© 2012 г. Д.А. ЗОРИН
(Московский Государственный Университет имени М.В. Ломоносова, факультет Вычислительной Математики и Кибернетики)

СРАВНЕНИЕ РАЗЛИЧНЫХ СТРАТЕГИЙ ПРИМЕНЕНИЯ ОПЕРАЦИЙ В АЛГОРИТМЕ ИМИТАЦИИ ОТЖИГА ДЛЯ ЗАДАЧИ ПОСТРОЕНИЯ РАСПИСАНИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ СИСТЕМ

В данной работе рассматривается алгоритм построения вычислительной системы с минимальным числом процессоров при соблюдении ограничений на надежность системы и время работы программы. Проблема сформулирована математически, предложен алгоритм на основе метода имитации отжига. Приведено сравнение различных стратегий поиска решений для данного алгоритма.

COMPARISON OF DIFFERENT OPERATION APPLICATION STRATEGIES IN SIMULATED ANNEALING ALGORITHM FOR SCHEDULING ON MULTIPROCESSORS / D.A. Zorin (Lomonosov Moscow State University, Department of Computational Mathematics and Cybernetics, E-mail: juan@lvk.cs.msu.su). This paper describes a method of designing a computational system with the minimal number of processors. Strict execution deadlines apply, as well as restrictions on the reliability of the system that imply the tolerance of the system to both hardware and software faults. The problem is formulated mathematically, an algorithm of solution based on simulated annealing is described, and experimental results are shown.

1. Введение

В данной работе рассматривается задача построения вычислительных систем, использующих наименьшее возможное число аппаратных ресурсов для выполнения прикладной программы за время, не превышающее заданного. Задачи такого типа возникают при проектировании систем реального времени, где есть ограничения на время выполнения программы, из-за которых появляется необходимость запускать программу параллельно на нескольких процессорах [1].

Предполагается, что программа допускает распараллеливание, то есть в ней есть фрагменты, не зависящие друг от друга по данным. Эти фрагменты можно выполнять одновременно и по завершении передавать результат фрагментам, использующим эти данные. Распределение таких фрагментов (далее называемых «заданиями») на несколько процессоров с указанием очередности выполнения называется расписанием.

Системы жесткого реального времени — это такие системы, в которых накладывается ограничение на время выполнения программы, и завершение программы не позже директивного срока является обязательным требованием к расписанию.

Надежностью называют свойство системы работать без ошибок. В дальнейшем под надежностью вычислительной системы будем подразумевать вероятность того, что, начиная с некоторого момента времени, вычислительная система будет работать по спецификации в течение заданного времени. Вычислительные системы с высокой надежностью применяются в тех областях, где безотказная работа особенно важна — авиакосмической промышленности, ядерной энергетике, медицинской промышленности. Под отказоустойчивостью понимают свойство системы (или ее компонента) сохранять работоспособность в случае возникновения отказов.

В данной задаче заранее определены требования к работе в режиме жесткого реального времени и требования по надежности: она должна быть не ниже определенной величины. Повышать надежность системы можно с помощью введения резервных элементов — как процессоров, так и копий программы. Доступны такие методы, как аппаратное резервирование и N-версионное программирование.

В работе предложен алгоритм решения этой задачи. Алгоритм допускает настройку на решение частных задач (заданы ограничения на возможные значения входных данных задачи) путем задания соответствующих значений параметров алгоритма. Также алгоритм допускает использование различных методик расчета надежности ВС РВ и имитационных моделей ВС РВ с различным уровнем детализации для оценки времени выполнения расписания. Это позволяет использовать алгоритм на различных этапах проектирования ВС РВ и делает возможным использования алгоритма в различных инструментальных средствах проектирования и отладки программного обеспечения ВС РВ.

2. Постановка задачи

2.1. Модель системы

Аппаратная часть системы состоит из множества процессоров соединенных друг с другом некоторой средой передачи данных

Программа, для которой строится расписание, состоит из конечного множества взаимодействующих заданий. Для каждого из заданий известно, какие вычисления оно производит и каков объем результата, получаемого на выходе. Программу можно представить в виде графа потока данных.

Формально модель системы состоит из следующих объектов:

- M – множество процессоров. Является счетным.
- G = {V, E} – граф потока данных программы, ориентированный граф без циклов.
- V – множество вершин, соответствующих заданиям.
- E – множество ребер. В графе есть ребро (v_1, v_2), если задание v_2 принимает на вход результаты работы задания v_1.
- $F: E \to \mathbb{R}$ – функция, задающая объем передаваемых данных для каждой взаимодействующей пары заданий.

2.2. Модель надежности

В данной работе будут рассмотрены два метода повышения отказоустойчивости: резервирование для процессоров и многоверсионное программирование.

Резервирование процессоров заключается в том, что в систему добавляется новый процессор, на котором выполняются те же задания, что и на другом. В этом случае система отказывает, только если отказывают оба процессора. Дублирующий процессор работа-

ет в режиме горячего резерва, то есть принимает все те же данные и выполняет те же вычисления, что и основной, но передает данные только в случае отказа основного. Устройство коммутатора таково, что введение резервного процессора не вызывает никаких задержек в работе системы.

При многоверсионном программировании создается несколько версий реализации какого-либо задания, и считается, что если больше половины из них выдали идентичный результат, то задание завершается успешно.

