Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Савицкий Илья Павлович

Жадные алгоритмы для построения многопроцессорного списочного расписания

Дипломная работа

Научный руководитель:
Доцент, к.т.н
Костенко Валерий Алексеевич

Москва, 2022

Аннотация

Построение многопроцессорного расписания это NP-трудная задача. Не существует полиномиального алгоритма В данной работе приводятся два возможных варианта решения задачи с дополнительными ограничениями на количество передач или сбалансированость распределения работ на процессорах при помощи алгоритма, сочетающего жадные стратегии и ограниченный перебор.

Содержание

1 Введение

Классическая задача построения расписания хорошо изучена и досканально описана в [3]. Поскольку данная задача принадлежит к классу NP-трудных, не существует алгоритма, который за полиномиальное время даст точный ответ, но существуют алгоритмы, которые дают приближенные результаты. Большинство таких алгоритмов резделяются на две категории: *конструктивные* и *итерационные*. Из основных примеров можно выделить (большинство из них упомянуты в [13]):

- Конструктивные алгоритмы
    1. Алгоритмы, основанные на поиске максимального потока в сети
    2. Алгоритмы, основанные на методах динамического программирования
    3. Алгоитмы, основанные на методе ветвей и границ
    4. Жадные алгоритмы
    5. Жадные алгоритмы с процедурой ограниченного перебора
- Итерационные алгоритмы
    1. Генетические алгоритмы
    2. Дифференциальная эволюция
    3. Алгоритм имитации отжига
    4. Алгоритм муравьиных колоний

Конструктивные алгоритмы работают, строя и дополняя частичные расписания до тех пор, пока все работы не будут размещены. Итерационные же алгоритмы строят приближения расписания и оптимизируют их.

В данной работе рассматриваются жадные алгоритмы с процедурой ограниченного перебора. Особенностью таких алгоритмов является баланс между двумя процессами построения расписания. Жадные стратегии строят расписание быстро, однако очень быстро могут зайти в тупик при построении расписания. В таком случае, если расписание строится си сильным отклонением от оптимального, процедура ограниченного перебора корректирует его.

2 Цели и задачи курсовой работы

Целью этой курсовой работы является разработка алгоритмов построения многопроцессорного расписания с дополнительными ограничениями на основе комбинации жадных алгоритмов.

Для достижения указанной цели требуется:

1. Провести обзор алгоритмов построения списочных расписаний с целью выявления жадных критериев и схем ограниченного перебора которые могут быть модифицированы для решения данной задачи.
2. Разработать алгоритмы.
3. Реализовать алгоритмы.
4. Провести исследование свойств алгоритма.

3 Постановка задачи

3.1 Дано

1. Ориентированный граф работ G без циклов, в котором дуги - зависимости по данным, а вершины - задания. Вершин n, дуг m

2. Вычислительная система, состоящая из p различных процессоров

3. Матрица C_{ij} длительности выполнения работ на процессорах, $i = 1 \dots n, j = 1 \dots p$. Каждая строка этой матрицы - длины выполнения n-й задачи на p процессорах.

4. Матрица D_{kl} передач данных между процессорами, $k = 1 \dots p, l = 1 \dots p, D_{kk} = 0$. D_{ij}-й элемент этой матрицы - время пеердачи данных между процессорами i и j.

3.2 Определение расписания

Расписание программы определено, если заданы:

1. Множества процессоров и работ

2. Привязка - всюду определенная на множестве работ функция, которая задает распределение работ по процессорам

3. Порядок - заданные ограничения на последовательность выполнения работ и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве работ, распределенных на один процессор, является отношением полного порядка.

