучением заключаются в перестройке вероятностных характеристик поиска, т.е. в определенном целенаправленном воздействии на случайный вектор ξ. Он уже перестает быть равно-

вероятным и в результате самообучения приобретает определенное преимущество в направлениях наилучших шагов. Это достигается введением вектора $P^k = (p_1^k, p_2^k, \ldots, p_n^k)$, где p_j^k – вероятность выбора положительного направления по j-ой координате на k-ом шаге. Алгоритм рекуррентно корректирует значение компонентов этого вектора на каждой итерации в зависимости от того, насколько удачным/неудачным (изменилось значение целевой функции) был сделанный шаг. Описание и анализ различных способов коррекции вектора P^k приведены в [4].

Алгоритмы имитации отжига с некоторой вероятностью допускают переход в состояние с более высоким значением целевой функции:

$$P(X^k \to X^{k+1}) = \begin{cases} 1, & \Delta f \le 0 \\ exp(-\Delta f \Big/ T), & \Delta f > 0 \end{cases},$$

где T – некоторая температура, $\Delta f = f(X^{k+1}) - f(X^k)$. Поиск минимума начинается при высоком значении температуры T. В ходе работы алгоритма эта температура постепенно снижается (например, наиболее часто используется закон Больцмана или Коши). Поиск продолжается до тех пор, пока система не захватывается минимумом, из которого она уже не может выйти за счет тепловых флуктуаций.

Концепцию построения генетических алгоритмов, предложенную Холландом [6], схематично можно представить следующим образом:

1. Сгенерировать случайным образом популяцию размера P.
2. Вычислить целевую функцию для каждой строки популяции.
3. Выполнить операцию селекции.
4. Выполнить операцию скрещивания.
5. Выполнить операцию мутации.
6. Если критерий останова не достигнут, перейти к шагу 2, иначе завершить работу.

Популяция – это множество битовых строк для генетических алгоритмов и множество строк, состоящих из символов конечного алфавита, для эволюционных алгоритмов. Каждая строка представляет в закодированном виде одно из возможных решений задачи. По строке может быть вычислена целевая функция, которая характеризует качество решения. В качестве начальной популяции может быть использован произвольный набор строк. Основные операции алгоритма – селекция, скрещивание и мутация – выполняются над элементами популяции. Результатом их выполнения является очередная популяция. Данный процесс продолжается итерационно до тех пор, пока не будет достигнут критерий останова.

Генетические алгоритмы отличаются от эволюционных алгоритмов в основном способом кодирования (при кодировании решений допустим лишь бинарный алфавит) и универсальностью операций мутации и скрещивания.

Операция селекции обеспечивает формирование на очередной итерации алгоритма из строк, полученных на шагах 4 и 5, новой популяции. В данной работе приведем лишь два варианта операции, предложенных Холландом: схема пропорциональной селекции и схема рулетки. В схеме пропорциональной селекции, строка со значением целевой функции f_i даёт в новую популяцию $f_i \Big/ \bar{f}$ потомков, где $\bar{f}$ – среднее значение целевой функции для популяции. В схеме рулетки каждой строке выделяется сектор рулетки с центральным углом $2\pi f_i \Big/ \bar{f}$. Строка получает потомка, если случайно сгенерированное число в пределах от 0 до 2π попадает в сектор, соответствующий этой строке. Эти действия выполняются до тех пор, пока не будет полностью получен новый набор строк.

Простейший вариант операции скрещивания (одноточечное скрещивание) выполняется следующим образом: 1) вся популяция случайным образом разбивается на пары; 2) для каждой пары случайным образом генерируется число p'_c, если $p'_c < p_c$, то выбирается случайное целое число i в интервале $(1,\ l-1)$, и строки обменяются фрагментами, находящимися после i-го бита, в противном случае ничего не происходит. Где l – длина строки, p_c – вероятность скрещива-

ния. При многоточечном скрещивании выбирается несколько точек разрыва строки.

Операция мутации заключается в инвертировании каждого бита с заданной вероятностью. Параметр операции – вероятность мутации (p_m). Операция выполняется следующим образом: 1) для каждого бита генерируется случайное число p'_m; 2) если $p'_m < p_m$, то бит инвертируется.

Локально-оптимальные алгоритмы. Если на шаге 6 полученное новое расписание выбирается текущим, то только в том случае, если выполняется условие $f'(TR) < f(TR)$ (т.е если полученное расписание лучше текущего). Тогда алгоритм является локально оптимальным в пространстве HP^*_{1-5}. Метрика в пространстве HP^*_{1-5} будет введена в следующем разделе.

В алгоритмах имитации отжига, случайного поиска и локально-оптимальных алгоритмов требуются лишь унарные операции преобразования расписания, в генетических алгоритмах, требуются как унарные (операция мутации), так и бинарные (операция скрещивания). В схемах генетических алгоритмов и алгоритмов случайного поиска с самообучением уже введены стратегии применения операций преобразования текущего расписания, а для алгоритмов имитации отжига, направленного случайного поиска без самообучения и локально-оптимальных алгоритмов стратегии применения операций преобразования необходимо разработать. В следующих разделах введем способ представления расписаний, систему операций преобразования текущего расписания и стратегии их применения для алгоритмов имитации отжига, направленного случайного поиска без самообучения и локально-оптимальных алгоритмов. Для генетических алгоритмов введем способ представления расписаний, позволяющий использовать известные операции мутации и скрещивания для целочисленного кодирования решений, например, операции, предложенные в работах [7, 8].

