Параллельная реализация алгоритма вершинной минимизации недетерминированных конечных автоматов

А.В. Цыганов

Ульяновский государственный педагогический университет им. И.Н. Ульянова

Задача вершинной минимизации недетерминированных конечных автоматов в общем случае является PSPACE-полной задачей и может потребовать значительных вычислительных затрат даже для автоматов с небольшим числом состояний. В настоящей работе рассматривается параллельная реализация одного из точных методов вершинной минимизации НКА — алгоритма Камеды-Вейнера. Программная реализация алгоритма выполнена на языке C++ с использованием двух технологий параллельного программирования: OpenMP и MPI. Приводятся результаты численных экспериментов, демонстрирующие некоторые статистические свойства рассматриваемого алгоритма и подтверждающие эффективность предложенной реализации.

Введение

Конечные автоматы (КА) находят самое широкое применение в различных областях науки и техники, например, в системах обработки текста, машинного перевода, распознавания речи и др. На практике исследователю очень часто приходится выбирать между двумя типами автоматов: детерминированными конечными автоматами (ДКА) и недетерминированными конечными автоматами (НКА). Кроме того, в некоторых приложениях важную роль может играть способ представления КА в памяти компьютера. Например, может потребоваться минимизация КА по числу состояний (вершинная минимизация), по числу переходов (дуговая минимизация) или по другим критериям, в частности, смешанным. Первая из этих задач является, пожалуй, самой известной. В то время как для ДКА задача вершинной минимизации имеет сложность $O(n \log n)$, аналогичная задача для НКА является PSPACE-полной [1]. Это означает, что вершинная минимизация даже небольших автоматов может потребовать больших вычислительных затрат.

К настоящему времени разработано значительное количество точных и приближенных алгоритмов вершинной минимизации НКА. Одним из самых первых точных методов является алгоритм Камеды-Вейнера [2]. Несмотря на почтенный возраст алгоритма, в многочисленных программах для работы с конечными автоматами (по крайней мере, бесплатных) его реализация отсутствует.

В настоящей работе будет рассмотрена параллельная реализация алгоритма Камеды-Вейнера, выполненная автором на языке C++ с использованием двух технологий параллельного программирования: OpenMP и MPI.

1. Постановка задачи и описание алгоритма Камеды-Вейнера

Приведем основные определения из теории КА, которые нам понадобятся для формулировки алгоритма Камеды-Вейнера (более подробно с соответствующей теорией можно ознакомиться в многочисленных монографиях, например, [3]).

Детерминированным конечным автоматом называется пятерка $A = (Q, \Sigma, \delta, s, F)$, где Q — некоторое конечное множество состояний (вершин) автомата, Σ — рассматриваемый алфавит, δ — функция переходов вида $\delta : Q \times \Sigma \to Q$, $s \subset Q$ — стартовое состояние (вход) автомата, $F \subset Q$ — множество финальных состояний (выходов).

Недетерминированным конечным автоматом называется пятерка $A = (Q, \Sigma, \delta, S, F)$, где

Q — некоторое конечное множество состояний автомата, Σ — рассматриваемый алфавит, δ — функция переходов вида $\delta : Q \times \Sigma \to 2^Q$ (2^Q — множество всех подмножеств множества Q), $S \subset Q$ — множество стартовых состояний, $F \subset Q$ — множество финальных состояний.

Задача вершинной минимизации НКА состоит в построении такого НКА, который был бы эквивалентен исходному, то есть задавал бы тот же самый регулярный язык, что и исходный автомат, но имел бы при этом как можно меньшее число состояний.

Введем следующие обозначения (аналогичные обозначениям [2]). Пусть A — некоторый КА, тогда через $\bar{A}$ обозначим соответствующий ему зеркальный автомат (то есть автомат с противоположным направлением дуг, входы которого являются выходами автомата A, а выходы — входами). Если A — НКА, то через $D(A)$ будем обозначать ДКА, полученный из A процедурой детерминизации (subset construction). Для ДКА B, через $\hat{B}$ будем обозначать соответствующий минимальный автомат, то есть автомат, полученный из B объединением эквивалентных состояний.