3.2.1 Способы представления расписаний

Пусть дан следующий граф потока данных:

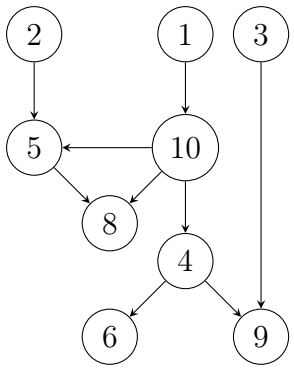

Рис. 1: Граф G потока данных

Пусть в оптимальном расписании работы 1, 10, 4, 6 будут поставлены на $Pr1$. 2, 5, 8 - на $Pr2$, а 3 и 9 - на $Pr3$. Рассмотрим как такое расписание будет выглядеть в различных формах:

1. Графическая форма представления

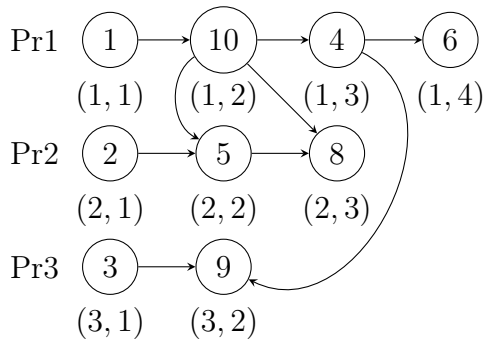

Рис. 2: Графическое представление расписания

В такой форме представления расписания каждой задаче сопоставляется пара из номера процессора и порядкового номера работы на процессоре, а так же секущие дуги, если задачи зависят друг от друга.

2. Временная диаграмма

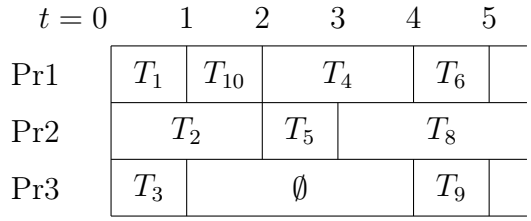

Рис. 3: Представление расписания в виде временной диаграммы

В такой форме представления расписания каждой задаче сопоставляется пара из номера процессора и временис старта задачи на процессоре.

Доказано, что эти формы полностью эквивалентны и, имея одну, возможно построить другую. В предложенном решении расписание строится в виде временной диаграммы.

3.3 Требуется

1. Построить расписание HP, то есть для i-й работы определить время начала ее выполнения s_i и процессор p_i на которм она будет выполняться
2. В расписании требуется минимизировать время выполнения набора работ, данных в графе G
3. В задаче так же присутствуют дополнительные ограничения, котрым расписание обязано удовлетворять.

Ограничения на корректность расписания следующие:

1. Каждый процессор может одновременно выполнять не больше одной работы;
2. Прерывание работ недопустимо, перенос частично выполненной работы на другой процессор недопустим;
3. Если между двумя работами есть зависимость на данным, то между завершением работы-отправителя и сиартом работы-получателя должен быть интервал времени, не меньший чем задержка на передачу данных между ними (с учетом привязки работ к процессорам и маршрута передачи данных).

3.4 Различные постановки задачи

1. Задача с однородными процессорами (длительность выполнения работы не зависит от того, на каком процессоре она выполняется) и дополнительными ограничениями на количество передач:

   - $CR = \frac{m_{ip}}{m} < 0.4$, где m_{ip} - количество передач данных между работами на каждый процессор. (Cut ratio)

2. Задача с неоднородными процессорами, но без дополнительных ограничений на расписание

4 Обзор предметной области

4.1 Критерии обзора

Ниже приведены критерии, по которым будут рассматриваться и сравниваться алгоритмы

1. Насколько сильно рассматриваемая в статье задача отличается от решаемой. Можно ли взять алгоритм, описанный в статье за базу для алгоритма для решения данной задачи.

2. Порядок сложности алгоритма.

3. На данных какой размерности протестирован алгоритм.

