ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ПОИСКА В ШИРИНУ В ГРАФЕ

Поиск в ширину (BFS – Breadth-First Search) является одним из методов обхода графа. Поиск в графах играет важную роль в анализе больших наборов данных. Поиск в ширину можно использовать для нахождения кратчайших путей между вершинами, для определения двудольности связной компоненты графа, для выделения компонент связности неориентированного графа, а также для поиска множества путей между двумя вершинами, длина которых лежит в заданном диапазоне.

BFS входит в тестовый комплект проекта Graph 500. Список Graph 500 содержит рейтинги (оценку производительности) суперкомпьютеров, связанных с задачами по обработке очень больших массивов данных.

Последовательный алгоритм

Пусть $G(V,E)$ – граф, V – множество вершин, E – множество ребер. Будем считать, что вершины графа занумерованы последовательными целыми числами, начиная от 0 и заканчивая n–1, а граф задан матрицей смежности A размера $n \times n$, где $a_{u,v}=1$, если существует ребро (дуга в случае орграфа), идущая из вершины u в вершину v, и $a_{u,v}=0$ в противном случае.

Приведем последовательный алгоритм BFS [1].

Вход: граф $G(V,E)$, стартовая вершина v_s.

1. Присвоить стартовой вершине v_s метку (уровень) $L(v_s)=0$, занести ее в очередь и считать помеченной.

2. Повторить следующую последовательность действий, пока все вершины не помечены и очередь не станет пустой. Пусть первым элементом очереди является вершина u и $Q(u)$ – множество непомеченных вершин, смежных с вершиной u; тогда

а) каждую вершину $v \in Q(u)$:

пометить меткой (уровнем) $L(v)=L(u)+1$,

занести v в очередь,

запомнить, что вершина v была помечена из вершины u: $P(v)=u$;

б) вершину u считать просмотренной, удалить ее из очереди и вернуться к шагу 2 алгоритма.

Выход: массив вершин $L[1..n]$, где $L(v)$ – расстояние между вершинами v и v_s (уровень v); массив вершин $P[1..n]$, где $P(v)$ – вершина, из которой была помечена вершина v (родитель v).

Рассмотрим пример. Пусть орграф имеет четыре вершины, $v_s=0$, задана матрица смежности

$$A=\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Помечаем стартовую вершину: присваиваем $L(v_S)=L(0)=0$; заносим вершину 0 в очередь F: $F=\{0\}$.

Просматриваем вершину 0. Помечаем вершины множества $Q(0)=\{1, 3\}$: $L(1)=1$, $P(1)=0$, $L(3)=1$, $P(3)=0$; заносим эти вершины в очередь, вершину 0 из очереди удаляем: $F=\{1, 3\}$.

Просматриваем вершины 1 и 3. Помечаем вершину множества $Q(1)=\{2\}$: $L(2)=2$, $P(2)=1$; заносим вершину 2 в очередь. Множество $Q(3)$ пустое, так как смежная с вершиной 3 вершина 1 уже помечена. Вершины 1 и 3 из очереди удаляем. Теперь $F=\{2\}$.

Просматриваем вершину 2. Эта вершина не имеет смежных вершин, поэтому $Q(2)=\varnothing$, все вершины помечены, $F=\varnothing$.

Начальный параллельный алгоритм

Рассмотрим начальную, упрощенную версию параллельного алгоритма BFS для выполнения на многоядерном компьютере [2]. Отметим, что в работе [2] уровень вершин (расстояние от вершины до стартовой вершины) не запоминается; вершина v помечается вершиной, из которой v занесена в очередь.

