УДК 004

ПАРАЛЛЕЛЬНЫЙ ПОДХОД К РЕШЕНИЮ ЗАДАЧИ ОДНОМЕРНОЙ ПРОДОЛЖЕННОЙ УПАКОВКИ (1CBPP) С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ CUDA

© В. М. Картак*, А. В. Рипатти

Уфимский государственный авиационный технический университет
Россия, Республика Башкортостан, 450000 г. Уфа, ул. К. Маркса, 12.
E-mail: kvmail@mail.ru

Рассматривается задача одномерной продолженной упаковки (1CBPP). Для ее решения предлагается параллельный алгоритм, основанный на методе ветвей и границ. Алгоритм адаптирован к выполнению на графических видеокартах с поддержкой технологии CUDA.

Ключевые слова: *одномерная продолженная упаковка, метод ветвей и границ, параллельные вычисления, GPGPU, CUDA.*

Введение

Задача одномерной продолженной упаковки (1D Continuous Bin Packing Problem, 1CBPP) состоит определении минимального количества одномерных контейнеров длины W, в которые необходимо разместить предметы n типов заданной длины w_i, удовлетворяя потребности b_i ($1 \le i \le n$) с соблюдением последовательности загрузки контейнеров при их упаковке. Предметы каждого типа должны содержаться в смежных контейнерах (нет таких двух контейнеров, содержащих предмет данного типа, что имеется между ними третий контейнер без предмета данного типа). Последовательность контейнеров – фиксированная.

Эта задача исследовалась различными авторами [1]. В литературе она так же известна как задача одномерной прямоугольно ориентированной упаковки (1D Rectangular Oriented Cutting Stock Problem, 1ROCSP), которая используется в блочной технологии для решения задачи двухмерной упаковки в полосу (2D Strip Packing Problem, 2SPP) [2]. В работе [3] показано, что решение задачи 1CBPP является нижней границей решения задачи 2DPP. В ряде случаев из решения задачи 1CBPP удается получить оптимальное решение задачи 2DPP [4]. Задача 1CBPP также имеет применение в задачах теории расписаний, например, как задача расписания ресурсов [5].

Задача 1CBPP является NP-трудной. В качестве точных методов решения используются метод ветвей и границ, методы линейного программирования, а также их комбинации [1, 2, 4, 6].

В данной работе рассмотрен параллельный подход решения задачи 1CBPP. Распараллеливание вычислений является одним из способов ускорения решения различных задач. Однако только увеличение числа процессоров не приводит к увеличению производительности. Для эффективного распараллеливания нужно придумать способ разбиения задачи на информационно-независимые подзадачи, которые распределяются между процессорами так, чтобы нагрузка на каждый из них была примерно одинакова.

В качестве программно-аппаратных средств параллельных вычислений выбраны графические видеокарты фирмы Nvidia с поддержкой технологии CUDA. Выбор обусловлен высоким отношением производительности видеокарты к ее стоимости, а также наличием подробной документации.

Архитектура CUDA вносит дополнительные ограничения на алгоритм решения задачи. Например, согласно принципу SIMD (Single Instruction Multiple Data), все процессоры должны выполнять один и тот же алгоритм, а число ветвлений в алгоритме должно быть сведено к минимуму.

Статья имеет следующую структуру. В разделе 1 дана математическая модель задачи 1CBPP. Раздел 2 описывает способ сокращения пространства рассматриваемых решений, основанный на отображении множества допустимых решений на множество перестановок. В разделе 3 описан точный параллельный алгоритм решения задачи 1CBPP с учетом ограничений архитектуры CUDA. Вычислительные результаты реализации данного алгоритма представлены в разделе 4.

1. Математическая модель задачи 1CBPP

В данном разделе словесное описание задачи, приведенное во введении, записывается формально в виде математической модели. В конце раздела приведен пример.