Пусть A — некоторый НКА, $B = D(A)$, $C = D(\bar{A})$, а $\hat{B}$ и $\hat{C}$ — соответствующие минимальные автоматы. Состояниями автоматов B, C, $\hat{B}$ и $\hat{C}$ являются подмножества множества состояний исходного автомата A (заметим, что после выполнения процедуры детерминизации число состояний в автомате может экспоненциально возрасти). Матрицей RAM (Reduced Automaton Matrix) назовем матрицу, состоящую из нулей и единиц, число строк в которой совпадает с числом состояний автомата $\hat{B}$, а число столбцов — с числом состояний автомата $\hat{C}$. Пусть $p_1, p_2, \ldots, p_m$ — состояния автомата $\hat{B}$, а $q_1, q_2, \ldots, q_n$ — состояния автомата $\hat{C}$, тогда по определению элемент r_{ij} матрицы RAM равен 0, если $p_i \cap q_j = \emptyset$ и 1 в противном случае.

Некоторую пару подмножеств строк и столбцов матрицы RAM назовем гридом (блоком), если, во-первых, на всех их пересечениях стоят 1 и, во-вторых, это множество нельзя пополнить ни строкой, ни столбцом без нарушения первого свойства. Покрытием Z назовем такое множество гридов, для которого каждая единица матрицы RAM принадлежит, по крайней мере, одному из них. Размером покрытия будем называть число гридов в нем. Минимальным покрытием назовем покрытие, содержащее наименьшее возможное количество гридов.

В работе [2] описана процедура, называемая правилом пересечений (intersection rule), позволяющая по заданному покрытию Z и автомату $\hat{B}$ построить НКА $I(Z, \hat{B})$, который может оказаться эквивалентным исходному. При этом число вершин автомата I равно размеру покрытия. Доказано, что минимальные автоматы следует искать только среди таких автоматов.

Сформулируем теперь алгоритм Камеды-Вейнера вершинной минимизации НКА A.

Алгоритм Камеды-Вейнера.

1. Построить автоматы $B = D(A)$ и $C = D(\bar{A})$.
2. Для автоматов B и C построить минимальные автоматы $\hat{B}$, $\hat{C}$.
3. По автоматам $\hat{B}$ и $\hat{C}$ построить матрицу RAM.
4. Найти все гриды матрицы RAM.
5. Найти минимальное покрытие матрицы RAM. Пусть $i_{\min}$ — размер минимального покрытия. Положим $i = i_{\min}$.

   А. Для каждого покрытия Z размера i построить НКА $I(Z, \hat{B})$ и проверить его эквивалентность исходному автомату A.

   Б. Если эквивалентного НКА не найдено, то положить $i = i + 1$ и перейти к п. А.

Описанный алгоритм останавливается либо при нахождении минимального автомата, либо, если i становится равно числу вершин автомата A. В отличие от ДКА, минимальный автомат для заданного НКА может оказаться не единственным. Очевидно, что с помощью алгоритма Камеды-Вейнера могут быть найдены все минимальные автоматы.

2. Программная реализация

Программная реализация алгоритма вершинной минимизации НКА Камеды-Вейнера была выполнена в виде проекта на языке C++ с использованием технологий параллельного программирования OpenMP и MPI и кросс-платформенной библиотеки HeO. С подробным описанием библиотеки можно ознакомиться в работе [5] (официальная страница проекта расположена по адресу: http://code.google.com/p/heo/, в настоящее время для скачивания доступна версия 1.1 библиотеки). Для реализации алгоритма минимизации были написаны все необходимые классы, методы и вспомогательные функции, а для проведения численных экспериментов — процедуры случайной генерации НКА с заданными свойствами.