4. СПОСОБЫ ПРЕДСТАВЛЕНИЯ И ОПЕРАЦИИ ПРЕОБРАЗОВАНИЯ РАСПИСАНИЙ ДЛЯ АЛГОРИТМОВ ИМИТАЦИИ ОТЖИГА, СЛУЧАЙНОГО ПОИСКА И ЛОКАЛЬНО-ОПТИМАЛЬНЫХ АЛГОРИТМОВ

Можно предложить следующие способы целочисленного представления расписания:

1. Расписание задается матрицей $Y(HP)_{N\times M}$, где элемент матрицы определяется:

$$y_{ij} = \begin{cases} c, & \text{если } p_i \in SP_j \\ 0, & \text{если } p_i \notin SP_j \end{cases},$$

c – порядковый номер рабочего интервала p_i в SP_j.

При данном способе представления расписания число целочисленных переменных равно $N \cdot M$.

2. Расписание задается: вектором привязки $Y(HP)_K$ и вектором порядка $X(HP)_N$, где i-й элемент вектора $Y(HP)_K$ равен номеру списка, в который назначены рабочие интервалы i-го процесса, а i-й элемент вектора $X(HP)_N$ равен порядковому номеру рабочего интервала в соответствующем списке. При данном способе представления расписания число целочисленных переменных равно $K + N$.

Операции преобразования расписания. Можно выделить следующие варианты отличия расписаний HP и HP' друг от друга:

- расписания HP и HP' отличаются лишь порядком рабочих интервалов как минимум в одном SP_j;
- расписания HP и HP' отличаются привязкой рабочих интервалов.

Введем соответствующие операции преобразования расписаний, позволяющие устранить указанные варианты отличия: $O = \{O1, O2\}$. Операции $O1, O2$ определим для первого способа представления расписания. Операции будем определять при предположениях: каждый процесс имеет не более одного рабочего интервала или при условии, что на расписание не накладывается ограничение 5.

Операция изменения порядка рабочих интервалов в одном списке изменяет порядковый номер рабочего интервала p_i в списке SP_m (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в данном списке (NS_m – число рабочих интервалов в списке SP_m) :

$$O1(p_i,\ SP_m,\ c) \equiv \begin{cases} \{c1 = y_{im}, \\ y_{im} = c \in (1, \ldots, c1-1); \\ \forall j |\ y_{jm} \neq 0 \vee c \leq y_{jm} < c1 : \\ y_{jm} = y_{jm} + 1\} - \\ \text{уменьшение порядкового} \\ \text{номера} \\ \{c1 = y_{im}, \\ y_{im} = c \in (c1+1, \ldots,\ NS_m); \\ \forall j |\ y_{jm} \neq 0 \vee c1 < y_{jm} \leq c : \\ y_{jm} = y_{jm} - 1\} - \\ \text{увеличение порядкового} \\ \text{номера} \end{cases}$$

Операция изменения привязки рабочих интервалов переносит рабочий интервал p_i из списка SP_m в список SP_k (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в списках SP_m и SP_k:
$\mathrm{O2}(\mathrm{p}_i,\ SP_m \to SP_k,\ c) \equiv \{y_{ik} = c \in (1 \ldots NS_k + 1); \forall j |\ y_{jk} \neq 0 \vee c \leq y_{jk}\ :\ y_{jk} + 1, NS_k = NS_k + 1;\ \forall j |\ y_{jm} \neq 0 \vee y_{im} < y_{jm}\ :\ y_{jm} = y_{jm} - 1,\ NS_m = NS_m - 1,\ y_{im} = 0\}$

Указанные интервалы параметра операций c могут привести к нарушению ограничений 3, 4. Покажем замкнутость и полноту системы операций $O = \{O1, O2\}$ и получим условия их применимости (выбор значения параметра c), не нарушающие ограничений 3, 4.

Теорема 1. Если HP и HP' – произвольные корректные расписания ($HP, HP' \in HP^*_{1-4}$), то существует конечная цепочка команд $\{O_i\}_{i=1}^K$, $O_i \in \{O1, O2\}$, переводящая расписание HP в HP', такая, что все K промежуточных расписаний являются корректными и $K \leq 2N$.

Доказательство этой теоремы приведено в работе [9].

Следствие 1. Существует стратегия перехода от произвольного допустимого расписания к оптимальному расписанию, такая, что длина цепочки команд $\{O_i\}_{i=1}^K$, $O_i \in \{O1, O2\}$, переводящей произвольное допустимое расписание в оптимальное, не превосходит значения $2N$ ($K \leq 2N$) и все K промежуточных расписаний являются корректными.

Условия применимости операций, не нарушающие ограничений на корректность расписания. Получим интервал значений параметра c при применении операции $O1/O2$ к рабочему интервалу p_i^s. Расписания HP будем представлять в ярусной форме максимальной высоты. Обозначим через $IN_i = \{p_k^l :\prec_{ki}\neq 0\}$ множество непосредственных предшественников рабочего интервала p_i^s (всегда выполняется $k < i$ и $l < s$). $OUT_i = \{p_k^l :\prec_{ik}\neq 0$ – множество непосредственных последователей рабочего интервала p_i^s (всегда выполняется $k > i$ и $l > s$). Операция $lin = \max\limits_{IN_i}(l)$ получает максимальный номер яруса, на котором расположен один из непосредственных предшественников рабочего интервала p_i^s, для рабочих интервалов, не имеющих предшественников $lin = 0$ (нулевой ярус всегда пуст). Операция $lout = \min\limits_{OUT_i}(l)$ получает минимальный номер яруса, на котором расположен один из непосредственных последователей рабочего интервала p_i^s, для рабочих интервалов, не имеющих последователей $lout = N$ (N – число ярусов в HP, для ярусной формы максимальной высоты число ярусов всегда равно числу рабочих интервалов). Тогда рабочий интервал p_i^s может быть размещен в любой из списков SP_j на любой из ярусов из интервала $lin < s < lout$. Если выбранный ярус занят, то осуществляется коррекция ярусной формы путем соответствующего сдвига ярусов.

