DOI:

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПЛАНИРОВАНИЯ ГРУПП ВЫЧИСЛИТЕЛЬНЫХ РАБОТ НА ОСНОВЕ МЕТОДА ВЕТВЕЙ И ГРАНИЦ

Гончар Д.Р.

Вычислительный центр им. А.А. Дородницына ФИЦ «Информатика и управление» РАН, Россия, г. Москва, ул. Вавилова, д. 40

dGonchar@ccas.ru

Аннотация: Рассматривается минимаксная задача построения расписания наименьшей длины без прерываний для многопроцессорной системы. Для решения данной задачи предложен параллельный алгоритм на основе метода ветвей и границ. Исследовано реальное ускорение расчётов при увеличении числа используемых вычислительных ядер.

Ключевые слова: многопроцессорная система, работы без прерываний, расписание наименьшей длины.

Введение

Задачи по построению оптимальных расписаний часто встречаются при планировании производственной деятельности, управлении энергетическими объектами, подготовке и проведении испытаний сложных технических систем (к примеру, в авиации), работе различных систем экологического, медицинского, промышленного и других видов мониторинга и в ряде других случаях. Различные обстоятельства применения нередко обусловливают и весьма разные требования к точности и скорости получения искомого решения.

Одним из очевидных способов ускорения расчётов при решении подобных задач является разработка параллельных разновидностей используемых алгоритмов планирования. Для алгоритмов на основе метода ветвей и границ этот вопрос весьма актуален в связи с достаточно высокой вычислительной сложностью метода (при том, что он позволяет получать точное решение).

В данной работе приведено как описание самого алгоритма, так и итоги расчётов с использованием разного числа процессоров (точнее, вычислительных ядер) на вычислительном комплексе Межведомственного суперкомпьютерного центра РАН.

1 Постановка задачи

Пусть имеется множество работ $N = \{1, 2, \ldots, n\}$, которое необходимо выполнить с помощью m процессоров, составляющих вычислительную систему для их обработки. Длительность выполнения работы i на процессоре j равно t_{ij} ($i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$). В каждый момент времени каждая работа может выполняться не более чем одним процессором, а каждый процессор может выполнять не более одной работы. Переключения с одного процессора на другой и прерывания при выполнении работ не допускаются.

Расписание выполнения работ N определим как разбиение множества N на m непересекающихся подмножеств $N_1, N_2, \ldots, N_m$ ($N = \bigcup_{j=1}^{m} N_j$; $N_{j_1} \bigcap N_{j_2} = \varnothing$ при $j_1 \neq j_2$). Работы из множества N_j приписываются процессору j и выполняются на нем одна за другой в произвольном порядке. Под загруженностью процессора j ($j = 1, 2, \ldots, m$) будем понимать величину $Q_j = \sum_{i \in N_j} t_{ij}$, а $\max_{j=1, 2, \ldots, m} Q_j$ – это длина расписания. Задача заключается в построении оптимального по быстродействию расписания, т.е. расписания минимальной длины.

Подобные задачи широко освещены в литературе. При их решении применяются, например, такие методы, как случайный и исчерпывающий поиск, методы математического программирования [1], метод ветвей и границ [2, 7], муравьиные алгоритмы, поиск с запретами, вероятностные алгоритмы, генетические алгоритмы [3], метод имитации отжига, различные эвристические алгоритмы [4, 5], алгоритмы агрегирования и др.

2 Метод ветвей и границ

Для решения вышеприведенной задачи предлагается метод ветвей и границ, основанный на результатах работы [2]. Этот метод подразумевает структуру поиска оптимального решения в виде дерева (что выполняется для нашей задачи), последовательное разбиение исходного множества решений на подмножества (ветви дерева решений) и применение оценочных процедур для

определения перспективности подробного исследования очередной ветви дерева решений. На каждом последующем шаге новые подмножества образуются в результате разбиения некоторых подмножеств, полученных на предыдущих шагах, пока для подмножеств, соответствующих концевым вершинам дерева, решение задачи уже не требует разбиения.