Самыми трудоемкими операциями алгоритма Камеды-Вейнера являются шаги 4 и 5: поиск гридов матрицы RAM и ее покрытий. Последняя задача является разновидностью задачи о покрытии множества (SCP — Set Covering Problem), которая в общем случае является NP-трудной, и в худшем случае алгоритмы ее решения по сложности не отличаются от переборных, при этом нахождение минимального покрытия матрицы RAM каким-либо из известных алгоритмов решения SCP не гарантирует построения автомата, эквивалентного исходному. Поэтому для выполнения шагов 4 и 5 алгоритма было решено использовать метод «грубой силы» — параллельный перебор. Для нахождения гридов и покрытий матрицы RAM в программе используется генератор сочетаний из n элементов по k элементов, основанный на алгоритме Twiddle [6].

Сформулируем алгоритм нахождения гридов, использующий приведенное выше определение грида.

Алгоритм нахождения гридов.

Для каждого $k = 1, 2, \ldots, n$, где n — число столбцов матрицы RAM:

1) сгенерировать все возможные подмножества столбцов матрицы RAM, состоящие из k элементов;
2) для каждого подмножества столбцов Y выбрать такое подмножество X строк матрицы RAM, чтобы на пересечении всех строк и столбцов были только единицы;
3) проверить, можно ли к данному множеству столбцов Y добавить еще столбцы без нарушения условия 2 и, если это невозможно, добавить грид $X \times Y$ в список найденных гридов.

Для уменьшения перебора описанный выше алгоритм поиска гридов «по столбцам» применяется, если число столбцов в матрице RAM меньше числа строк, в противном случае поиск гридов осуществляется аналогичным образом «по строкам». Генерация подмножеств (сочетаний) осуществляется каждым потоком программы независимо, начиная со своего номера, с шагом N, где N — общее число потоков. Найденные гриды помещаются в специальный вектор grids (нумеруются). В OpenMP-версии программы вектор grids является общим для всех потоков, а в MPI-версии он у каждого потока свой. По окончании поиска в MPI-версии потоки обмениваются найденными гридами.

После нахождения гридов производится поиск и анализ всех покрытий матрицы RAM размера $i = i_{\min}, i_{\min}+1, \ldots, i_{\max}$, где $i_{\min}$ и $i_{\max}$ вычисляются автоматически или задаются пользователем. Для каждого i всеми потоками генерируются подмножества вектора grids, состоящие из i элементов и для каждого такого подмножества Z проверяется, является ли оно покрытием матрицы RAM. Если Z является покрытием, то согласно правилу пересечений находится НКА $I(Z, \hat{B})$ и проверяется его эквивалентность исходному автомату, для чего сначала строится ДКА $D = D(I)$, а затем минимальный автомат $\hat{D}$, после чего проверяется изоморфизм автоматов $\hat{D}$ и $\hat{B}$. Если автоматы $\hat{D}$ и $\hat{B}$ изоморфны, то автомат $I(Z, \hat{B})$ является искомым минимальным автоматом и помещается в специальный вектор, который в OpenMP-версии программы является общим для всех потоков, а в MPI-версии у каждого потока свой. Поиск минимальных автоматов продолжается либо до первого найденного,

либо до нахождения всех минимальных автоматов. По окончании поиска в MPI-версии все потоки передают найденные минимальные автоматы нулевому потоку.

В алгоритмах работы с КА активно используются структуры данных, основанные на множествах. Для работы с множествами в реализованном проекте используется библиотека BitMagic [7]. Данная библиотека позволяет эффективно работать с большими и разреженными множествами, а также позволяет выполнять их сериализацию и десериализацию, что необходимо в MPI-версии программы. Кроме того, для MPI-версии программы были написаны процедуры сериализации и десериализации автоматов (для обмена найденными решениями).

3. Численные эксперименты

Статистические свойства алгоритма Камеды-Вейнера (точнее говоря, свойства объектов, возникающих в процессе его работы) для разных типов НКА изучены крайне мало и параллельная реализация данного алгоритма позволяет изучать их с большей эффективностью.