На каждой итерации (на каждом уровне) алгоритма используется очередь CQ (current queue) – набор вершин, которые должны быть просмотрены на текущем уровне l (в [2] употреблено «посещены» (must be visited); но visited далее используется также и смысле «помечены»; чтобы не было путаницы, «посещены» употреблять не будем). Инициализируется очередь CQ начальной вершиной v_S; на нулевом уровне начальная вершина просматривается. На первом уровне CQ содержит соседей v_S, параллельно просматриваются соседи v_S. На втором уровне в CQ содержатся соседи соседей v_S (только те, которые не были просмотрены на уровнях 0 и 1) и так далее. На каждом итерации алгоритма используется также очередь NQ – вершины, которые требуется просмотреть на следующем уровне (сюда добавляются соседи вершин из CQ, которые еще не были помечены). После просмотра всех вершин на текущем уровне, т.е. когда на текущем уровне из CQ удалены все вершины, CQ и NQ меняются местами (Swap) и алгоритм переходит на новый уровень. Представленный ниже псевдокод иллюстрирует работу алгоритма.

Алгоритм 1 (параллельный алгоритм BFS)

Вход: граф $G(V,E)$, стартовая вершина v_S.

Выход: массив $P[1..n]$, где $P[v]$ – вершина, из которой занесена в очередь вершина v (родитель v) // можно рассматривать и выдавать массив $L(v)$.

Данные: CQ – очередь вершин для просмотра на текущем уровне,
NQ – очередь вершин для просмотра на следующем уровне.

```
1:  for v ∈ V in parallel do
2:      P[v] ← ∞
    endfor
3:  P[v_s] ← 0;  // здесь можно пометить v_s уровнем: присвоить L(v_s)=0
4:  CQ← Enqueue v_s;  // добавить в очередь вершину v_s
5:  while CQ≠Ø do  // это цикл, каждая итерация которого есть
                                текущий уровень l
6:      NQ ← Ø;  //
7:      for u ∈ CQ in parallel do
8:          u ← Dequeue CQ  // извлечь из очереди CQ вершину u;
9:          for каждой v смежной к u in parallel do
10:             if P[v] ← ∞ then
11:                 P[v] ←u;  // здесь можно присвоить  L(v)=L(u)+1
12:                 NQ ← Enqueue v;  // добавить в очередь вершину v;
                endif
            endfor
        endfor
13:     Swap(CQ,NQ);
    endwhile
```

Строка "**for** .. *in parallel* **do**" обозначает, что итерации цикла можно выполнять независимо. Строки 10-12 и 8 должны выполняться атомарно (как единая операция, выполняемая как бы за один такт процессора), чтобы избежать гонок (избежать конфликтов при чтении и записи данных).

Параллельный алгоритм для реализации на многоядерном компьютере

Следующий Алгоритм 2 является улучшенной модификацией предыдущего алгоритма: добавлены атомарные операции (строки 11, 15, 18) и две важные оптимизации [2].

Алгоритм 2 (оптимизированный параллельный алгоритм BFS)

Вход: граф $G(V,E)$, стартовая вершина v_s, число вершин n.

Выход: массив $P[1..n]$, где $P[v]$ – вершина, из которой занесена в очередь вершина v (родитель v) // можно рассматривать и выдавать массив $L(v)$.

Данные: $Bitmap[v]$: бит содержит 1, если вершина v помечена, 0 – иначе,
CQ – очередь вершин для просмотра на текущем уровне,
NQ – очередь вершин для просмотра на следующем уровне.