В итоге указанного разбиения мы получаем множество подзадач, которые могут обрабатываться независимо (совмещено, одновременно по времени).

2.1 Дерево решений

Опишем множество всех расписаний (их число равно m^n) в виде дерева решений (см. рис. 1). Корень дерева находится на нулевом уровне. Корень соответствует множеству всех расписаний. На первом уровне находится m вершин, каждая из которых соответствует множеству всех расписаний, в которых первая работа назначена на определенный процессор. На втором уровне дерева находится m^2 вершин, каждая из которых соответствует множеству всех расписаний, в которых первые две работы назначены на один или два определенных процессора. На n-м уровне дерева расписаний находится m^n листьев, каждый из которых соответствует некоторому расписанию выполнения множества работ N.

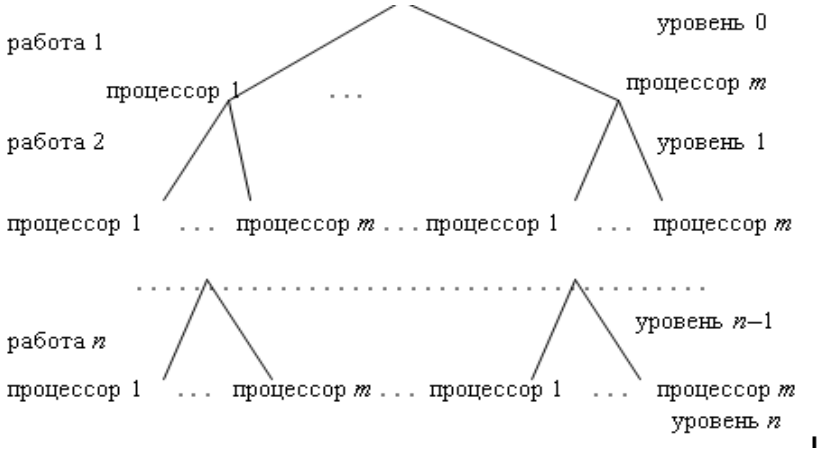

Рис. 1. Дерево решений, соответствующее множеству всех расписаний

Пусть x_k – некоторый узел уровня k дерева расписаний, $R(x_k)$ – множество всех расписаний, соответствующих этому узлу (т.е. множество расписаний, в которых работы 1, 2, …, k назначены на определенные процессоры), x_{k+1}^j – узел уровня $k+1$ ($k < n$), связанный с узлом x_k ребром, соответствующим процессору j. Наша цель – вычисление нижней и верхней оценок минимальной длины расписания на множестве $R(x_k)$. Имея эти оценки, можно применить стандартную схему метода ветвей и границ [7] (например, одностороннего или фронтального ветвления).

2.2 Вычисление нижней оценки

Пусть T_j ($j = 1, \ldots, m$) – загруженность процессора j после назначения первых k работ (т.е. T_j – это суммарная длительность работ из числа 1, 2, …, k, назначенных на процессор j). Нижнюю оценку $L(x_k)$ минимальной длины расписания на множестве $R(x_k)$ будем вычислять следующим образом:

$$L(x_k) = max(L_1(x_k), L_2(x_k), L_3(x_k)),$$

где $L_1(x_k)$, $L_2(x_k)$, $L_3(x_k)$ – это нижние оценки, вычисленные тремя различными способами.

Величина $L_1(x_k)$ вычисляется как следующий максимум: $L_1(x_k) = \max\limits_{j=1,2,\ldots,m} T_j$. При хранении величины $T_1, T_2, \ldots, T_m$ в виде обычного массива сложность вычисления $L_1(x_k)$ составляет $\theta(m)$.

