Обхождане на граф в дълбочина

Проект по Системи за паралелна обработка

8 юни 2019 г.

Изготвил: Никола Стоянов
ф.н. 81151, КН, минали години

Проверил: .....
/ас. Христо Христов/

1 Цел на проекта

Задачата е да се реализира паралелен алгоритъм за обхождане на граф в дълбочина (Depth First Search). Графът може да бъде ориентиран или неориентиран, свързан или слабо свързан. Целта е да се постои дърво на обхождането.

Програмата трябва да отговаря на следните изисквания:

- позволява въвеждане на граф от входен файл или генериране на произволен граф по подадени брой върхове и ребра
- позволява извеждане на резултата от обхождането в избран изходен файл
- позволява задаване на максималния брой нишки, които се използват за решаване на задачата

2 Реализация

Проектът е реализиран като приложение за командния ред. Поддържа се следния интерфейс

```
./parallel_dfs gen -n 20 -m 40 -t 4 --algo par_mat
```

- `gen` - режим на работа. Единственият имплементиран е gen - генерирай случаен граф и направи обхождане по него.
- `-n 20` - брой върхове
- `-m 40` - брой ребра
- `-t 4` - колко нишки да използва
- `--algo par_mat` - кои алгоритми и стуктури от данни да използва. Вариантите са `seq_list`, `par_list`, `cheat_list`, `seq_mat`, `par_mat`, `cheat_mat`

Всички аргументи могат да се видят с `./parallel_dfs --help`

Следните изисквания към проекта не са имплементирани:

- въвеждане на граф от входен файл
- извеждане на резултата в изходен файл (има опция за извеждане на резултата на стандартния изход в "debug"формат).

Използван е езикът Rust. Използвани са следните външни библиотеки:

- `crossbeam` - lock free структури от данни
- `spin` - имплементация на Spin Lock
- `structopt` - обработка на аргументи от командния ред
- `rand` - генератори на случайни числа
- `rayon` - библиотека за паралелна обработка на данни

3 Генериране на графа

Решението на задачата може да се раздели на две независими части - генерирането на графа и обхождането му. Реализирани са два варианта за записване на графа - като списъци на съседство и като матрица на съседство.

3.1 Списъци на съседство

Списъците на съседство са в общия случай по-бързи за генериране и обработка и заемат по-малко памет от матрицата, но не позволяват много добро разпаряляване при генерирането на неориентиран граф.

Ориентиран граф За генерирането на ориентиран граф се използва следната стратегия. Графа се разделя на множество части, всяка с фиксиран размер от k върха (k = 128) и $m * \frac{n}{k}$ ребра. За всяка част се пуска задача в глобален thread pool, като часта с върхове $a..b$ генерира ребра започващи от нея, т.е. ребра *(c, v)*, $a <= c <= b$. Така всяка нишка работи върху отделна част от графа.

Неориентиран граф Генерирането на неориентиран граф е по-сложна задача, защото за всяко добавено ребро *(u, v)*, трябва да се добави и обратното ребро *(v, u)*. Първата стъпка е да се генерират ребрата *(u, v)* така, че $u > v$. Използва се същия подход като при ориентирания граф, с разликата, че броят ребра, които се генерират за часта с върхове от a до b е $\frac{m*(b^2-a^2)}{n^2}$.

След това трябва да се добавят огледалните ребра. Идеята е да обходим вече генерираните ребра, и за всяко ребро *(a, b)* да добавим реброто *(b, a)*. Ако се опитаме да направим това паралелно трябва да се добави синхронизация, защото може да се срещнат две ребра *(a, b)* и *(c, b)* едновременно, при което се получава едновременно писане в вектора b.

Пробвани са няколко подхода

- Последователен алгоритъм - използвайки само една нишка, няма нужда от синхронизация.
- Векторите се заключват поотделно. Изпробвано е с ОС mutex и spin lock.
- Използва се допълнителна lock-free структура от данни към всеки вектор, в която се записват върховете които трябва да се добавят. След обхождането те могат да се добавят паралелно без проблем

След тестване най-бързо се оказа последователния алгоритъм, затова се използва той.

3.2 Матрица на съседство

При матрицата на съседство разпарелляването на алгоритъма е лесно и в двата случая

Ориентиран граф Разделяме задачата на подзадачи, като всяка подзадача генерира фиксиран брой ребра. Подзадачите се изпълняват едновременно от thread pool. Всяка подзадача генерира случайно ребро *(u, v)* и го записва в матрицата и проверява дали вече съществува с една атомарна операция. Това продължава докато не се генерират 128 ребра.