Пусть рабочий интервал p_i^s переносится в SP_j или изменяется его порядковый номер в этом списке. Разобьем множество SP_j на три подмножества: $PIN_j = \{p_k^l\ :\ l \geq lin,\ p_k^l \in SP_j\}$ – множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных выше яруса $(lin - 1)$; $PI_j = \{p_k^l : lin < l < lout,\ p_k^l \in SP_j\}$ – множество рабочих интервалов из списка SP_j, находящихся на ярусах $]lin, lout[$; $POUT_j = \{p_k^l : l \leq lout,\ p_k^l \in SP_j\}$ – множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных ниже яруса $(lout + 1)$. Параметр c при размещении рабочего интервала p_i^s в SP_j может принимать следующие значения, не нарушающие ограничения на HP (индекс j для PIN, PI, $POUT$ будем опускать).

Указанные выше интервалы значений параметра c, не нарушающие ограничения на HP могут быть расширены, если ярусная форма мак-

Таблица 1

	PIN	PI	$POUT$	C
1	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$c = 1$
2	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$c = 1$
3	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$1 \le c \le \max\limits_{PI}(y_{jk}) + 1$
4	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$1 \le c \le \max\limits_{PI}(y_{jk}) + 1$
5	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$c = \min\limits_{PIN}(y_{jk}) + 1$
6	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$c = \min\limits_{PIN}(y_{jk}) + 1$
7	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$\min\limits_{PI}(y_{jk}) \le c \le \max\limits_{PI}(y_{jk}) + 1$
8	$PIN = \varnothing$	$PI = \varnothing$	$POUT = \varnothing$	$\min\limits_{PI}(y_{jk}) \le c \le \max\limits_{PI}(y_{jk}) + 1$

симальной высоты будет приведена к такому виду, что все рабочие интервалы SP_j, которые не находятся в отношении транзитивного порядка с непосредственным предшественником рабочего интервала p_i^s (находящимся на ярусе lin) и расположены на ярусах с меньшими номерами чем lin, будут перенесены на ярусы с номерами большими, чем lin. Аналогичное преобразование ярусной формы может быть осуществлено и для непосредственного последователя рабочего интервала p_i^s, находящимся на ярусе $lout$.

Пусть непосредственным предшественником рабочего интервала p_i^s является рабочий интервал p_m^{lin}, находящийся на ярусе lin, и рабочий интервал p_i^s переносится в SP_j, или изменяется его порядковый номер в этом списке. Пусть lin^* ярус, на котором находится транзитивный предшественник рабочего интервала p_m^{lin}, принадлежащий SP_j и с наибольшим порядковым номером в SP_j. Рабочие интервалы SP_j, находящихся между ярусами lin^* и lin, могут быть перераспределены по ярусам, таким образом, что они будут находиться на ярусах с большими номерами, чем lin (при этом осуществляется соответствующее перераспределение по ярусам рабочих интервалов и других списков). Аналогичное преобразование ярусной формы осуществляется и для последователей.

Возможность снятия условий: каждый процесс имеет не более одного рабочего интервала или на расписание не накладывается ограничение 5. Поскольку при доказательстве теоремы 1 и получении условий применимости операций $O1/O2$ использована ярусная форма максимальной высоты (на каждом ярусе находится лишь один рабочий интервал), то полученные результаты легко могут быть сведены к следующим случаям: когда каждый процесс может иметь более одного рабочего интервала и когда на расписание накладывается ограничение 5. При перемещении рабочего интервала из одного списка в другой, перемещаются также и все рабочие интервалы процесса, которому принадлежит перемещаемый рабочий интервал, но при этом остаются на прежних ярусах. В результате получаем ярусную форму нового расписания, и, следовательно, полученное расписание удовлетворяет ограничениям 3–5.

Метрика в пространстве расписаний. Метрика в пространстве HP^*_{1-5} корректных расписаний будет строиться как количество элементарных операций $O = \{O1,\ O2\}$ преобразования расписания.

Пусть $S = \{s_1, \ldots, s_K\}$ – произвольная цепочка операций $O1$ и $O2$. Цепочка S допустима для расписания $HP \in HP^*_{1-5}$, если операции из этой цепочки могут быть применены к HP в заданном порядке, причем так, что все получаемые промежуточные расписания являются корректными. Обозначим: $\Omega(HP)$ – множество всех допустимых цепочек операций преобразования для расписания HP; $HP^1 = S(HP)$ – расписание, полученное последовательным применением операций из цепочки S к расписанию HP. Длиной $L(S)$ цепочки S будем называть количество операций в этой цепочке.

Метрика в пространстве HP^*_{1-5} вводится следующим образом. Пусть $HP^1, HP^2 \in HP^*_{1-5}$ – два произвольных корректных расписания. Расстоянием $\rho(HP^1, HP^2)$ между расписа-

ниями HP^1 и HP^2 будем называть число, равное длине минимальной допустимой цепочки операций O_1 и O_2, которая переводит расписание HP^1 в расписание HP^2: $\rho(HP^1, HP^2) = \min\limits_{S\in\Omega(HP^1),\, S(HP^1)=HP^2} L(S)$.

Покажем корректность введенной метрики. Для этого надо доказать, что

1. функция ρ определена всюду на HP^*_{1-5};

2. для ρ выполнены аксиомы метрики.

Выполнение первого требования, очевидно, следует из теоремы 1 о полноте и замкнутости системы операций $O = \{O1,\ O2\}$. Покажем, что для функции ρ выполняются все свойства метрики:

Свойство 1.
$\forall HP^1,\ HP^2 \in HP^* : \rho(HP^1,\ HP^2) \geq 0$ и $\rho(HP^1,\ HP^2) = 0 \Leftrightarrow HP^1 = HP^2$.