Для задания модели надежности системы необходимо, чтобы были известны величины $P(m_i)$ – надежность отдельного процессора, $Vers(v_i)$ – множество доступных версий каждого из заданий, $P(v_i)$ – надежность отдельного задания с учетом количества версий. Формулы для вычисления $P(v_i)$ при различных конфигурациях версий приведены в [2-5], а формулы для вычисления общей надежности – в разделе 2.4.

2.3. Модель расписания

Будем считать, что для программы задано расписание, если для каждого из заданий определена привязка – однозначно определено, на каком процессоре оно выполняется, и задан порядок – для каждого процессора известно, в какой очередности выполняются задания.

Если используется многоверсионное программирование, то помимо указания задания, необходимо указывать номер его версии, то есть привязка и порядок задаются не для задания, а для пары «задание-версия».

Формально расписание (для системы с резервированием и многоверсионным программированием) определяется как пара (S, D), где S – множество четверок (v, k, m, n) где $v \in V, k \in Vers(v), m \in M, n \in \mathbb{N}$, такое что

$$\forall v \in V\ \forall k \in Vers(v):\ \exists!\, s=(v_i, k_i, m_i, n_i) \in S: v_i=v,\ k_i=k;$$

$$\forall s_i=(v_i, k_i, m_i, n_i) \in S,\ \forall s_j=(v_j, k_j, m_j, n_j) \in S:\ (s_i \neq s_j \wedge m_i=m_j) \Rightarrow n_i \neq n_j.$$

D – мультимножество, состоящее из элементов множества процессоров M. Содержательно m и n задают соответственно привязку к процессору и порядок выполнения для каждой версии каждого задания. Мультимножество D задает резервируемые процессоры.

Расписание можно представить в виде графа. Вершинами этого графа являются элементы множества S. Если между соответствующими заданиями есть дуга в графе G, то она добавляется в граф расписания, также в граф расписания добавляются дуги между вершинами, назначенными на один процессор и имеющими соседние номера.

Элемент расписания s_2 будем называть зависимым от s_1, если либо $(v_1, v_2) \in E$, либо $m_1 = m_2 \wedge n_1 < n_2$. Иначе говоря, s_2 зависит от s_1, если s_2 не может начать выполняться раньше s_1 из-за зависимости по данным или порядка выполнения на процессоре.

Из определения следует, что каждая версия каждого задания может присутствовать в расписании в единственном экземпляре, а также на каждом процессоре у всех заданий различные номера и как минимум одна версия каждого задания должна быть включена в расписание. Помимо этих ограничений, необходимо ввести еще одно, чтобы гарантировать завершимость программы.

Определение. Будем говорить, что расписание S является корректным, если его граф является ациклическим. Не существует набора элементов расписания $s_1 \ldots s_n$, такого что s_i зависит от s_{i-1} для всех $i \in [2..n]$ и s_1 зависит от s_n.

В дальнейшем будем рассматривать только корректные расписания. Множество всех корректных расписаний для заданного графа программы будем обозначать $\overline{S}$.

Для каждого корректного расписания определены функции t(S) – время выполнения расписания, R(S) – надежность системы при заданном расписании, и M(S) =|D| – количество процессоров, требуемых для работы по данному расписанию.

2.4. Вычисление времени выполнения расписания и надежности

Пусть заданы программа G и расписание (S, D) для нее.

Определение. Будем говорить, что для расписания S задана временная диаграмма, если определены две функции:

$$C: S \rightarrow \mathbb{R} \times \mathbb{R},$$
$$F: E \rightarrow \mathbb{R} \times \mathbb{R},$$

удовлетворяющие следующим условиям:

$$\forall s \in S; a, b = C(s): a < b$$
$$\forall e \in E; a, b = F(e): a < b$$
$$\forall s_1, s_2 \in S; a_1, b_1 = C(s_1);\ a_2, b_2 = C(s_2):\ m_1 = m_2 \wedge n_1 < n_2 \Rightarrow b_1 \leq a_2$$
$$\forall s_i = (v_i, k_i, m_i, n_i) \in S; e = (v_j, v_i) \in E; a_1, b_1 = C(s_i);\ a_2, b_2 = F(e):\ b_2 \leq a_1$$

Определение. Функцией интерпретации называется функция, по произвольному расписанию строящая временную диаграмму:

$$ft(S): S \rightarrow (C, F)$$

Различным средам передачи данных (общая шина, различные виды коммутаторов) соответствуют разные функции интерпретации.

Определение. Время выполнения t(S), зависящее от функции интерпретации, – это время завершения последнего задания во временной диаграмме:

$$t(S) = \max_{s \in S} C_2(s)$$

Надежность R(S) вычисляется следующим образом [6]:

$$R(S) = R_H \cdot R_S$$
$$R_H = \prod_{m_i \in M} (1 - R(m_i))$$
$$R(m_i) = P(m_i)^{Dup(m_i)}$$
$$R_S = \prod_{(v_i, k_i, m_i, n_i) \in S} (1 - P(v_i))$$

Здесь R_H — надежность аппаратной части, R_S — надежность программы.

2.5. Постановка оптимизационной задачи

Пусть заданы программа *G*, t^{dir} – срок, к которому программа должна быть выполнена, и R^{dir} – надежность, которой должна обладать система.