Дано:

Четверка $< W, n, w, b >$, где

- W – целое число, $W > 0$.
- n – целое число, $n > 0$.
- w – вектор целых чисел $(w_1, w_2, \cdots, w_n)$, все $0 < w_i \le W$.
- b – вектор целых чисел $(b_1, b_2, \cdots, b_n)$, $b_i > 0$.

Требуется:

Найти решение – вектор $s = (s_1, s_2, \cdots, s_n)$ (все s_i – целые числа, $s_i \ge 0$) такой, что

$$T(s) = \max_{1 \le i \le n} (s_i + b_i) \to \min,$$

При условии:

$$\sum_{i \in K(t)} w_i \le W \text{ для } \forall t \in \{0, 1, \cdots, \max(s_i + b_i) - 1\},$$

где $K(x) = \{i : s_i \le x < s_i + b_i\}$.

* автор, ответственный за переписку

Вектор s описывает решение задачи 1CBPP. Присвоим последовательно каждому контейнеру номер начиная с нуля. Очевидно, для каждого типа предметов достаточно указать наименьший номер контейнера, в который был упакован предмет данного типа. Для i-го типа это число равно s_i.

Введем функцию $T(s)$, обозначающую количество контейнеров, которое требуется для упаковки всех предметов в решении s.

За OPT обозначим оптимальное решение задачи: $OPT = \arg\min_{s \in S} T(s)$, где S – множество всех решений. OPT всегда существует.

На *рис.* 1 показан пример решения задачи 1CBPP.

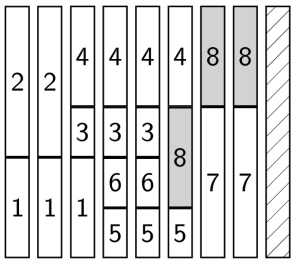

Рис. 1. Одно из оптимальных решений задачи при $W = 5$, $n = 8$, $w = (2,3,1,2,1,1,3,2)$, $b = (3,2,3,4,3,2,2,3)$: $OPT = (0,0,2,2,3,3,6,5)$, $T(OPT) = 8$.

2. Сокращение пространства решений

В разделе рассматривается способ сокращения числа рассматриваемых решений путем кодирования решений при помощи перестановок. Перестановка длины n – это последовательность из n различных целых чисел из диапазона [1... n]. Число перестановок длины n равно $n!$ – именно этим числом и будет ограничиваться количество решений для рассмотрения.

В конце раздела рассматривается кодирование перестановки целым числом, а также обратное преобразование. Этот прием позволяет сократить объем данных, которые нужно будет посылать на видеокарту.

Пусть S – множество всех решений, а P – множество всех перестановок чисел от 1 до n. Определим функцию $f : S \to P$ следующим образом. Упорядочим все n типов предметов по номеру контейнера, в котором находится самый левый предмет данного типа. При равенстве номеров контейнеров упорядочим типы по возрастанию номеров. Итоговой перестановкой будет последовательность номеров типов после упорядочивания.

$$f(\{s_1, s_2, \cdots, s_n\}) = \{p_1, p_2, \cdots, p_n\},$$

$$s_{p_i} < s_{p_j} \text{ или } s_{p_i} = s_{p_j}, p_i < p_j, \text{для} 1 \le i \le j \le n.$$

Определим функцию $g : P \to S$ с помощью правила Next Fit (NF). Его суть заключается в том, что группа предметов каждого типа укладывается, начиная с как можно более левого контейнера, в который может поместиться предмет данного типа, но не левее, чем первый предмет предыдущего типа:

$$g(\{p_1, p_2, \cdots, p_n\}) = \{s_1, s_2, \cdots, s_n\},$$

$$s_1 = 0,$$

$$s_i = \min\{x : x \ge s_{i-1}, \sum_{j \in K(x)} w_j \le W - w_i\}, \text{для } 2 \le i \le n,$$

где $K(x) = \{k : s_k \le x < s_k + b_k, 1 \le k < i\}$.