Доказательство. Первая часть утверждения вытекает из того, что длина цепочки операций не может быть отрицательной. В совпадающих расписаниях каждый процесс имеет одинаковые значения привязки и порядка, поэтому применение любой операции делает эти расписания несовпадающими, откуда следует вторая часть утверждения.

Свойство 2.
$\forall HP^1,\ HP^2 \in HP^* : \rho(HP^1,\ HP^2) = \rho(HP^2,\ HP^1)$.

Доказательство. Пусть S – цепочка минимальной длины, переводящая расписание HP^1 в расписание HP^2. Тогда $\rho(HP^1,\ HP^2) = L(S)$. Из теоремы 1 следует, что существует обратная допустимая последовательность $S_1 \in \Omega(HP^2)$, $S_1(HP^2) = HP^1$ такая, что $L(S_1) = L(S)$ и поэтому $\rho(HP^2,\ HP^1) \leq L(S_1) = L(S) = \rho(HP^1,\ HP^2)$. Теперь докажем, что строгое неравенство $\rho(HP^2,\ HP^1) < \rho(HP^1,\ HP^2)$ не может выполняться. Если бы это неравенство выполнялось, то это означало бы существование цепочки операций $S^* \in \Omega(HP^2)$, $S^*(HP^2) = HP^1$, где $L(S^*) = \rho(HP^2,\ HP^1) < L(S)$. В этом случае должна существовать и обратная к S^* цепочка $S_1^* \in \Omega(HP^1)$, $S_1^*(HP^1) = HP^2$, причем $L(S_1^*) = L(S^*)$. Это, в свою очередь, означает, что $L(S_1^*) < L(S) = \rho(HP^1,\ HP^2)$, что противоречит предположению о минимальности цепочки S. Таким образом, неравенство $\rho(HP^1,\ HP^2) < \rho(HP^2,\ HP^1)$ не может иметь места, и всегда верно равенство $\rho(HP^1,\ HP^2) = \rho(HP^2,\ HP^1)$.

Свойство 3.
$\forall HP^1,\ HP^2,\ HP^3 \in HP^* : \rho(HP^1,\ HP^3) \leq \rho(HP^1,\ HP^2) + \rho(HP^2,\ HP^3)$ (неравенство треугольника).

Доказательство. Пусть S_1 – цепочка минимальной длины, переводящая расписание HP^1 в расписание HP^2, и S_2 – цепочка минимальной длины, переводящая расписание HP^2 в расписание HP^3. По определению метрики $L(S_1) = \rho(HP^1,\ HP^2)$ и $L(S_2) = \rho(HP^2,\ HP^3)$. Построим цепочку $S = S_1 + S_2$. Очевидно, что $S(HP^1) = HP^3$, и цепочка S допустима. При этом $L(S) = L(S_1) + L(S_2)$. Искомое неравенство вытекает из неравенства $\rho(HP^1,\ HP^3) \leq L(S)$. Выполнение свойств $1-3$ позволяет заключить, что функция $\rho(HP^1,\ HP^2)$ действительно является метрикой в пространстве HP^*_{1-5} корректных расписаний.

5. СТРАТЕГИИ ПРИМЕНЕНИЯ ОПЕРАЦИЙ ПРЕОБРАЗОВАНИЯ ТЕКУЩЕГО РЕШЕНИЯ

Стратегия уменьшения задержек. Эта стратегия основана на следующем утверждении. Если время начала выполнения каждого рабочего интервала равно длине критического пути в графе H от истоков до рабочего интервала, то расписание будет оптимальным. Длина критического пути является минимально возможным временем начала выполнения рабочего интервала и равна сумме времен выполнения рабочих интервалов, соответствующих вершинам критического пути. Разница между минимально возможным временем начала выполнения рабочего интервала и временем, когда рабочий интервал начал выполняться в расписании, является задержкой рабочего интервала. При применении операций $O1, O2$ делается попытка уменьшить задержки рабочих интервалов.

Стратегия заполнения простоев. Эта стратегия основана на утверждении: если загрузка всех процессоров равна 100%, то расписание является оптимальным. При применении операций $O1, O2$

делается попытка уменьшить суммарный простой процессоров.

Однако, не для всех индивидуальных задач возможно выполнение для оптимального расписания условий этих двух утверждений. Поэтому введена смешанная стратегия.

Смешанная стратегия. Смешанная стратегия объединяет две предыдущих. Во временной диаграмме выполнения расписания выделяются интервалы простоя процессоров и вычисляются задержки рабочих интервалов. При применении операций $O1, O2$ делается попытка уменьшить задержку рабочих интервалов путем перемещения их в интервалы простоя процессоров.

Возможны следующие схемы реализации стратегий:

- случайный выбор операций и их параметров из допустимого диапазона;
- детерминированный выбор операций на основе эвристических критериев;
- комбинированные схемы, сочетающие первые две.

Скорость сходимости алгоритма при использовании схемы со случайным выбором операции ниже, чем скорость сходимости алгоритма при использовании схемы с детерминированным применением операций. Однако использование схемы с детерминированным применением операций может приводить к зацикливанию алгоритма, когда на очередной итерации выполняется операция, обратная операции примененной на предыдущей итерации [10]. Комбинированная схема позволяет избежать зацикливания алгоритма и решить проблему низкой скорости сходимости алгоритма.

Более подробно со стратегиями применения операций и схемами их реализации можно ознакомиться в работах [10–13].

6. СПОСОБ ПРЕДСТАВЛЕНИЯ РАСПИСАНИЙ ДЛЯ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ

В данном разделе рассмотрим способ представления расписаний, позволяющий использовать известные операции мутации и скрещивания для целочисленного кодирования решений, для элементов которого заданы допустимые диапазоны изменения значений. В отличие от способа представления расписания, рассмотренного в разделе 3, порядковый номер выполнения рабочего интервала на процессоре восстанавливается по значению его приоритета. Это позволяет задавать ограничения на изменения значений параметров кодирующих расписание в виде $a_i \le y_i \le b_i$, $y_i \in Z$ и использовать любые известные операции мутации и скрещивания для целочисленного кодировании решений.