4.2 Конструктивные алгоритмы

4.2.1 Жадные алгоритмы

Жадные алгоритмы подразумевают декомпозицию задачи на ряд более простых подзадач. На каждом шаге решение принимается исходя из принципа получения оптимального решения для очередной подзадачи. То есть, на каждом шаге алгоритм делает выбор, оптимальный с точки зрения получения решения очередной подзадачи, предполагая, что эти локально-оптимальные решения приведут к приемлемому решению задачи. Какие-либо жадные стратегии, гарантированно получающие оптимальное расписание, на настоящий момент времени неизвестны, за исключением небольшого числа вариантов задач составления расписаний не принадлежащих к классу NP-полных. Например, известен жадный алгоритм, получающий точное решение для задачи обслуживания одним процессором максимального числа работ из заданного набора работ с фиксированными сроками начала и окончания [5]. Набор локальных критериев оптимизации сильно зависит от класса архитектуры. Для архитектур, в которых возможно последействие (распределяемый в расписание рабочий интервал оказывает влияние на времена инициализации ранее распределенных рабочих интервалов) возникает проблема выбора локальных критериев оптимизации, позволяющих учесть эффект последействия (на настоящий момент времени какие-либо обоснованные решения этой проблемы не известны). Кроме того, единого локального критерия (или набора и способа их использования), приводящего к наилучшему конечному результату, для решения всех подзадач не существует. Более того, при усложнении архитектуры набор и способ использования локальных критериев оказывает более сильное влияние на конечный результат. Таким образом, применение жадных алгоритмов для составления расписаний классом архитектур без последействия или даже без разделяемых ресурсов, если их влияние на значение функции построения временной диаграммы не может быть локализовано, а также проблемой выбора критериев оптимизации индивидуально для каждой подзадачи.

4.2.2 Структкра жадных алгоритмов построения многопроцессорного расписания.

При построении расписания жадным алгоритмом для каждой заадчи необходимо определить два параметра:

1. Привязку задачи к процессору p_i

2. Время старта задачи на процессоре s_i

Каждый из этих параметров может быть определен своим жадным критерием. Время старта (как показано в [12]) единственным образом определяется из порядка работ на процессоре. Следовательно, имеет роль порядок, в котором работы добавляются в расписание.

Таким образом, для построения расписания достаточно определить:

1. Привязку задачи к процессору p_i
2. Ее номер в очереди на добавление в расписание q_i

4.2.3 Другие конструктивные алгоритмы

(Написать про [1])

4.3 Итерационные алгоритмы

Итерационные алгоритмы работают путем создания аппроксимированного варианта решения и последующего его улучшения. Однако, большинство из них рандомизированные и, следовательно, из нескольких различных запусков теоретически возможно получить различные расписания (несмотря на то то они все сходятся). Более того, многие из таких алгоритмов плохо масштабируемы. Муравьиные алгоритмы разобраны в [14], имитации отжига - в [8].

Таблица 1: Существующие алгоритмы

Название алгоритма	Рандомизированность	Итерационный	Возможность масштабирования
Генетические алгоритмы	Рандомный	Итерационный	+/-
Алгоритм имитации отжига	Рандомный	Итерационный	+
Муравьиные алгоритмы	Рандомный	Итерационный	-
Жадные стратегии	Детерминированный	Конструктивный	+

Обзоры этих алгоритмов для схожих задач представлены в [3; 4; 11]

- Достоинства:
  1. Поскольку большинство из итерационных методов - рандомизированные алгоритмы, они меньше зависят от подбора параметров.
  2. Некоторые из итерационных алгоритмов, например, муравьиные алгоритмы, могут адаптироваться к изменению начальных условий, что не релевантно в рассматриваемой постановке задачи.
- Недостатки:
  1. Некоторые из итерационных алгоритмов могут быть плохо масштабируемы.
  2. Многие из итерационных алгоритмов требуют генерации начальных состояний расписаний, от которых они могут сильно зависеть.

5 Алгоритм построения расписания

5.1 Алгоритмы построения расписания

5.1.1 Общая схема жадных алгоритмов

Жадные алгортмы, представленные в данной работе, построены по следующей схеме:

1. Выбрать работу для постановки в расписание.
2. Выбрать процессор с и время начала выполнения задачи из п.1 на выбранном процессоре.
3. Поставить работу на процессор.
4. Остановиться, если все задачи поставлены, иначе перейти к п.1

5.1.2 Жадный алгоритм с жадным критерием

Жадный алгоритм с жадным критерием следует общей схеме, описанной в 5.1.1

Эту схему можно уточнить путем выбора критерия отбора в п.1 и критерия выбора процессора и начала времени выполнения в п.2. Для постановки задачи с дополнительными ограничениями, такого как CR, так же может быть неудача в постановки задачи в раписание, в случае невозможности постановки без нарушения дополнительного ограничения. В таком случае, алгоритм завершается с неудачей.