Отметим, что ни f, ни g не являются биекциями; $g(f(s)) = s$ лишь для некоторых $s \in S$, однако $f(g(p)) = p$ для любого $p \in P$.

Утверждение 1. $\forall s \in S : T(g(f(s))) \le T(s)$.

Доказательство. Пусть $s' = g(f(s))$.

Покажем, что $s'_i \le s_i$ для всех $1 \le i \le n$, методом математической индукции. Для $i = 1$ утверждение, очевидно, выполняется. Предположим, что для некоторого $2 \le k \le n$ все неравенства вида $s'_j \le s_j$ выполняются для любого $1 \le j < k$. Тогда, исходя из определения правила NF, $s'_k \le s_k$.

Поскольку $s'_i \le s_i$ для всех $1 \le i \le n$, то $T(s') \le T(s)$ по определению функции T. ■

Следствие. $\exists p \in P : T(g(p)) = T(OPT)$.

Таким образом, для решения задачи 1CBPP необходимо перебрать все перестановки длины n. Для каждой из них с помощью правила NF сгенерировать решение и выбрать наилучшее. Всего перестановок $n!$, правило NF можно реализовать за $O(n^2)$, тогда оценка сложности алгоритма сверху – $O(n!n^2)$.

Для передачи перестановок графической видеокарте для проверки удобно кодировать перестановки целыми числами.

Обозначим через F множество целых чисел от 0 до $n!-1$. Отсортируем все перестановки лексикографически ($p < p'$ если $\exists i : p_j = p'_j$ для $1 \le j < i$, $p_i < p'_i$) и занумеруем их последовательными числами от 0 до $n!-1$. Данное взаимно-однозначное соответствие дает нам две функции encode : $P \to F$ и decode : $F \to P$. Обе эти функции можно реализовать за $O(n^2)$.

3. Модификация метода ветвей и границ

В данном разделе описан параллельный алгоритм решения задачи 1CBPP. Этот алгоритм является модификацией метода ветвей и границ. Его мысленно можно разбить на две части: перебор на CPU и вычисления на GPU. Перебор на CPU последовательно обходит верхние слои дерева перебора только до определенной заранее заданной глубины. Для сокращения перебора используются различные отсечения. Самые глубокие достигнутые вершины сохраняются в очередь – каждая из этих вершин задает поддерево перебора заранее известного размера. Как только число вершин в очереди станет достаточно большим – все они засылаются на GPU, где обрабатываются параллельно.

Опишем этап перебора на CPU. Будем рекурсивно строить перестановку так, чтобы на d-м уровне глубины рекурсии был построен префикс перестановки длины d.

В процессе обхода будем строить частичные решения в соответствии с алгоритмом NF и отсекать ветви следующими правилами, описанными в [6]:

- Исключение вертикально эквивалентных решений. Если для текущей глубины рекурсии d $s_d = s_{d-1}$ и $p_d < p_{d-1}$, то завершаем обработку ветви, поскольку если поменять p_d и p_{d-1} местами, получим то же самое.
- Исключение улучшаемых решений. Если текущую группу предметов можно подвинуть левее хотя бы на один контейнер относительно предыдущей группы (тем самым получив решение не хуже), то завершаем обработку ветви.

Пусть мы рекурсивно спустились в дереве поиска на глубину $n-h$. Теперь мы находимся в корне дерева высотой h, у данного дерева имеется $h!$ листьев. Каждый лист соответствует перестановке, а номера этих перестановок образуют отрезок $[z \cdots z+h!-1]$, где $z = \text{encode}_{(p_t)}$, а p – перестановка, соответствующая самому левому листу в текущем поддереве. Если принять число h за константу, то данное множество перестановок можно закодировать одним числом – z. Именно это число мы и будем пересылать видеокарте. В качестве ответа будем принимать число $z' \in [z \cdots z+h!-1]$ – номер перестановки, на которой достигается минимум функции T.