Представление расписаний с использованием приоритетов. Расписание задается вектором Y_{N+K} :

$$Y_{N+K} \equiv \left(\bigcup_{i=1}^{k} \langle YPR \rangle_i \right)$$

$$\langle YPR \rangle_i \equiv \left(\langle YE \rangle_i \bigcup \langle YP \rangle \right)$$

$$\langle YP \rangle \equiv \left(\bigcup_{j=1}^{N_i} \langle YP \rangle_j \right)$$

K – число процессов в H, N_i – число рабочих интервалов в i-ом процессе, $N = \sum_{i=1}^{K} N_i$ – число рабочих интервалов в H, $Y \equiv (\langle y1 \rangle \bigcup \langle y2 \rangle)$ – операция построения вектора $Y = (y1, y2)$ из параметров $\langle y1 \rangle$ и $\langle y2 \rangle$.

Параметр $\langle YE \rangle_i \in [1, \ldots, K] \subset Z$ содержит номер процессора, на котором выполняются рабочие интервалы i-го процесса, т.е. параметры $\langle YE \rangle$ однозначно определяют распределение рабочих интервалов по SP (привязку). Параметры $\langle YE \rangle$ могут принимать значения от 1 до K. Если значения всех параметров $\langle YE \rangle$ равны, то все рабочие интервалы выполняются на одном процессоре, если все значения различны, то рабочие интервалы каждого процессора выполняются на своем процессоре. В силу ограничения 5, максимальное число не пустых процессоров равно числу процессов K в H. Если число процессоров в ВСРВ фиксировано, то значение K равно заданному числу процессоров.

Параметры $\langle YP \rangle \in [1, \ldots, L] \subset Z$ используются алгоритмом восстановления расписания (определение отношения полного порядка в каждом SP_i) в качестве приоритетов рабочих интервалов.

При данном способе представления расписания число целочисленных переменных равно $N + K$.

Для описания алгоритма восстановления расписания HP множество рабочих интервалов разобьем на три подмножества: $P = P1 \bigcup P2 \bigcup P3$. $P1$ – множество рабочих интервалов, распределенных в SP (определен порядковый номер рабочего интервала) алгоритмом восстановления на предыдущих шагах; $P2$ – множество рабочих интервалов, у которых все предшественники принадлежат $P1$; $P3$ – множество рабочих интервалов, у которых хотя бы один из предшественников не принадлежит $P1$.

Алгоритм восстановления полного порядка рабочих интервалов в SP ($A1$).

1. Начальное разбиение множества P:
   $P1 = \varnothing$;
   $P2 = \{p_j : \forall j,\ k \in [1, \ldots, N] | \prec_{jk} = \varnothing\}$ – множество рабочих интервалов в H без предшественников;
   $P3 = \{p_j : p_j \in P \backslash P2\}$ – множество рабочих интервалов в H, у которых имеются предшественники.

2. Находим в $P2$ рабочий интервал с наименьшим значением параметра $\langle YP \rangle$ (если таких интервалов более одного, то выбираем интервал с наименьшим номером):

   - размещаем его в конец соответствующего списка SP (номер списка определяется значением параметра $\langle YE \rangle$);
   - переносим его из $P2$ в $P1$.

3. Проверяем $P3$ с целью возможности переноса рабочих интервалов в $P2$: если есть рабочие интервалы, у которых все предшественники принадлежат $P1$, то переносим их в $P2$.

4. Если $P2 \neq \varnothing$, то переходим к п. 2, иначе завершаем работу.

Утверждение 1. Алгоритм $A1$ восстановления расписания по его параметрическому представлению Y_{N+K} получает расписание $HP \in HP^*_{1-5}$, и расписание восстанавливается однозначно.

Теорема 2. Любое корректное расписание $HP \in HP^*_{1-5}$ может быть задано параметрическим представлением с использованием приоритетов (Y_{N+K}) и однозначно восстановлено алгоритмом $A1$, если допустимая верхняя граница L значений параметров $\langle YP \rangle$ больше или равна числу рабочих интервалов ($L \geq N$).

Доказательства утверждения 1 и теоремы 2 приведены в работе [9].

Операции преобразования решения

При параметрическом представлении расписания с использованием приоритетов операции преобразования решения могут быть введены как операции изменения значений параметров $\langle YE \rangle$ и $\langle YP \rangle$:

- параметры $\langle YE \rangle$ могут принимать целочисленные значения в диапазоне $[1, \ldots, K]$;
- параметры $\langle YP \rangle$ могут принимать целочисленные значения в диапазоне $[1, \ldots, L]$;
- количество изменяемых параметров может изменяться от 1 до $N + K$.

Операции преобразования решений, удовлетворяющие условиям 1–3 при использовании алгоритма восстановления $A1$, будут получать решения, удовлетворяющие ограничениям на корректность расписаний 1–5, что следует непосредственно из утверждения 1. Из теоремы 2 следует, что данные операции преобразования решения и алгоритм восстановления $A1$ позволяют получить любой допустимый вариант решения. Поскольку ограничения на изменения значений параметров задаются в виде $a_i \leq y_i \leq b_i$, $y_i \in Z$, то могут быть использованы любые известные операции мутации и скрещивания, используемые при целочисленном кодировании решений.

7. ИЗВЕСТНЫЕ МЕТОДЫ РАСПАРАЛЛЕЛИВАНИЯ ИТЕРАЦИОННЫХ АЛГОРИТМОВ

При использовании имитационных моделей с высокой степенью детализации время работы алгоритма построения расписания может быть неприемлемо большим. В данном разделе рассмотрим известные методы распараллеливания итерационных алгоритмов.