В качестве примера рассмотрим результаты минимизации 1000 НКА с одним входом и двумя выходами без недостижимых и бесполезных состояний (trim-автоматов), сгенерированных по алгоритму, аналогичному алгоритму Лэсли (см., например, [8]). Входными данными для этого алгоритма генерации автоматов являются: число состояний автомата, число начальных состояний, число конечных состояний, размер алфавита и плотность переходов $D = \frac{T}{|Q|^2|\Sigma|}$, где T — это число переходов в автомате. Ниже приведены результаты работы алгоритма для $|Q| = 5$, $|\Sigma| = 2$, $D = 0{,}2$. Эксперименты проводились на суперкомпьютерном кластере «Скиф-политех» («Т-Платформы» HPC–0011102–001) Томского политехнического университета. Результаты расчетов проверялись в программе GAP [9].

На рис. 1 представлено распределение размеров матрицы RAM для рассматриваемой выборки.

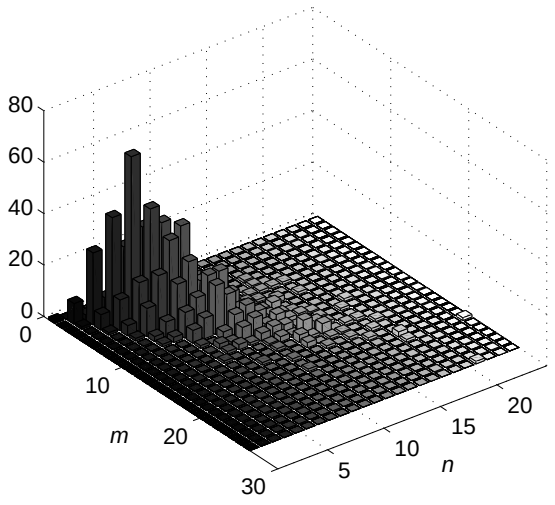

Рис. 1. Распределение размеров матрицы RAM

В таблице 1 приведены остальные результаты эксперимента. Обозначения: m, n — соответственно число строк и столбцов матрицы RAM; d — плотность единиц в матрице RAM; N_G — число гридов в матрице RAM; N_I — число найденных минимальных автоматов; $|Q_I|$ — число вершин в минимальном автомате; t — время вычислений в секундах для 64 потоков;

min — минимальное значение; max — максимальное значение; μ — среднее значение; σ — среднеквадратическое отклонение.

Таблица 1. Результаты эксперимента

	m	n	d	N_G	N_I	$\lvert Q_I \rvert$	t
min	1	1	0,3333	1	0	0	0,0017
max	26	24	1,0000	1329	6	4	36,9774
μ	7,2100	7,8330	0,6140	23,8160	0,7540	1,7020	0,1848
σ	3,0502	3,1405	0,0878	61,1850	1,0671	1,8699	1,5989

На рис. 2 приведена зависимость среднего времени вычислений в секундах (t) от числа потоков (N).