Задача d_i наывается **доступной для постановки** в расписание, в случае, если либо все ее предшественники уже установлены в расписание, либо все ее предшественники уже постановлены в расписание. Назовем множеством всех доступных для постановки задач $D = (d_0, d_1, \ldots, d_n)$.

Жадный алгоритм с жадным критерием выбирает задачу для постановки по следующему критерию:

1. Задача $d_i \in D$ доступна для постановки.
2. Пусть $Succ(d)$ - функция, определяющая количество непосредственных последователей работы в графе. Тогда $\forall d_j \in D, d_j \neq d_i : Succ(d_j) < Succ(d_i)$.

Для постановки с дополнительным ограничением CR жадный алгоритм с жадным критерием берет распределение работ по процессорам из специального алгоритма распределения (5.2.1), поэтому выбор процессора в п.2 всегда заранее детерминирован. Выбор начала выполнения работы в расписание производится в соответствии с алгоритмом постановки задачи на процессор (5.2.2).

Для постановки без дополнительных ограничений, жадный алгоритм с жадным критерием производит пробную постановку на каждый процессор и выбирает процессор с самым ранним временем завершения работы с учетом алгоритма постановки задачи на процессор (5.2.2).

5.1.3 Жадный алгоритм с EDF эвристикой

Данный алгоритм следует общей схеме, описанной в пункте 5.1.1, однако отличается от алгоритма, описанного в пункте 5.1.2 критерием выбора работы на постановку.

Эвристика “саммый ранний директивный срок первый” (earliest deadline first, или **EDF**) упорядочивает работы по возрастанию директивных сроков и выбирает работу с нименьшим директивным сроком на постановку. Однако, постановка задачи не предполагает у

задач директивных сроков, поэтому в данном алгоритме у каждой работы строятся фиктивные директивные сроки.

В случае, если существует директивный срок всего расписания d, то директивный срок d_A работы A может быть рассчитан следующим образом (при известном распределении работ на процессоры):

1. Найти длиннейший путь в графе потока управления от A до работы $S : Succ(S) = \emptyset$.
2. Рассчитать длину этого пути. Длина пути равна сумме всех передач задержек данных и времен выполнения работ. Задержки передач данных известны, так как известно распределение работ на процессоры. Пусть длина этого пути равна p.
3. $d_A \coloneqq d - p$

Видно, что работа A должна завершиться до d_A; иначе путь, найденный в п.1 завершится позже, чем d, даже если процессоры ни имеют никакой другой нагрузки, кроме этих работ.

Даже без изветсного директивного срока расписания, EDF эвристика все еще может быть использована для сортировки работ по уменьшению "потенциальной длины пути до конца расписания", учитывая, что расписание всегда завершится какой-либо работой $S : Succ(S) = \emptyset$. Также, нет необходимости вводить настоящие директивные сроки, в которые работы должны быть завершены. Таким образом, можно выставить директивный срок расписания в 0, и получить формальные директивные сроки по алгоритму, представленному выше. Такие директивные сроки могут быть отрицательными, что не препятствует сортировать работы по их возрвствнию.

Описанный алгоритм без модификаций применим к задаче с дополнительным ограничением CR, поскольку распределение работ на процессоры может быть рассчитано заранее, и поэтому, время задержек межпроцессорных передач известно заранее.

Однако, для постановки задачи без дополнительных ограничений привязка задач к процессорам заранее неизвестна, поэтому, для вычисления директивных сроков не учитываются задержки межпроцессорной передачи данных, а время выполнения данной задачи считается усредненным по всем процессорам. Например, если в системе три процессора, на которых задача выполняется 1, 1 и 4 у.е., то для рассчета директивного срока время выполнения данной задачи считается $(1+1+4)/3 = 2$. Такая аппроксимация не нарушает работу алгоритма, поскольку директивные сроки требуется только для сортировки работ.