Асинхронный параллельный алгоритм. Данный метод подробно рассмотрен в работе [14] для решения задачи коммивояжера с использованием алгоритма имитации отжига. Параллельные процессы независимо выполняют алгоритм имитации отжига, стартуя с различных начальных приближений. По окончании работы алгоритма процессы отсылают наилучшее решение координатору, который выбирает среди них оптимальное.

Основными достоинствами данного метода являются простота распараллеливания на любое число процессоров и низкие требования к среде обмена. В работе [14] показано, что скорость поиска решений и качество получаемых решений практически не отличаются от последовательного алгоритма. Техника распараллеливания с помощью асинхронного параллельного алгоритма является универсальной и может быть использована для любых итерационных алгоритмов.

Параллельный алгоритм с синхронизацией. В работах [14–17] рассматриваются различные методы распараллеливания алгоритма имитации отжига, объединенные общей идей. Параллельные процессы начинают поиск решения с одного исходного приближения. Через фиксированное число итераций процессы обмениваются своими решениями и корректируют свое текущее приближение.

Существуют различные схемы обмена полученными решениями и стратегии принятия решения в качестве текущего. Например:

1. Процесс i посылает процессу $i+1$ своё текущее решение. Процесс $i+1$ принимает это решение в качестве текущего, если $f(X_i) < f(X_{i+1})$. Где $f(X_i)$ – значение целевой функции текущего решения i-ого процесса.

2. Широковещательный обмен решениями. Принятие j-ым процессом решения от i-ого процесса в качестве текущего, если $f(X_i) < f(X_j)$ и $i < j$.

3. Отправка текущих решений координатору, выбор из них произвольного решения (возможно с разными вероятностями, в зависимости от значений целевой функции) и принятие его в качестве текущего всеми процессами.

Похожая техника распараллеливания рассмотрена в работе [19]. На разных процессорах алгоритм имитации отжига работает при различных температурах и режимах понижения температуры. Процесс с номером $i+1$ отвечает за локализацию решения, полученного процессом с номером i, алгоритм имитации отжига процесса $i+1$ работает при меньшей температуре и более медленном режиме понижения температуры. Если процесс i получает решение X_i лучше, чем текущее решение X_{i+1} процесса $i+1$, то решение X_i принимается в качестве текущего для процесса $i+1$.

Параллельные генетические алгоритмы с синхронизацией основаны на “модели островов”, где каждая популяция развивается на своем острове и между островами производится обмен особями. Существуют различные схемы обмена особями. Обзор таких схем приведен в работе [19].

Среди достоинств данного метода стоит отметить хорошую масштабируемость и универсальность, из недостатков – высокие требования к пропускной способности среды обмена и несущественное уменьшение времени работы алгоритма с ростом числа процессоров.

Алгоритм, основанный на декомпозиции целевой функции.

Большую часть времени итерационный алгоритм затрачивает на вычисление целевой функции. Если целевая функция является декомпозируемой $F(x_1,\ x_2, \ldots,\ x_i, \ldots,\ x_n) = F(x_1) + F(x_2) + \cdots + F(x_i) + \cdots + F(x_n)$, то возможно параллельное её вычисление на $m (m \le n)$ процессорах. В работах [20–22] рассмотрены задачи, для которых удается существенно повысить скорость работы алгоритма имитации отжига благодаря использованию декомпозиции целевой функции.

Несомненным достоинством данного метода является практически линейный рост производительности с ростом числа процессоров, тем не менее, возможности по масштабированию ограничены видом целевой функции. Главным недостатком является узкая направленность метода, возможность его применения только для задач с декомпозируемой целевой функцией.

Методы, основанные на декомпозиции пространства решений.

Для некоторых задач возможно разбить все

пространство решений на области [23, 24]. В таком случае возможна параллельная работа итерационного алгоритма в различных областях. Для каждой конкретной задачи разбиение на области осуществляется отдельно, с учётом индивидуальных особенностей пространства решений. При использовании такого способа распараллеливания необходимо обеспечивать полное покрытие пространства решений непересекающимися областями. Следует позаботиться об изменении операций преобразования решений таким образом, чтобы полученное на следующем шаге решение не выходило за рамки области. Итоговое решение получается как наилучшее среди решений, полученных во всех областях.

К преимуществам данного метода следует отнести: сужение пространства поиска решений, значительное уменьшение времени работы алгоритма и хорошие возможности по масштабированию (при увеличении количества процессоров необходимо лишь увеличить количество областей разбиения или иначе распределять области между процессорами). Основным недостатком является узкая направленность метода. Для каждого приложения итерационного алгоритма надо разрабатывать свою схему разбиения на области и соответствующим образом менять операции преобразования текущего решения.

8. МЕТОД РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ ПОСТРОЕНИЯ РАСПИСАНИЙ

В данном разделе рассмотрим построение параллельных алгоритмов построения расписаний, основанных на декомпозиции пространства расписаний на области. Для этого множество всех возможных решений HP^*_{1-5} задачи построения расписаний представляется совокупностью областей $HP_1, HP_2, \ldots, HP_k$. Такое разбиение должно удовлетворять следующим условиям:

1. $HP_1 \cup HP_2 \cup \cdots \cup HP_k = HP^*_{1-5}$.
2. $HP_i \cap HP_j = \varnothing,\ \forall i,\ j : i \neq j$.
3. Введённые в HP^*_{1-5} операции преобразования должны быть замкнутыми на областях $HP_1, HP_2, \ldots, HP_k$ и сохранять на них свойство полноты.

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:

- разбиение исходного пространства корректных расписаний на несколько непересекающихся областей, дающих в объединении все пространство;
- выбор начального корректного расписания в каждой из областей;
- модификация операций преобразования расписаний таким образом, чтобы модифицированные операции были замкнуты в каждой из областей и сохраняли все свойства базовых операций;
- выбор способа распределения областей по узлам вычислительной системы и схемы отсечения областей.