Пусть также фиксирована функция интерпретации *ft(S).*

Необходимо построить расписание S, для которого требуется минимальное количество процессоров, но при этом выполняются ограничения на время выполнения и надежность:

$$(1)\qquad \begin{aligned} &\min_{S\in\overline{S}} M(S)\\ &t(S) < t^{dir}\\ &R(S) > R^{dir} \end{aligned}$$

Утверждение 1. Задача (1) является NP-трудной.

Доказательство данного утверждения может быть проведено путем полиномиального сведения к известной задаче о сумме подмножества.

3. Алгоритм имитации отжига

В данном разделе введем конкретизацию общей схемы алгоритма для решения задачи (1), а именно введем операций преобразования текущего решения, стратегию применения операций преобразования текущего решения на шаге 3 общей схемы алгоритма и операцию выбора текущего приближения на шаге 4.

3.1. Операции преобразования решения

Введем следующие обозначения:

- *Dup(m_i)* – кратность процессора m_i в мультимножестве *D*.
- *Dep(s)* – это множество таких элементов s_i, что в графе *G* есть ребро (v_i,v), то есть множество вершин – непосредственных предшественников *s*.
- *Succ(s)* – множество заданий косвенно зависящих от *s*. $s_i \in Succ(s)$, если в графе расписания есть путь из *s* в s_i, не содержащий ребро от *s* к следующему за ним заданию на процессоре, если такого ребра нет в графе программы.

Операция добавления резервного процессора. В исходном расписании *(S, D)* к мультимножеству *D* добавляется новый элемент. При этом создается дополнительная нагрузка на среду передачи данных, так как на резервный процессор передаются все необходимые данные, однако время передачи данных между основными процессорами не меняется, так как в силу описанного ранее устройства среды обменов для передачи данных на резервные процессоры гарантированно найдутся свободные каналы.

Операция удаления резервного процессора. В исходном расписании *(S, D)* из мультимножества *D* удаляется элемент *m*, для которого *Dup(m)* > 1.

Операция переноса задания. В исходном расписании *(S, D)* выбирается элемент $s_1=(v_1, k_1, m_1, n_1)$, процессор m_2 и номер n_2, такой что

$\forall s_i=m_i=m_2: (n_i<n_2 \Rightarrow s_i \notin Succ(s_1)) \wedge (n_i \geq n_2 \Rightarrow s_1 \notin Succ(s_i))$,

и происходит следующая замена:

$s'_1=(v_1, k_1, m_1, n_1)$, $\forall s_i: m_i=m_2: n_i \geq n_2 \Rightarrow s'_i=(v_i, k_i, m_i, n_i+1)$.

Эта операция позволяет или изменить порядковый номер выполнения задания на процессоре или перенести задание на другой процессор.

Операция добавления версии. Версии можно добавлять только парами в силу принципа работы NVP. Добавление одной версии *(v,k)* эквивалентно следующей последовательности операций: 1) добавить новый процессор m_0; 2) назначить версию первым заданием на этот процессор $s=(v, k, m_0, 1)$; 3) перенести *s* на другой процессор в соответствии с определением операции переноса задания; 4) удалить m_0.

Операция удаления версии. Версии удаляются также парами. Из расписания удаляются два элемента, соответствующих удаляемым версиям.

Утверждение 2. Если (S_1, D_1), (S_2, D_2) – корректные расписания, то существует последовательность операций, приводящая (S_1, D_1) к (S_2, D_2), такая, что все промежуточные расписания корректны.

Доказательство этого утверждения приведено в работе [7].

3.2. Стратегия применения операций преобразования текущего решения

Выбор операций происходит в зависимости от того, какое из следующих отношений имеет место.

(3)
- $t > t^{dir}$, $R > R^{dir}$
- $t > t^{dir}$, $R < R^{dir}$
- $t < t^{dir}$, $R > R^{dir}$
- $t < t^{dir}$, $R < R^{dir}$

Если надежность меньше, чем требуется, то следует добавлять процессоры и версии, иначе их можно удалять. Если время выполнения превышает директивный срок, то имеет смысл либо удалять версии, либо переносить задания на другие позиции в расписании.

Чтобы избежать потенциальных зацикливаний алгоритма, выбор операции сделан недетерминированным. Для каждой из четырех возможных ситуаций заданы вероятности выбора той или иной операций (в том числе возможны нулевые значения). Эти вероятности являются параметрами настройки алгоритма и задаются до начала работы алгоритма.

Применение некоторых операций на определенной итерации алгоритма может быть невозможно. Например, если ни у одного процессора нет резервных копий, то удалять процессоры нельзя, а если все имеющиеся версии использованы, то нельзя добавлять версии. Такую ситуацию всегда можно определить, поэтому на этом шаге невозможные операции не рассматриваются.

Добавление версии. Случайно выбирается задание, версии которого добавляются (среди заданий, для которых имеется нужное количество доступных версий). Вероятность выбора задания обратно пропорциональна количеству уже использующихся его версий.

Удаление версии. Случайно выбирается задание, версии которого удаляются. Вероятность выбора задания пропорциональна количеству уже использующихся его версий.

Добавление резервного процессора. Аналогично добавлению версии задания, процессоры с меньшим числом резервов имеют большую вероятность добавления.

Удаление резервного процессора. Удаляется резерв для случайно выбранного процессора. Чем больше резервов процессоров, тем больше вероятность его удаления.

Вероятности удаления и добавления процессоров и версий заданы таким образом, чтобы стремиться соблюдать баланс между надежностью различных компонентов.

Перенос задания.