В силу особенностей архитектуры CUDA, мы не можем сразу отправлять не проверку только что полученное число z. Вместо этого мы создадим очередь q, в которую будем помещать числа z каждого из достигнутых поддеревьев высоты h. Как только размер очереди станет равным k, где k – число потоков, отправим все числа из очереди на проверку. Каждый поток будет обрабатывать свою версию числа z, при этом все потоки будут действовать параллельно по одному и тому же алгоритму. А именно: генерировать все перестановки из заданного диапазона, строить для них решения и вычислять значение функции T. Такой подход хорошо ложится на архитектуру CUDA: мало ветвлений, все потоки делают одно и то же.

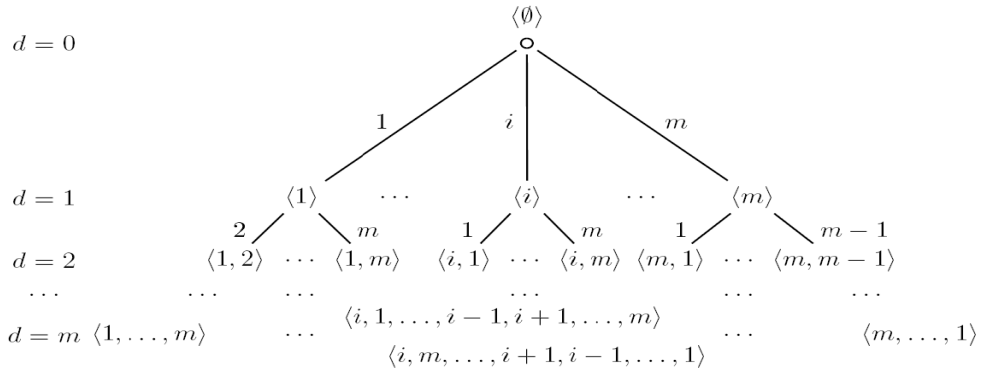

Рис. 2. Дерево перебора перестановок.

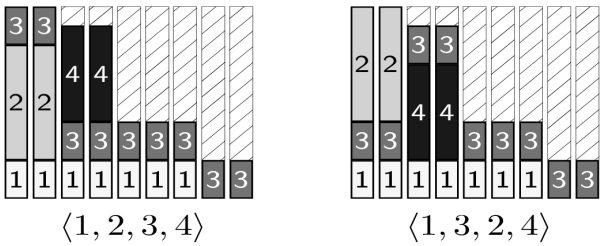

Рис. 3. Вертикально эквивалентные решения.

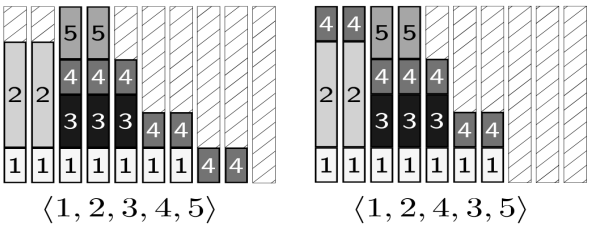

Рис. 4. Улучшаемые решения.

После окончания перебора в очереди q могут еще остаться числа – их также следует отправить на обработку видеокарте.

Теперь рассмотрим этап вычислений на GPU. Каждый поток на вход получает число z, после чего в цикле перебирает $h!$ перестановок и находит число $z' = \arg\min_{x\in[z\cdots z+h!-1]}\{T(g(\text{decode}(x)))\}$, которое затем и возвращает.