Разбиение исходного пространства решений на области 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 вто-

рой рабочий интервал выполняется после первого;

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

На рис. 1. 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, 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 назначен лишь на один процессор.

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.

Алгоритм разбиения исходного пространства решений на области. Введём 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.

Шаг 1. Задать количество областей разбиения.

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

Шаг 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.*

Шаг 3.1. $\hat{P} = P$, *где* $\hat{P}$ *– временное множество рабочих интервалов. В начале каждой итерации оно совпадает с множеством* P *всех рабочих интервалов.*

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

Шаг 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*

Рис. 1. Принцип разбиения пространства на области.

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* $(P^{p_j}_{anc}, \prec^{p_j}_{anc}, \tilde{J}, \tilde{K})$.

Шаг 3.4. Построить графы H_1, H_2, H_3 *для областей* HP_1, HP_2, HP_3 :

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

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

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

Шаг 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 счёт 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 построения расписания), в которых критический путь больше времени выполнения расписаний, полученных при запуске алгоритма построения расписания в других областях.

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

Введём 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; $I_2 = \bigcup_{p_i \in I_1} \{p_j \in P | (p_j, p_i) \in K\}$ – множество рабочих интервалов, для которых запрещено выполнение на одном процессоре с рабочим интервалом p; SP – множество всех процессоров; SP^p_{HP} – процессор, на который распределён рабочий интервал p в расписании HP.

Алгоритм выполнения операции O_1 включает следующие шаги:

Шаг 1. Случайным образом выбирается рабочий интервал p.

Шаг 2. Строится множество процессоров SP', *на которые возможно перенести рабочий интервал* p: $SP' = SP \setminus \bigcup_{p_k \in I_2} SP^{p_k}_{HP}$.

Шаг 3. Случайным образом выбирается процес-

сор из SP' и на него переносятся все рабочие интервалы из множества I_1.

Алгоритм выполнения операции O_2 включает следующие шаги:

Шаг 1. Случайным образом выбирается рабочий интервал p.

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

Шаг 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 всём пространстве решений, но применяются к графам с дополнительными дугами, то сохраняется свойство полноты и замкнутости для области.

Распределение областей по узлам вычислительной системы осуществляется так, чтобы обеспечить максимально равномерную загрузку процессоров в течении 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 $j : j\ mod\ m = i (1 \le j \le n)$. (рис. 2.).

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 уменьшение 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 уменьшение 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.

В работе [11] рассмотрен параллельный алгоритм имитации отжига для решения задачи построения расписания с минимальным временем выполнения на исходно заданном числе процессоров и приведены результаты его экспериментального исследования. Последовательный алгоритм с разбиением пространства решения на области позволяет уменьшить время решения задачи в 3 раза по сравнению с классическим алгоритмом имитации отжига при сохранении, и во многих случаях даже при улучшении качества получаемых решений. 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

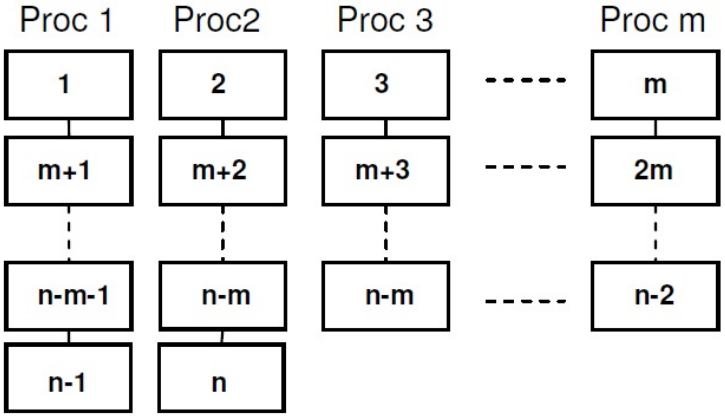

Рис. 2. Распределение областей по узлам вычислительной системы.

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.

9. ЗАКЛЮЧЕНИЕ

Разработанный подход к построению итерационных алгоритмов был использован для построения алгоритмов решения следующих задач:

1. Построение расписания с минимальным временем выполнения на исходно заданном числе процессоров. Для решения этой задачи были разработаны: последовательный алгоритм имитации отжига [10], последовательный алгоритм имитации отжига с разбиением пространства решений на области [11, 12], параллельный алгоритм имитации отжига с разбиением пространства решений на области [11, 12], локально-оптимальный алгоритм [25]. Для построения временной диаграммы выполнения расписания использовались имитационные модели ВСРВ, построенные в среде Диана [1, 26]. Вычислительная сложность рабочего интервала в зависимости от степени детализации имитационной модели может задаваться или временем его выполнения или кодом на языке Си.

2. Определение минимального необходимого числа процессоров и построение расписания выполнения прикладных программ. При этом должны выполняться заданные ограничения на время выполнения расписания и требования к надежности ВСРВ. Для решения этой задачи был разработан последовательный алгоритм имитации отжига [13, 27]. Архитектура ВСРВ: процессоры однородные по производительности и надежности, среда обмена – коммутатор FibreChannel.

3. Построение архитектуры ВСРВ минимальной стоимости и расписания время выполнения которого не превышает заданный директивный срок. Стоимость архитектуры определяется характеристиками среды обменов, числом и типом процессоров. Для решения этой задачи был разработан эволюционный алгоритм. Допустимые топологии сред обмена: полносвязная, набор шин с централизованным управлением, локально-связная, трехмерный тор, гиперкуб.

4. Определение минимального необходимого числа процессоров и построение расписания, время выполнения которого не превышает заданный директивный срок. Для решения этой задачи был разработан генетический алгоритм [28]. Данный алгоритм был использован при построении вычислительных

систем цифровой обработки сигналов от фазированных антенных решеток [29, 30].