Если $t < t^{dir}$, то следует пытаться уменьшить число процессоров. Происходит следующая операция: выбирается процессор, на котором меньше всего заданий, и все задания с него переносят на другие процессоры.

Если $t > t^{dir}$, то необходимо улучшить время выполнения расписания. Это можно сделать либо перенеся часть заданий на пустой процессор, либо поменяв порядок или привязку на существующих. Предлагаются три стратегии выбора параметров для этой операции.

Стратегия уменьшения задержек. Эта стратегия основана на следующем утверждении. Если время начала выполнения каждого задания равно длине критического

пути в графе *G* от истоков до задания, то расписание будет оптимальным. Длина критического пути является минимально возможным временем начала выполнения задания и равна сумме времен выполнения заданий соответствующих вершинам критического пути.

Для каждого элемента *s* можно определить минимально возможное время, когда *s* может начать выполняться, то есть все задания из *Dep(s)* завершены. Разница между этим временем и временем, когда задание *s* начало выполняться в расписании, является задержкой задания *s*.

В качестве параметра алгоритма задается натуральная константа C_{ver}, и в качестве первого параметра операции переноса выбирается одно из C_{ver} заданий с наибольшей задержкой, чем больше задержка – тем больше вероятность выбора задания. Далее выбирается позиция (пара *(m, n)* из четверки, задающей элемент расписания), куда можно перенести выбранное задание. Это задание переносится на случайно выбранную позицию (в которую перенос возможен без нарушения условий корректности), причем, чем раньше по времени стоит задание, занимающее эту позицию в текущий момент, тем вероятнее перенос.

Пример работы стратегии приведен на рисунке 1. Задание 4 не зависит от задания 2, поэтому перенос задания 2 на первый процессор уменьшает задержку задания 4, и общее время выполнения также уменьшается.

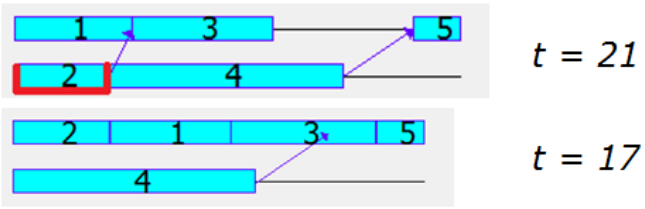

Рис.1 Пример стратегии уменьшения задержек.

Стратегия заполнения простоев. Эта стратегия основана на эмпирической гипотезе: чем меньше времени в сумме простаивают процессоры, тем лучше расписание.

Для каждой позиции *(m, n)* можно определить время простоя. Если *n*=1, то простой – время от начала работы до начала выполнения задания на позиции *(m,* 1*)*. Если позиция *(m, n)* обозначает место после завершения последнего задания на процессоре *m*, то простой – время от конца работы последнего задания на *m* до конца выполнения всего расписания. Иначе, простой позиции *(m, n)* – это время между завершением работы задания в *(m, n-1)* и началом работы задания в *(m, n)* .

Из всех позиций выбираются C_{pos} позиций с максимальным простоем, на которые допустимо переместить выбранное задание, после чего из них выбирается случайно одна позиция, причем наибольший приоритет имеют позиции, задания на которых начинаются раньше по времени. Задание для переноса выбирается случайно (при соблюдении условий корректности), при этом чем позже оно стоит, тем больше вероятность его выбора.

На рисунке 2 приведен пример применения данной стратегии. Между заданиями 5 и 2 большой простой, поместив туда задание 1, можно добиться уменьшения времени выполнения всей программы.

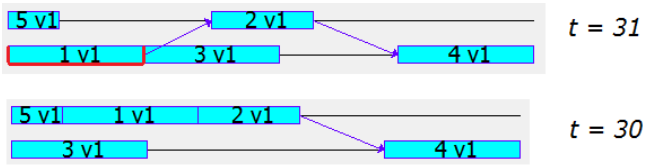

Рис.2. Пример стратегии заполнения простоев.

Смешанная стратегия. Смешанная стратегия объединяет две предыдущих. В качестве первого параметра операции переноса выбирается одно из C_{ver} заданий с наибольшей задержкой, чем больше задержка – тем больше вероятность выбора задания. Из всех позиций выбираются C_{pos} позиций с максимальным простоем, на которые допустимо переместить выбранное задание, после чего из них выбирается случайно одна позиция, причем наибольшую вероятность имеют позиции, задания на которых начинаются раньше по времени. Таким образом, в этой стратегии ищутся области в расписании, когда один из процессоров бездействует в ожидании завершения работ на других процессорах, при этом делается попытка поставить задание с большой задержкой как можно раньше, чтобы уменьшить время задержки.

3.3. Выбора текущего приближения решения

После применения выбранной операции получается новое расписание, для которого можно рассчитать время выполнения, надежность и число процессоров. В зависимости от отношений характеристик нового и предыдущего расписаний, новое расписание может стать текущим приближением на следующей итерации алгоритма. Как и в стандартном алгоритме имитации отжига, присутствует параметр, моделирующий температуру, значение которого в ходе работы алгоритма уменьшается в соответствии с выбранным законом понижения температуры.

4. Эксперименты

4.1. Сравнение качества результатов

На графиках (рис. 3-5) приведены результаты сравнения трех стратегий. Было проведено 1000 экспериментов с различными исходными данными (число заданий варьировалось от 10 до 150), на каждом алгоритм запускался поочередно с каждой из стратегий. Число итераций алгоритма каждый раз задавалось одинаковым.