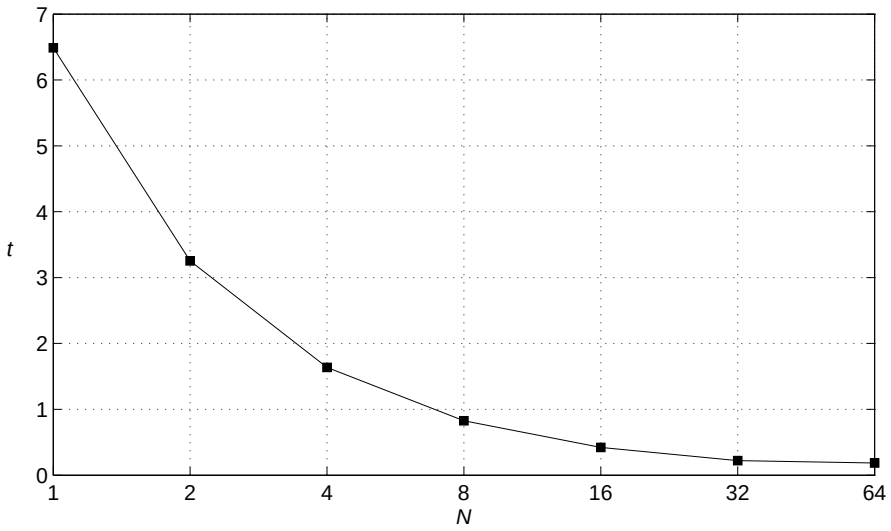

Рис. 2. Среднее время вычислений

Заключение

В работе описана параллельная реализация алгоритма вершинной минимизации НКА Камеды-Вейнера. Точное решение данной задачи даже для НКА с небольшим числом состояний может потребовать больших вычислительных затрат, поэтому актуальным остается вопрос разработки эффективных приближенных методов ее решения. В частности для нахождения минимального покрытия матрицы RAM могут использоваться метаэвристические алгоритмы, такие, например, как метод имитации отжига [10].

В настоящее время автором ведутся работы по реализации для подзадачи поиска минимального покрытия параллельной версии мультиэвристического алгоритма [11], основанного на использовании метода ветвей и границ в сочетании с комплексом эвристик для выбора разделяющего элемента (заметим, что данный алгоритм может быть применен для достаточно широкого круга задач дискретной оптимизации).

Автор выражает благодарность руководству и сотрудникам центра коллективного пользования «Суперкомпьютерный кластер» Томского политехнического университета и лично Дубакову Анатолию Алексеевичу и Белоцерковскому Александру Владимировичу.

Литература

1. Jiang T., Ravikumar B. Minimal NFA problems are hard // SIAM Journal on Computing, 22(6):1117–1141, December 1993.

2. Kameda T., Weiner P. On the state minimization of nondeterministic finite automata // IEEE Transactions on Computers. 1970. Vol. C-19, no. 7. P. 617–627.

3. Мельников Б. Ф. Недетерминированные конечные автоматы. Тольятти: Изд-во ТГУ, 2009. 160 с.

4. Gansner E. R., North S. C. An open graph visualization system and its applications to software engineering // Softw. Pract. Exper. 2000. V. 30, no. 11. P. 1203–1233.

5. Цыганов А. В., Булычов О. И. HeO: библиотека метаэвристик для задач дискретной оптимизации // Программные продукты и системы. 2009. №4. С. 148–151.

6. Chase P. J. Algorithm 382: Combinations of M out of N Objects [G6] // Communications of the Association for Computing Machinery 13:6:368, 1970.

7. Kuznetsov A., Shemanarev M., Tolstoy I., Lewis E., Khovayko O. BitMagic Library [Электронный ресурс]. URL: http://bmagic.sourceforge.net (дата обращения: 12.12.2010).

8. Almeida M., Moreira N., Reis R. On the performance of automata minimization algorithms: Tech. rep. / DCC-FC & LIACC, Universidade do Porto, 2007.

9. The GAP Group. GAP — Groups, Algorithms, and Programming, Version 4.4.12; 2008 [Электронный ресурс]. URL: http://www.gap-system.org (дата обращения: 5.01.2011).

10. Цыганов А. В. Имитационная нормализация в задаче минимизации недетерминированных конечных автоматов // Всероссийская конференция с элементами научной школы для молодежи «Проведение научных исследований в области обработки, хранения, передачи и защиты информации», 1–5 декабря 2009 г. Россия, г. Ульяновск : сборник научных трудов. В 4 т. Т. 2. Ульяновск: УлГТУ, 2009. С. 270–275.

11. Melnikov B. F., Tsyganov A. V., Bulychov O. I. A Multi-Heuristic Algorithmic Skeleton for Hard Combinatorial Optimization Problems // CSO'09: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization. Washington, DC, USA: IEEE Computer Society, 2009. P. 33–36.