СПИСОК ЛИТЕРАТУРЫ

1. *Smeliansky R.L., Bakhmurov A.G.* DYANA: An Environment for Distributed System Design and Analysis // Proc. The 32nd Annual Simulation Symposium, San Diego, California, 1999. P. 50–57.
2. *Смелянский Р.Л.* Модель функционирования распределенных вычислительных систем // Вестн. Моск. Ун-та. Сер. 15. Вычисл. Матем. и Кибернетика. 1990. № 3. С. 3–21.
3. *Смелянский Р.Л.* Об инварианте поведения программ // Вестн. МГУ. Сер. 15. Вычислительная математика и Кибернетика. 1990. № 4. С. 54–60.
4. *Растригин Л.А.* Статистические методы поиска. М.: Наука, 1968.
5. *Уоссермен Ф.* Нейрокомпьютерная техника. Теория и практика. М.: Мир, 1992. 240 с.
6. *Holland J.N.* Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.
7. *Deb K., Goyal M.* A Combined Genetic Adaptive Search GeneAS for Engineering Design. Computer Science and Informatics. 1996. 26(4). P. 30–45.
8. Deb K., Agrawal R.B. Simulated binary crossover for continuous search space // Complex Systems. 1995. № 9. P. 115–148.
9. *Kostenko V.A.* The Problem of Schedule Construction in the Joint Design of Hardware and Software // Programming and Computer Software. 2002. V. 28. № 3. P. 162–173.
10. *Костенко В.А., Калашников А.В.* Исследование различных модификаций алгоритмов имитации отжига для решения задачи построения многопроцессорных расписаний // Дискретные модели в теории управляющих систем. Труды VII Международной конференции. М.: МАКС Пресс, 2006. С. 179–184.
11. *Kalashnikov A.V., Kostenko V.A.* A Parallel Algorithm of Simulated Annealing for Multiprocessor Scheduling // J. of Computer and Systems Sciences International. 2008. V. 47. № 3. P. 455–463.
12. *Калашников А.В., Костенко В.А.* Итерационные алгоритмы построения расписаний, основанные на разбиении пространства решений на области // Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. 2008. № 3. С. 56–60.
13. *Zorin D.A., Kostenko V.A.* Algorithm for Synthesizing a Reliable Real_Time Computing System Architecture // J. of Computer and Systems Sciences International. 2012. V. 51. № 3. P. 410–417.
14. *Czech Z.J.* Parallel Simulated Annealing for the Delivery Problem. Ninth Euromicro Workshop on Parallel and Distributed Processing. 2001. Mantova Italy. P. 219–226.
15. *Schmid M., Schneider R.* Parallel Simulated Annealing Techniques for Scheduling and Mapping DSP-Applications onto Multi-DSP Platforms // In Proceedings of the International Conference on Signal Processing Applications Technology. 1999. Orlando, U.S.A.: Miller Freeman.
16. *Varanelli J.M.* On the Acceleration of Simulated Annealing // PhD thesis, University of Virginia, USA. 1996. P. 77–81.
17. *Chinyao L.* Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines // Computers and operations research. 2005. V. 32. № 8. P. 2013–2025.
18. *Gomez J.C.* A General Interface for Distributed and Sequential Simulated Annealing. http://citeseer.ist.psu.edu/110625.html.
19. *Скобцов Ю.А.* Основы эволюционных вычислений. Донецк.: ДонНТУ, 2008. 326 с.
20. R. Azencott, Ed., Simulated Annealing: Parallelization Techniques. New York: John Wiley and Sons. 1992. P. 47–79.
21. *Kravitz S.A., Rutenbar R.A.* Placement by Simulated Annealing on a Multiprocessor // IEEE Trans. CADICS. 1987. № 6. P. 534–549.
22. *Fidanova S.* Simulated Annealing for Grid Scheduling Problem // IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA'06). 2006. P. 41–45.
23. *Ram D.J., Sreenivas T.H., Subramaniam K.G.* Parallel Simulated Annealing Algorithms // J. of parallel and distributed computing. 1996. № 37. P. 207–212.
24. *Allwright J., Carpenter D.* A distributed implementation of simulated annealing for traveling salesman problem // Parallel Computing. 1989. V. 3. № 9–10. P. 335–338.
25. *Калашников А.В., Костенко В.А.* Алгортимы локальной оптимизации расписаний // Труды Всероссийской научной конференции "Методы и средства обработки информации" (1 октября–3 октября 2003 г., г. Москва) М.: Издательский отдел факультета ВМиК МГУ, 2003. С. 381–388.

26. *Бахмуров А.Г., Костенко В.А., Смелянский Р.Л.* Среда моделирования DYANA: синтез, анализ и оптимизация вычислительных систем реального времени // Сборник докладов 1-й Международной конференции "Цифровая обработка сигналов и ее применения" (30 июня–3 июля 1998, Москва). МЦНТИ. Т. IV. С. 152–156.

27. *Zorin D.A., Kostenko V.A.* Co-design of Real-time Embedded Systems under Reliability Constraints // Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDeS). Brno, Czech Republic, 2012. P. 392–396.

28. *Kostenko V.A., Smeliansky R.L., Trekin A.G.* Synthesizing Structures of Real-Time Computer Systems Using Genetic Algorithms // Programming and Computer Software. 2000. V. 26. № 5. P. 281–288.

29. *Kostenko V.A.* Large-grain parallelism in signal processing problems //Programming and Computer Software. 1997. V. 23. № 2. P. 109–115.

30. *Kostenko V.A.* Design of computer-systems for digital signal-processing based on the concept of open-architecture // Automation and Remote Control. 1994. V. 55. № 12. P. 1830–1838.