Первый график (рис. 3) показывает среднее значение целевой функции в зависимости от числа заданий. Результаты для смешанной стратегии и стратегии заполнения простоев почти одинаковые, стратегия уменьшения задержек заметно хуже.

Графики на рисунках 3-4 показывают попарное сравнение стратегий. Для каждой пары стратегий температуры значения целевой функции – числа процессоров, полученные на одних и тех же исходных данных, вычислялась разница значений целевых функций. Графики показывают частоту получения различных значений этой разницы. В этом сравнении также уменьшение задержек показывает худшие результаты по сравнению с заполнением простоев, которое, в свою очередь, примерно эквивалентно по точности смешанной стратегии.

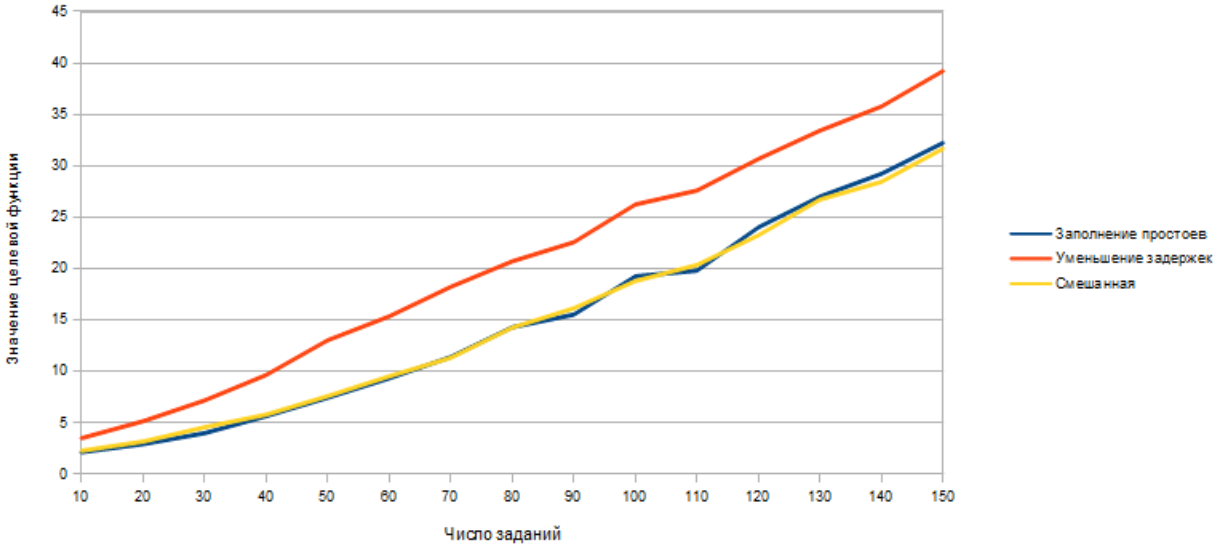

Рис.3. Среднее значение целевой функции в зависимости от числа заданий для трех стратегий

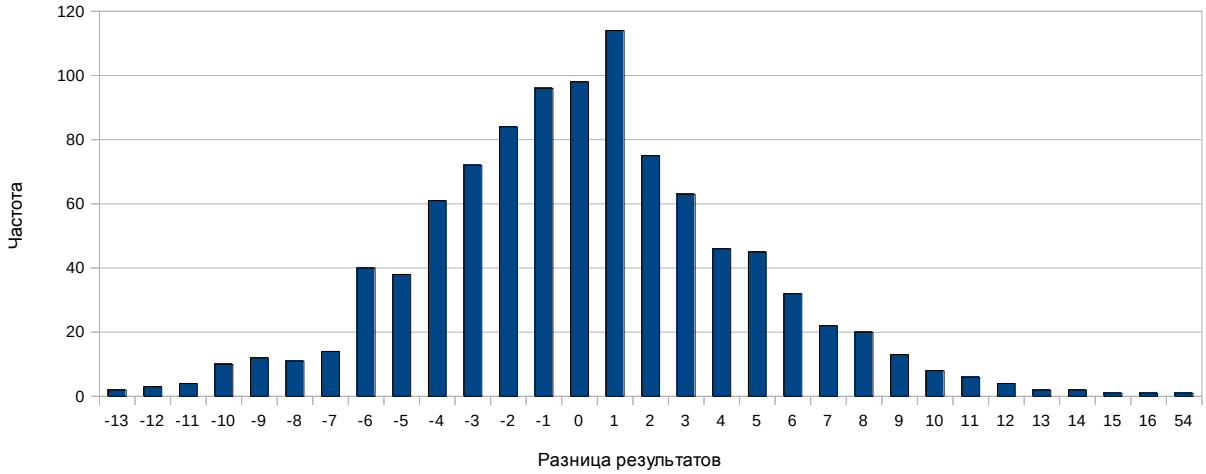

Рис.4. Сравнение смешанной стратегии и стратегии заполнения простоев (отрицательные числа в тех случаях, когда заполнение простоев лучше)