Таким образом, решение схематично выглядит так:

thread_kernel(z) // код для одного потока

1 $z' \leftarrow z$, $m \leftarrow +\infty$
2 цикл $i \leftarrow z \cdots z + h!-1$
3 $m' \leftarrow T(g(\text{decode}(i)))$
4 если $m' < m$, то
5 $z' \leftarrow i$, $m \leftarrow m'$
6 return z'

dfs_device(d) // рекурсивная генерация перестановки

1 если $d = n - h$, то
2 добавить в очередь q число z = encode (p)
3 если размер очереди q равен k, то
4 запустить thread_kernel для всей очереди q
5 определить и сохранить лучшее найденное решение
6 очистить очередь q
7 иначе
8 цикл – перебор значений p_d
9 если не сработали отсечения
10 dfs_device($d+1$)

main_device(z) // главная функция

1 dfs_device (1)
2 запустить thread_kernel для всей очереди q
3 определить и сохранить лучшее найденное решение
4 вывести лучшее найденное решение

4. Результаты вычислений

В эксперименте участвовали несколько программ:

- Cuda – решение, описанное в настоящей статье, использующее возможности CUDA
- Cpu – решение, использующее только мощности процессора; представляет собой модификацию программы Cuda, в которой значение h понижено до 1, а код работы с CUDA исключен за ненадобностью.
- LP – решение на основе методов линейного программирования для оценки ответа снизу [3].
- Gene – генетическое решение для оценки ответа сверху.

Тесты сгенерированы случайным образом. Каждая из величин задается случайным образом на соответствующих отрезках: $W \in [30,100]$, $w_i \in [1,W/2]$, $b_i \in [1,W/3]$.

Вычисления проводились на машине Core i7, 8Gb RAM, GeForce GTX 560. Результаты вычислительного эксперимента представлены в *табл.*

Таблица

Результаты вычислительного эксперимента

Тест	n	Время Cpu	Время CUDA	$T(s)$ LP	$T(s)$ exact	$T(s)$ gene
B1	10	0.2 с	0.4 с	70	72	72
B2	12	0.9 с	0.1 с	13	14	14
B3	13	3.5 с	0.3 с	15	16	16
B4	14	12.5 с	1 с	10	10	10
B5	15	6 мин	25 с	48	50	51
B6	15	16 ч	1.3 ч	131	132	133
B7	16	58 мин	4 мин	20	21	21
B8	17	11 ч	1 ч	52	53	54

Анализ времени выполнения говорит о том, что программа Cuda в среднем работает в 12 раз быстрее программы Cpu.

Выводы

Представленный в статье параллельный алгоритм хорошо адаптируется к архитектуре CUDA. Достигнуто 12-кратное уменьшение времени работы по сравнению с аналогичным решением, не использующим возможности CUDA.

Полученные результаты показывают перспективность параллельного подхода в отношении решения задачи 1CBPP.

Работа поддержана грантом РФФИ № 12-07-00631-а.

ЛИТЕРАТУРА

1. Martello S., Monachi M., Vigo D. An exact approach to the strip-packing problem // INFORMS Journal on Computing. 2003. №15(3). P. 310–319.
2. Месягутов М. А. Задача двумерной ортогональной упаковки: поиск нижней границы на базе решения одномерной продолженной упаковки // Информационные технологии. 2010. №6. С. 13–23.
3. Картак В. М., Месягутов М. А., Мухачева Э. А., Филиппова А. С. Локальный поиск ортогональных упаковок с использованием нижних границ // Автоматика и телемеханика. 2009. №6. С. 167–180.
4. Месягутов М. А. Методы принятия оптимальных решений на основе анализа эффективности значений функции цели в задачах прямоугольной упаковки: Автореф. дис. канд. физ.-мат. наук. Уфа, 2010. 19 с.
5. Hartmarm S. Project Scheduling under limited resources. Models, methods and applications. Berlin: Springer, 1999.
6. Мухачева Э. А., Мухачева А. С. Задача прямоугольной упаковки: методы локального поиска оптимума на базе блочных структур // Автоматика и телемеханика. 2004. №2. С. 10–15.

Поступила в редакцию 09.12.2012 г.