Величина $L_2(x_k)$ вычисляется как следующий максимум:

$$L_2(x_k) = \max_{i=k+1,\ldots,n} \min_{j=1,\ldots,m} (T_j + t_{ij})$$

При использовании для этого двумерного массива A с элементами $a_{ij} = T_j + t_{ij}$, $i = k+1, \ldots, n$; $j = 1, 2, \ldots, m$ сложность вычисления величины $L_2(x_k)$ составляет $\theta(mn)$.

Величина $L_3(x_k)$ вычисляется по формуле.

$$L_3(x_k) = \frac{1}{m}\left(\sum_{j=1}^{m} T_j + \sum_{i=k+1}^{n} \min_{j=1,\ldots,m} t_{ij}\right)$$

Величину $\min_{j=1,\ldots,m} t_{ij}$ вычислим для всех $i = 1, 2, \ldots, n$ сразу до начала вычисления нижних оценок. Тогда сложность вычисления величины $L_3(x_k)$ составляет $O(n + m)$. Перейдем в дереве расписаний от узла x_k к узлу $x_{k+1}^{j_0}$, $k < n$ (т.е. будем считать, что работа $k+1$ назначена на процессор j_0).

Тогда

$$L_3(x_{k+1}^{j_0}) = \frac{1}{m}\left(\sum_{j=1}^{m} T_j + t_{k+1,j_0} + \sum_{i=k+1}^{n} \min_{j=1,\ldots,m} t_{ij}\right)$$

Вычислим разность

$$L_3(x_{k+1}^{j_0}) - L_3(x_k) = \frac{1}{m}\left(t_{k+1,j_0} - \min_{j=1,\ldots,m} t_{k+1,j}\right)$$

Таким образом,

$$L_3(x_{k+1}^{j_0}) = L_3(x_k) + \frac{1}{m}\left(t_{k+1,j_0} - \min_{j=1,\ldots,m} t_{k+1,j}\right)$$

и с помощью данного рекуррентного соотношения, используя $L_3(x_k)$, величина $L_3(x_{k+1}^{j_0})$ вычисляется за время $O(1)$.

2.3 Вычисление верхней оценки

В качестве верхней оценки $H(x_k)$ минимальной длины расписания на множестве $R(x_k)$ возьмем длину расписания, в котором работы 1, 2, …, k в соответствии с вершиной x_k дерева расписаний распределены на процессоры, а работы k+1, …, n распределяются по следующему “жадному” алгоритму. Пусть уже распределены работы 1, 2, …, p ($k \le p < n$), T_j – загруженность процессора j (j = 1, 2, …, m) и $\min(T_1 + t_{p+1,1}, ..., T_m + t_{p+1,m}) = T_{j_0} + t_{p+1,j_0}$. Тогда работа p+1 назначается на процессор j_0.

Указанные действия повторяются для $p = k, k + 1, \ldots, n - 1$. Сложность процедуры вычисления величины $H(x_k)$ составляет $O(mn)$.

2.4 Ветвление

При применении метода ветвей и границ происходит последовательное разбиение множества допустимых решений на подмножества: на каждом последующем шаге новые подмножества образуются в итоге разбиения некоторых подмножеств, полученных на предыдущих шагах. Так строится уже упоминавшееся выше дерево решения исходной задачи. Такое разбиение продолжается до тех пор, пока для подмножеств, соответствующих концевым вершинам дерева, решение задачи уже не требует разбиения.

В итоге разбиения начальная задача распадается на ряд подзадач, которые могут решаться в значительной степени независимо друг от друга. Необходимость поддерживать определенные связи (зависимости) между полученными подзадачами объясняется следующими двумя причинами:

дерево решения может оказаться плохо уравновешенным, что приводит к тому, что некоторая часть процессоров вычислительной системы оказывается неравномерно загруженными;

возникающие при попытке уравновешивания нагрузки зависимости по данным между подзадачами, связанные с передачей оценок, наилучших значений оптимизируемого функционала и других подобных сведений, могут приводить к большим накладным расходам на взаимодействие процессов, препятствующих повышению параллельной эффективности;

Для преодоления перечисленных причин снижения успешности распараллеливания решения задачи применяются методы оптимизации загрузки процессов, минимизации обменов данными, а также распределения обменов по вычислительному пространству [7, 8].