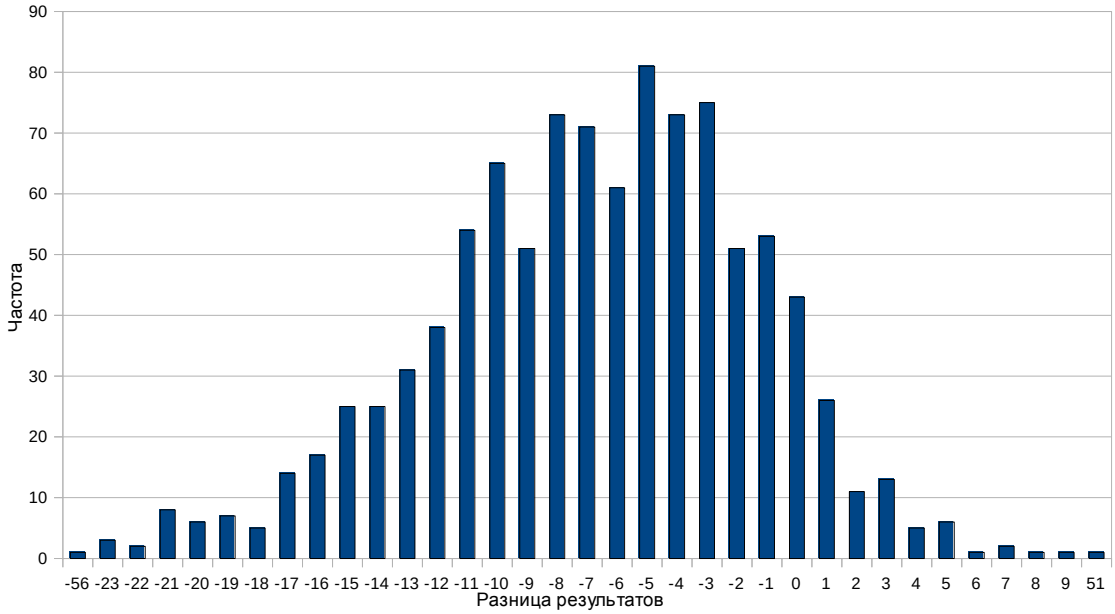

Рис.5. Сравнение стратегий уменьшения задержек и заполнения простоев (отрицательные числа в тех случаях, когда заполнение простоев лучше)

4.2. Сравнение скорости

На графиках (рис. 6-8) показаны результаты сравнения скорости работы алгоритма с разными стратегиями. По оси X на графике приведена разница (в процентах) числа итераций, которые потребовались алгоритму для достижения одного и того же значения целевой функции. Например, если с одной стратегией было найдено решение с 10 процессорами за 100 итераций, а с другой решение с 10 процессорами было достигнуто за 110 итераций, это означает, что вторая стратегия дает результат на 10% медленнее. По вертикальной оси указано число экспериментов, на которых был достигнут результат.

Следует отметить, что на отдельно взятом эксперименте такой подсчет скорости не может быть коммутативным: если в данном эксперименте первая стратегия нашла решение с 10 процессорами, а вторая – с 9, то можно сравнить только число итераций, которое потребовалось, чтобы дойти до худшего из двух результатов, то есть до 10 процессоров. Поэтому для каждой пары стратегий имеет смысл разбить все эксперименты на две группы в зависимости от того, на какой стратегии показан лучший результат. Если стратегия уменьшения задержек в большинстве случаев показывает худшие результаты, и остальными можно пренебречь, то при сравнении смешанной стратегии и стратегии заполнения простоев имеет смысл рассмотреть обе группы экспериментов.

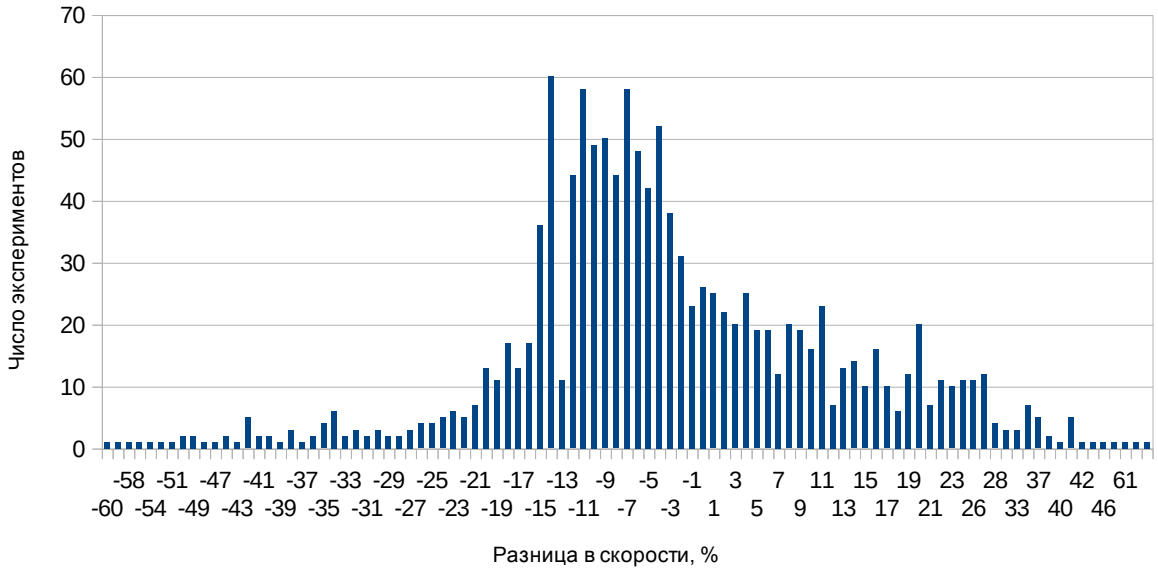

Рис. 6. Сравнения скорости работы стратегии уменьшения задержек и смешанной стратегии