Неориентиран граф Идеята е същата като при ориентирания, с изключението че се генерират само ребра за които $u < v$. Нишката пробва да добави реброто *(u, v)*. Ако това успее, то реброто *(v, u)* също не съществува и никой друг няма да се опита да го добави, затова нишката го добавя и него.

3.3 Резултати

Списъци на съседство - ориентиран граф

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0

Списъци на съседство - неориентиран граф

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0

Матрица на съседство - ориентиран граф

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0

Матрица на съседство - неориентиран граф

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0

4 Обхождане на графа

4.1 Алгоритми

Резултатът от обхождането на графа е гора от дървета, като всяко дърво е представено като корен и списък от наредени двойки *(родител, дете)*. Списъкът е сортиран в реда в който DFS алгоритъмът е намерил съответното дете. Може да има повече от едно дърво ако графът не е свързан. Имплементирани са три различни DFS алгоритъма.

Seq Стандартен еднонишков dfs алгоритъм. Използва се за база за сравнение.

Cheat Взима t на брой случайни върха, по един за всяка нишка, и пуска dfs от всеки от тях. Използва се споделен масив за проверка на това кои върхове са обходени. Първата нишка, която намери връх го маркира за обходен. Ако нишка види обходен връх тя го игнорира. Резултатът от алгоритъма се различава от това което стандартния еднонишков вариант би върнал, но за сметка на това се печели скорост.

Par Пуска се еднонишков dfs алгоритъм, който изпълнява само "спускането". Ако в момента сме във връх u то се преминава към първото необходено дете на u. Ако u няма необходени деца, алгоритъма спира. Резултатът от това е стек с всички намерени по време на спускането върхове. Следващата стъпка е връщането назад (backtracking) - при което трябва да се проверят и останалите необходени деца (след първото) на елементите в стека. Тази стъпка се изпълнява паралелно. За всеки елемент от стека се създава под-задача, която изпълнява нормален (пълен) dfs започвайки от този връх. Задачите се добавят в thread pool.

Така пуснатите задачи все още могат да намерят едни и същи върхове. Затова всеки пуснат dfs има цифров приоритет, като dfs-ите с елементи от началото на стека имат по-голям приоритет. Пази се споделен масив, в който за всеки намерен връх се записва приоритета, с който е намерен. Когато една нишка намери връх, тя проверява в масива с приоритетите. Ако този елемент вече съществува с по-голям приоритет, той се смята за обходен и се игнорира. Ако не съществува или има по-малък приоритет, то приоритета се презаписва с по-големия и обхождането продължава.

След приключване на втората стъпка имаме отделни дървета от всяка под-задача. Последната стъпка включва премахване на елементи, които са били обходени повече от веднъж от всички дървета освен това с най-голям приоритет и обединяването на всички дървета в едно. Времето за тези операции е незначително спрямо по-горните стъпки.

Този алгоритъм връща същия резултат като еднонишков dfs.

4.2 Резултати

par - списъци на съседство

нишки	време(сек)	спускане	връщане	S_p	E_p
seq	1	-	-	1	1
1	2	0	0	0	0
2	3	0	0	0	0
3	3	0	0	0	0
4	3	0	0	0	0
6	3	0	0	0	0
8	3	0	0	0	0
10	3	0	0	0	0
12	3	0	0	0	0
14	3	0	0	0	0
16	3	0	0	0	0
20	3	0	0	0	0
24	3	0	0	0	0

par - матрица на съседство

нишки	време(сек)	спускане	връщане	S_p	E_p
seq	1	-	-	1	1
1	2	0	0	0	0
2	3	0	0	0	0
3	3	0	0	0	0
4	3	0	0	0	0
6	3	0	0	0	0
8	3	0	0	0	0
10	3	0	0	0	0
12	3	0	0	0	0
14	3	0	0	0	0
16	3	0	0	0	0
20	3	0	0	0	0
24	3	0	0	0	0

cheat - списъци на съседство

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0

cheat - матрица на съседство

нишки	време(сек)	S_p	E_p
seq	1	1	1
1	2	0	0
2	3	0	0
3	3	0	0
4	3	0	0
6	3	0	0
8	3	0	0
10	3	0	0
12	3	0	0
14	3	0	0
16	3	0	0
20	3	0	0
24	3	0	0