Жадный алгоритм с EDF эвристикой начинается с вычисления фиктивных директивных сроков, после чего выполняется цикл, описанный в 5.1.1.

Еще не добавленная работа с минимальным фиктивным директивным сроком выбирается как очередной кандидат на добавление в расписание.

Аналогично алгоритму, описанному в 5.1.2, для задачи с дополнительным ограничением CR для выбора процессора для постановки очередной задачи используется распределение, построенное алгоритмом распределения задач на процессоры (5.2.1), а для постановки без дополнительных ограничений производит пробную постановку на каждый процессор с самым ранним временем завершения работы с учетом алгоритма постановки задачи на процессор (5.2.2).

5.2 Вспомогательные алгоритмы

5.2.1 Алгоритм распределения задач на процессоры

В качестве алгоритма распределения задач на процессоры был выбран алгоритм разбиения графа на кластеры METIS [7].

Дшля построения распределения работ на процессоры запускается алгоритм кластеризации графа с количеством кластеров, равным количеству процессоров, после чего каждый кластер распределенных задач присваивается одному процессору.

Для задачи с дополнительным ограничением CR используется взвешенное распределение METIS, где каждой вершине придается вес, равный времени выполнения задачи на процессоре. Поскольку в этой постановке проуессоры равный, конкретный процессор в которого берется время выполнения не имеет значения.

Разбиение, лучшее по балансу кластеров, может нарушать ограничение CR, в случае, если большие группы плотно взаимодействующих работ распределятся на разные процессоры. Эта проблема решается варьированием параметра `ufactor` алгоритма METIS, который контролирует отношение максимального количества работ в подграфе к вреднему количеству работ в подграфе. Другими словами, `ufactor` позволяет контролировать дизбаланс в количестве вершин в кластере. С увеличением этого параметра, CR понижается. Для генерации распределения, удовлетворяющего дополнительное ограничение CR, достаточно генерировать распределения с постепенным увеличением `ufactor` до тех пор, пока очередное распределение не выполнит CR.

5.2.2 Алгоритм постановки задачи на процессор

При постановке задачи на заданный процессор достаточно вычислить время начала выполнения задачи t такое, чтобы кждое частичное расписание после добавления оставалось корректным. Начальное время t выбирается как минимальное время, удовлетворяющее следующим условиям:

1. Все передачи данных от предшествующих задач завершились до t.

2. Существует свободный интревал времени (простой процессора или после завершения последней поставленной задачи), начинающийся в t и длительностью, больший или равный времени выполнения работы, в который не выолняется ни одназ работа. В некоторых случаях, задача будет поставлена до начала другой, не свзяанной с ней задачей, в случае, если времени простоя достаточно. В сложных графах потока управления, такие простои возникают в частичных расписаниях часто, а значит всегда есть смысл их заполнять.

6 Программная реализация алгоритма

Код реализации выложен в репозитории [9] В программной реализации были использованы следующие библиотеки:

1. `boost` 1.81 [2]
2. `json` 3.11.2 [6]
3. `toml11` 3.7.1 [10]

Проект обладает следующей структурой:

1. `greedy/greedy_algo.cpp`
2. `greedy/schedule.cpp`
3. `greedy/time_schedule.cpp`
4. `graph_part.cpp`
5. `input_class.cpp`
6. `json_dumper.cpp`
7. `logger_config.cpp`
8. `options.cpp`
9. `parser.cpp`

Для сборки проекта используется `CMake`.

```
mkdir build
cd build
cmake ..
make
```

Для сборки документации (на английском) используется `Doxygen`.

```
doxygen Doxyfile
```

7 Экспериментальное исследование алгоритма

8 Заключение

В ходе выполнения курсовой работы были достигнуты все ее цели, а именно:

1. Проведен обзор существующих решений задач. Произведено сравнений других стратегий с жадными критериями и ограниченным перебором. На основе обзора произведен выбор подхода, основанного на комбинации жадных стратегий и ограниченного перебора.