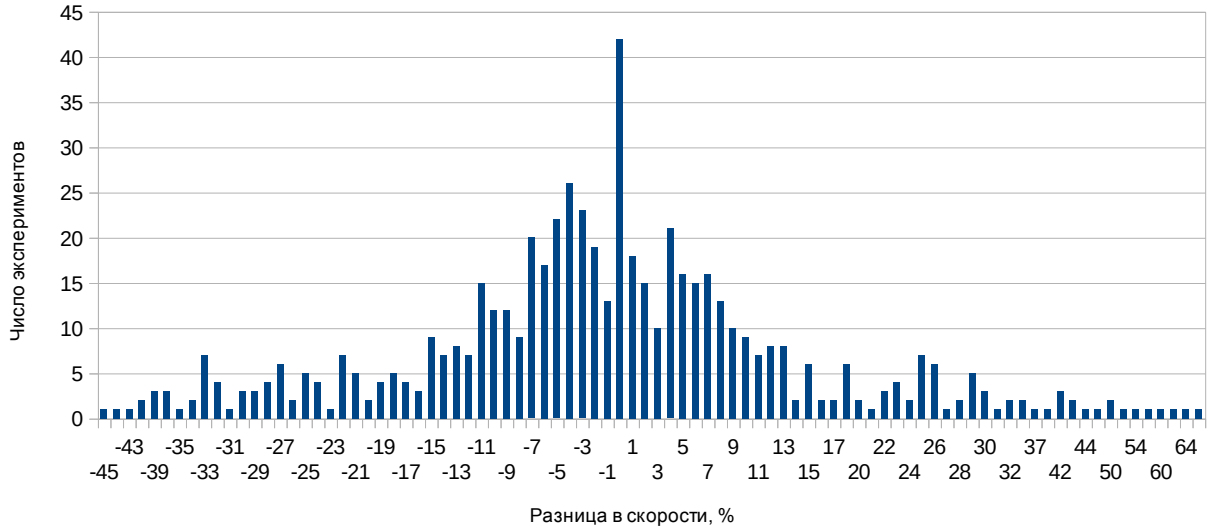

Рис. 7. Сравнения скорости работы стратегии заполнения простоев и смешанной стратегии на тех экспериментах, где стратегия заполнения простоев показывает лучший результат

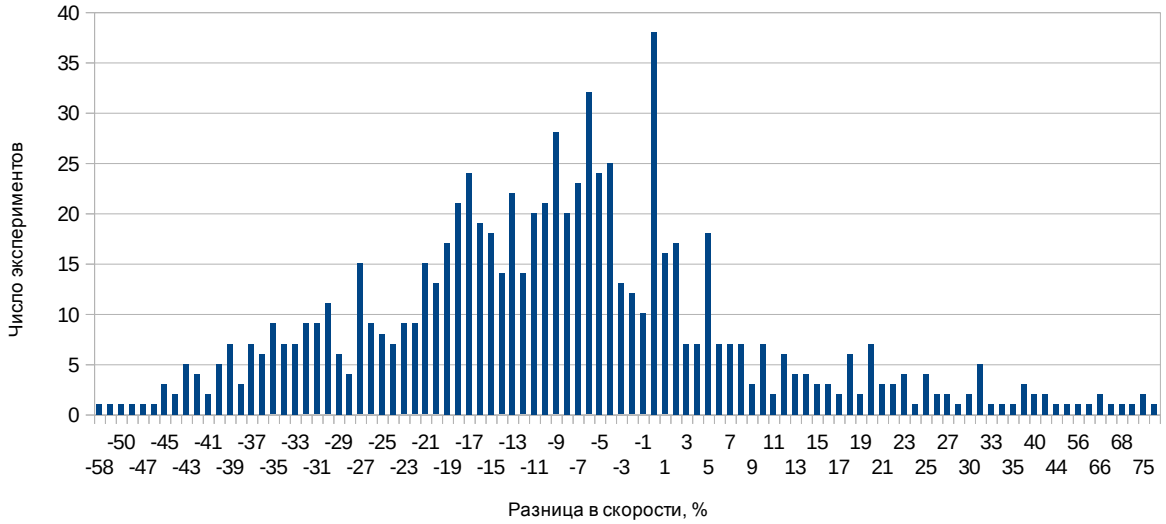

Рис. 8. Сравнения скорости работы стратегии заполнения простоев и смешанной стратегии на тех экспериментах, где смешанная стратегия показывает лучший результат

Как и следовало ожидать, стратегия уменьшения задержек не только дает менее точные результаты, но и работает медленнее. Низкую скорость сходимости можно считать одной из причин низкой точности, так как число итераций для всех трех стратегий задавалось одинаковым.

Сравнение двух других стратегий показывает, что смешанная стратегия работает в среднем на 10% быстрее.

4.3. Сравнение стабильности работы

Дополнительно было проведено тестирование стабильности работы алгоритма с каждой из стратегий. Для этого алгоритм запускался 100 раз на одних и тех же исходных данных, а не на различных, как в пункте 4.1. В данном случае важны полученные значения целевой функции и дисперсия (среднеквадратичное отклонение), показывающая, насколько сильно получаемые значения отличаются от среднего. Низкая дисперсия говорит о высокой стабильности алгоритма.