3 Порядок обхода дерева и распараллеливания в конкретной реализации алгоритма и исследование его масштабируемости в вычислительных опытах

При реализации метода ветвей и границ на однопроцессорной машине обход дерева выполняется последовательно, от одного узла к другому. В зависимости от порядка выбора узлов при этом различают обход в глубину и в ширину [9]. Если в каждом поддереве дерева решений сначала проходится корень, а потом его поддеревья, то это называется обходом в глубину. А если вначале проходится корень верхнего уровня, затем все вершины второго уровня и т.д., так, что никакая вершина некоторого уровня не может быть пройдена, пока не будут обследованы все вершины предыдущего уровня, то такой обход называется, соответственно, обходом в ширину (или фронтальным).

Если дерево имеет сравнительно небольшое число узлов, то обход в глубину удобно описывается и воплощается с помощью рекурсивной процедуры. Однако для большого числа узлов в дереве этот подход не применим, рекурсия не используется, а для упорядочивания обхода применяется стек, а узлы дерева воспроизводятся по мере их извлечения из стека. Для реализации обхода в ширину подобным образом используется очередь. Также, для определённости, обычно полагается, что потомки каждой вершины упорядочены некоторым образом, к примеру, при графическом изображении – слева направо.

Использование вычислительной системы со многими процессорами даёт возможность совместить по времени обход разных ветвей дерева. Основная задача, которая решается при таком совмещении вычислений, состоит в таком динамическом распределении неявно заданного дерева между процессорами (точнее, вычислительными ядрами), при котором достигается наименьшее значение времени обхода или наибольшее значение коэффициента использования системы (т.е. более полная загрузка выделенных для решения задачи процессоров).

В литературе неоднократно рассматривалась задача параллельного обхода дерева, в частности, в работе [6] показывается, что для эффективного использования ресурсов вычислительной системы необходимо по возможности снизить объём и частоту обмена данными между процессорами системы, а также обеспечить равномерную загрузку последних. Там же предложено два метода параллельного обхода: метод назначаемых поддеревьев и метод выделяемых поддеревьев, причём в первом методе основное внимание уделяется сокращению взаимного обмена сообщениями между процессорами, а во втором – организации по возможности более равномерной загрузки процессоров (вычислительных ядер) вычислительной системы.

Оба подхода основаны на вычислительной модели вида «Управляющий-рабочие», в которой управляющие задачи выполняет один выделенный процесс – «Управляющий», а все иные процессы выполняют рабочие вычисления. *Управляющий процесс* определяет, какое поддерево поиска выделить каждому рабочему процессу, а *рабочие процессы* выполняют обход полученных от управляющего процесса поддеревьев. При этом предлагается две стратегии параллельного обхода дерева.

В методе *назначаемых поддеревьев* управляющий процесс сначала выполняет построение и обход части дерева, начиная с его корня, в ширину, до тех пор, пока число вершин в очереди не превысит число предоставленных для параллельной работы алгоритму физических процессоров. Обосновывается, что данное превышение целесообразно сделать достаточно большим, поскольку это даёт возможность более равномерно загружать процессоры, если дерево решений оказалось таким, что обход разных ветвей дерева решений существенно различается по трудоёмкости вычислений. На выбор конкретного числа, разумеется, влияет размерность задачи и число предоставленных для вычислений процессоров. На следующем шаге вершины из очереди последовательно посылаются управляющим процессом через связующую среду рабочим процессам по мере осуществления теми обходов соответствующих поддеревьев, которые могут отличаться размерами.