2. Разработан и реализован алгоритм.

3. Произведено исследование свойств алгоритма, которое показало что алгоритм генерирует расписание, превосходящее оптимальное на 8%. Более 90% задач размещены при помощи жадной стратегии.

Предложенный алгоритм, сочетающий жадные стратегии и ограниченный перебор, строит расписание, то есть каждой работе сопоставляет процессор и время старта работы.

При исследовании алгоритма были подобраны оптимальные параметры и определены направления дальнейшего улучшения и исследования алгоритма.

Список литературы

1. *Akbari M.*, *Rashidi H.* AN EFFICIENT ALGORITHM FOR COMPILE-TIME TASK SCHEDULING PROBLEM ON HETEROGENEOUS COMPUTING SYSTEMS. — 2015. — Янв. — DOI: 10.7813/2075-4124.2015/7-1/A.45.
2. Boost C++ libraries. — URL: https://www.boost.org/ (дата обр. 02.04.2023).
3. *Coffman E. G.* Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.
4. *Davis R. I.*, *Burns A.* A Survey of Hard Real-Time Scheduling for Multiprocessor Systems // ACM Computing Surveys. — 2011. — Окт. — Т. 43, № 4.
5. Introduction to Algorithms / T. H. Cormen [и др.]. — 2nd. — The MIT Press, 2001. — ISBN 0262032937.
6. JSON parsing library. — URL: https://github.com/nlohmann/json (дата обр. 02.04.2023).
7. *Karypis G.* METIS and ParMETIS // Encyclopedia of Parallel Computing / под ред. D. Padua. — Boston, MA : Springer US, 2011. — С. 1117—1124. — ISBN 978-0-387-09766-4. — DOI: 10.1007/978-0-387-09766-4_500. — URL: https://doi.org/10.1007/978-0-387-09766-4_500.
8. *Rzadca K. S. F.* Heterogeneous Multiprocessor Scheduling with Differential Evolution // IEEE Congress on Evolutionary Computation. — 2005. — Т. 3.
9. *Savitsky I.* multiprocessor-scheduling. — URL: https://github.com/ipsavitsky/multiprocessor-scheduling/tree/main/code (дата обр. 23.05.2022).
10. TOML parsing library. — URL: https://github.com/ToruNiina/toml11 (дата обр. 02.04.2023).
11. *К.В. Шахбазян Т. Т.* Обзор методов составления расписаний для многопроцессорных систем // Журнал советской математики. — 1981. — Т. 15, № 5. — С. 651—669.
12. *Калашников А. В.* Алгоритмы оптимизации расписаний, основанные на исправлении неоптимальных фрагментов. — 2004.
13. *Костенко В. А.* Алгоритмы комбинаторной оптимизации, сочетающие жадные стратегии и ограниченный перебор // Известия Российской академии наук. Теория и системы управления. — 2017. — № 2. — С. 48—56.
14. *Штовба С. Д.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — № 4. — С. 70—75.

ПРИЛОЖЕНИЕ 1

Классы входных данных

В данных, присланных от Хуавей существует разделение на 2 класса.

1. Первый класс (примеры DAG_A и DAG_B) характеризуется относительно небольшим масштабом графа работ, небольшим числом процессоров, полнотой графа связности процессоров и одинаковыми задержками между любыми двумя процессорами.

2. Второй класс (примеры DAG_C и DAG_D) характеризуется относительно большим масштабом графа работ, большим числом процессоров

Таблица 2: Сравнение примеров из классов данных

	Примеры входных данных			
Критерии	DAG_A	DAG_B	DAG_C	DAG_D
Масштаб графа работ	45 вершин; 75 ребер	1121 вершина; 6229 ребер	197494 вершин; 719389 ребер	1823309 вершин; 6172920 ребер
Разброс длительностей работ	1-10	1-10	все работы одной длины	все работы одной длины
Связность графа работ	1.66	5.55	3.64	3.83
Количество процессоров	2	10	256	4096
Полный граф связности процессоров	да	да	нет	да
Одинаковые задержки на передачу данных	да	да	да	нет