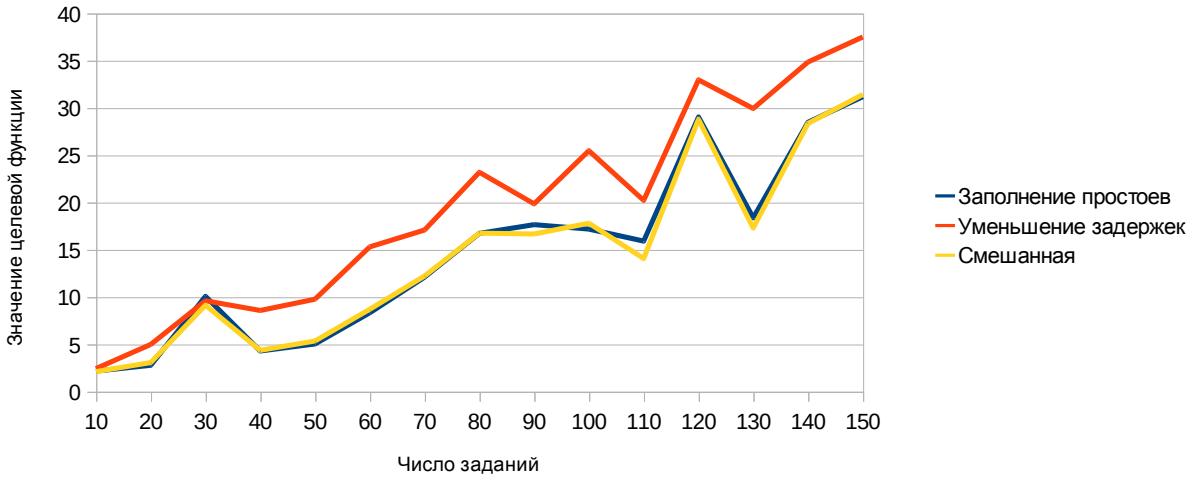

Рис.9. Среднее значение целевой функции в зависимости от числа заданий для трех стратегий

Число заданий	Заполнение простоев	Уменьшение задержек	Смешанная
10	0,36	0,67	0,3
20	1,02	1,35	1,05
30	2,52	2,01	3,13
40	1,14	2,02	1,03
50	1,1	3,96	1,32
60	2,13	3,8	2,67
70	3,41	1,96	2,87
80	2,88	3,11	3,33
90	3	3,41	3,48
100	3,93	4,16	3,35
110	3,24	3,67	3,51
120	2,18	2,82	2,24
130	2,99	3,57	2,55
140	2,66	4,13	2,47
150	2,43	5,25	3

Табл.1. Среднеквадратичное отклонение для каждой из трех стратегий

На рисунке 9 приведен график зависимости среднего значения целевой функции от числа заданий, в целом повторяющий картину из пункта 4.1. В таблице 1 приведены значения дисперсии. Отклонения для смешанной стратегий и стратегии заполнения простоев примерно равны, для стратегии уменьшения задержек это значение больше, что также свидетельствует в пользу слабости данной стратегии.

5. Заключение

В рамках данной работы была сформулирована задача синтеза архитектуры вычислительной системы реального времени с учетом требований к надежности как задача комбинаторной оптимизации. Был разработан итерационный алгоритм решения сформулированной задачи и теоретически обоснована его корректность.

Экспериментальное исследование различных стратегий для рассматриваемой задачи построения расписаний показало, что одна из рассмотренных стратегий существенно слабее двух других и нуждается в доработке. Смешанная стратегия и стратегия заполнения простоев показали приблизительно одинаковое качество результатов, смешанная стратегия имеет небольшое преимущество в скорости сходимости.

СПИСОК ЛИТЕРАТУРЫ

1. *Калашников А.В., Костенко В.А.* Параллельный алгоритм имитации отжига для построения многопроцессорных расписаний// Известия РАН. Теория и системы управления, 2008., N.3, С.101-110.
2. *Avizienis A., Laprie J.C., Randell B.* Dependability and its threats: a taxonomy. Toulouse: Building the Information Society Proc IFIP 18th World Computer Congress, 2004. P. 91-120
3. *Nicola V.F., Goyal A.* Modeling of correlated failures and community error recovery in multiversion software // Software Engineering, IEEE Transactions on . 1990. № 16 Issue:3. P. 350-359.
4. *Popov P., Strigini L., May J., Kuball S.* Estimating bounds on the reliability of diverse systems // Software Engineering, IEEE Transactions on . 2003. № 29 Issue:4. P. 345-359
5. *Eckhardt D. E., Lee L.D.* A theoretical basis for the analysis of multiversion software subject to coincident errors // IEEE Transactions on Software Engineering. 1985. № SE-11 Issue:12. P. 1511-1517.
6. *Wattanapongsakorn N., Levitan S.P.* Reliability optimization models for embedded systems with multiple applications// Reliability, IEEE Transactions on. 2004, 53, P. 406-416.
7. *Зорин Д.А.* Способ представления и преобразования расписаний в итерационных алгоритмах структурного синтеза вычислительных систем реального времени // Программные системы и инструменты. Тематический сборник № 12, М.: Изд-во факультета ВМиК МГУ, 2011. С. 163-171