Каждый рабочий процесс осуществляет обход в глубину назначенного ему поддерева. Заметим, что в тот временной промежуток, когда один процесс может выполнить обход нескольких поддеревьев, другой, возможно, успеет обработать только одно поддерево. Если очередь пуста, то завершивший обход очередного поддерева процесс останавливается (остальные процессы в это время ещё, возможно, продолжают работу). Таким образом, рабочие процессы в этом алгоритме обмениваются данными с главным процессом только при завершении обхода одного дерева и назначении другого (если хотя бы одно из них ещё не было передано в обработку) и не происходит никаких пересылок сообщений в ходе обхода назначенного поддерева кроме решения задач оптимального поиска (обнаружения нового рекорда и сообщения об этом управляющему процессу). Однако возможен случай, когда некоторые процессы получат деревья, во много раз превышающие другие по размеру и в то время, когда они будут продолжать их обход, иные деревья будут пройдены, и будет иметь место простой части процессоров, ухудшится такой показатель, как загруженность процессоров. Эффективность данного алгоритма

зависит от определяющих вид дерева решений исходных данных. Как один из приёмов повышения эффективности в данном подходе можно предложить увеличение начальной степени дробления поддеревьев в надежде на то, что самые большие из них будут разбиты на несколько подмножеств, но это же предложение приведёт и к возрастанию числа обращений к главному процессу, из-за чего рабочие процессы начнут простаивать в очереди, ожидая, пока управляющий процесс освободится от обработки ранее поступивших к нему запросов от других рабочих процессов.

Метод выделяемых поддеревьев подразумевает, что разбиение на задачи происходит по мере выполнения обхода. На первом шаге управляющий процесс сначала производит обход дерева в ширину, начиная от его корня, при этом достаточно, чтобы число концевых вершин было не меньше, чем число рабочих процессов (не требуется порождать концевых вершин существенно большее, чем число рабочих процессов). На втором шаге корневые вершины распределяются между рабочими процессами и каждый из них выполняет обход в глубину соответствующего поддерева. По завершении обхода своего поддерева рабочий процесс освобождается и производит запрос к управляющему процессу, который, в свою очередь, переадресует запрос другому рабочему процессу на выделение одного из поддеревьев обходимого им в данный момент дерева для обработки освободившемуся рабочему процессу. При выборе поддерева для передачи имеет смысл проверять некоторые условия разбиения дерева, показывающие целесообразность выделения в нём поддерева. Например, наибольший номер полностью пройденного уровня должен быть много меньше числа всех уровней в дереве. Из этих же соображений вводится понятие неделимого дерева (т.е. такого, для которого перестало выполняться условие разбиения). Как только дерево некоторого процесса становится неделимым, он посылает соответствующее сообщение управляющему процессу и тот исключает данный рабочий процесс из числа кандидатов на дробление дерева, номера которых хранятся в круговой очереди. В том случае, если среди незавершённых процессов остались лишь неделимые деревья, то освободившийся процесс полностью завершается. Алгоритм заканчивает свою работу при завершении обхода всех поддеревьев.

Метод выделяемых поддеревьев позволяет достичь более равномерной загрузки рабочих процессов, чем метод назначаемых поддеревьев, и не требует длительной предварительной процедуры порождения большого числа концевых вершин управляющим процессом, которая обычно выполняется последовательно. Недостатком метода выделяемых поддеревьев является необходимость производить большее по сравнению с методом назначаемых поддеревьев число взаимодействий с рабочими процессами.

Для реализации метода ветвей и границ в данной работе автором был выбран вариант метода назначаемых деревьев. Другие схемы и подробности их осуществления для рассмотренной в статье задачи предполагается опробовать и подробнее исследовать в дальнейшем.

В практическом плане нам интересен также вопрос, насколько в действительности ускоряются расчёты по решению задачи при её распараллеливании на предоставленном числе вычислительных ядер. Итоги ряда расчётов, выполненных при изучении этого вопроса, представлены в таблицах 1 и 2.

Таблица 1. Длительность выполнения программы при m = 2 и различной степени параллельности

n (число заданий)	Число вычислит. ядер	Время счёта	Ускорение
20	1	0,118	
20	8	0,024	4,92
20	16	0,031	3,81
24	1	0,807	
24	8	0,252	3,20
24	16	0,161	5,01
32	1	146,818	
32	8	20,275	7,24
32	16	10,751	13,66
34	1	520,57	
34	8	70,949	7,34
34	16	37,21	13,99
36	1	1890,78	
36	8	303,247	6,24
36	16	135,59	13,94

n (число заданий)	Число вычислит. ядер	Время счёта	Ускорение
36	32	81,82	23,11
36	64	39,99	47,28
38	1	8378,24	
38	8	943,29	8,88
38	16	586,39	14,29
38	32	305,72	27,40
38	64	175,27	47,80
38	96	158,74	52,78

Отметим, что на время выполнения программы на МСЦ РАН достаточно заметное влияние оказывает общая нагрузка на систему, поэтому становятся возможными, на первый взгляд, такие парадоксальные значения, как ускорение счёта более чем в 8 раз при распараллеливании на 8 вычислительных ядрах.

В Таблице 2 приведены итоги подобных расчётов при моделировании для $m = 3$.

Таблица 2. Длительность выполнения программы при m = 3 и различной степени параллельности

n (число заданий)	Число вычислит. ядер	Время счёта	Ускорение
16	1	0,86	
16	9	0,15	5,73
16	27	0,0986	8,72
16	81	0,186	4,62
20	1	52,09	
20	9	6,28	8,29
20	27	1,97	26,44
20	81	2,55	20,43
22	1	409,68	
22	9	47,42	8,64
22	27	16,95	24,17
22	81	6,075	67,44
23	1	1177,46	
23	9	135,27	8,70
23	27	47,34	24,87
23	81	17,24	68,30
24	1	3392,75	
24	9	388,4	8,74
24	27	139,36	24,35
24	81	47,029	72,14
25	27	382,09	
25	81	132,76	2,88
26	27	1082,32	
26	81	376,16	2,88
27	27	3025,09	
27	81	1080,53	2,80

Из таблицы видно, что при малых размерностях задачи заметное прибавление числа физических вычислительных ядер и увеличение степени параллельности решения задачи не ускоряют, а даже несколько замедляют её решение, что, видимо, объясняется большими накладными расходами при запуске заметно большего числа параллельных копий алгоритма.

Литература

1. *Алексеев О.Г.* Комплексное применение методов дискретной оптимизации. – М.: Наука, 1987.
2. *Фуругян М.Г.* Некоторые алгоритмы решения минимаксной задачи составления многопроцессорного расписания. // Изв. РАН, ТиСУ. 2014, № 2. – С. 50-56.

3. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов // Программирование. 2000. № 5. – С. 63-72.
4. *Brucker P.* Scheduling Algorithms. – Heidelberg, Springer, 2001.
5. *Гончар Д.Р.* Параллельная реализация мультиоценочного алгоритма составления многопроцессорного расписания без прерываний. // Некоторые алгоритмы планирования вычислений и методы многокритериальной оптимизации для многопроцессорных систем. – М.: ВЦ РАН, 2014. С. 21-31.
6. *Тимошевская Н.Е.* Параллельные методы обхода дерева // Математическое моделирование. 2004. 16(4), с. 105-114.
7. *Посыпкин М.А., Сигал И.Х., Галимьянова Н.Н.* Алгоритмы параллельных вычислений для решения некоторых классов задач дискретной оптимизации. – М.: ВЦ РАН, 2005.
8. *Посыпкин М.А., Сигал И.Х., Галимьянова Н.Н.* Параллельные алгоритмы в задачах дискретной оптимизации: вычислительные модели, библиотека, результаты экспериментов. – М.: ВЦ РАН, 2006.
9. *Кнут Д.* Искусство программирования для ЭВМ. Т. 1. М.: Мир, 1976. 734 с.