Функции (выполняются атомарно)**:**

- *LockedDequeue*(*Q*) – за одну операцию возвращает первый элемент из очереди *Q* и обновляет указатель на первый элемент,
- *LockedReadSet*(*a*,*val*) – за одну операцию возвращает текущее значение *a* и присваивает ему новое значение *val*,
- *LockedEnqueue*(*Q*,*val*) – за одну операцию заносит значение *val* в конец очереди *Q* и обновляет указатель на конец очереди.

```
 1: for v ∈ V in parallel do //
 2:     P[v] ← ∞;
    endfor
 3: for i = 1, n in parallel do
 4:     Bitmap[i] ← 0;
    endfor
 5: P[v_s] ← 0; // здесь можно пометить v_s уровнем: присвоить L(v_s)=0
 6: CQ ← Enqueue v_s; // добавить в очередь вершину v_s
 7: fork (разветвление по потокам); //
 8: while CQ≠Ø do // это цикл, каждая итерация которого есть
                     текущий уровень l
 9:     NQ ←≠Ø //
10:     while CQ≠Ø in parallel do
11:         u ← LockedDequeue(CQ);
12:         for каждой v смежной к u in parallel do
13:             a ← Bitmap[v];
14:             if a = 0 then // [if Bitmap[v]=0 then вместо 13, 14 хуже?]
15:                 prev = LockedReadSet(Bitmap[v],1);
16:                 if prev = 0 then
17:                     P[v] ← u; // здесь можно присвоить L(v)=L(u)+1
18:                     LockedEnqueue(NQ,v);
                    endif
                endif
            endfor
        endwhile
19:     Synchronize;
20:     Swap(CQ,NQ);
    endwhile
21: join;
```

Первая оптимизация – использование дополнительного бита (Bitmap) для маркировки помеченных вершин. Это позволяет значительно уменьшить число операций с вершинами. В 4МБ можно хранить информацию, являются помеченными или нет 32 миллиона вершин.

Более тонкая оптимизация выполняется в строках 13 и 14. Избегается потенциально затратная атомарная операция в строке 15. Вместо того чтобы всегда записывать данные в Bitmap (строка 15) проверяется, была ли помечена данная вершина (строки 13 и 14), и, если вершина еще не помечена,

происходит запись. В это время другие потоки могут выполнять строки 13 (читать Bitmap), 14 и 15 (так как несколько потоков могут получить в строке 13 значение $a = 0$). Но только один поток получит в строке 15 значение *prev*, равное 0.

Параллельный алгоритм для реализации на многоядерном многосокетном компьютере

Рассмотрим алгоритм BFS для реализации на суперкомпьютерах, использующих многосокетные узлы с многоядерными процессорами в сокетах. При реализации алгоритмов на таких компьютерах обмен данными (осуществляемый посредством общей разделяемой памяти) между потоками, выполняемыми на ядрах разных сокетов более дорогостоящий, чем обмен данными в пределах одного сокета.

Алгоритм 3 получен на основе Алгоритма 2. Фразы типа «вычисления сокета», «вершины сокета» будем понимать как «вычисления, выполняемыми потоками на ядрах сокета», «вершины, назначенные для вычислений на ядрах сокета».

Алгоритм 3 (параллельный алгоритм BFS для многосокетных вычислений) [2]

Вход: граф $G(V,E)$, стартовая вершина v_s, число вершин n (кратное числу сокетов).

Выход: массив $P[1..n]$, где $P[v]$ – вершина, из которой занесена в очередь вершина v (родитель v) // можно рассматривать и выдавать массив $L(v)$

Данные: *Bitmap*[v]: бит содержит 1, если вершина v помечена, 0 – иначе,

$CQ[s]$ – очередь вершин для просмотра на текущем уровне в сокете s,

$NQ[s]$ – очередь вершин для просмотра на следующем уровне в сокете s,

$SQ[s]$ – дополнительная очередь для просмотра на следующем уровне в сокете s; содержит пары (v,u), где u – родитель v; находится в сокете s, но данные в $SQ[s]$ заносятся из других сокетов.

Функции:

- *LockedDequeue*(Q) – за одну операцию возвращает первый элемент из очереди Q и обновляет указатель на первый элемент,
- *LockedReadSet*(a,*val*) – за одну операцию возвращает текущее значение a и присваивает ему новое значение *val*,
- *LockedEnqueue*(Q,*val*) – за одну операцию заносит значение *val* в конец очереди Q и обновляет указатель на конец очереди,
- *GetTotalSockets*() – возвращает число сокетов,
- *GetMySocket*() – возвращает ID текущего сокета,
- *DetermineSocket*(v) – возвращает ID сокета, которые владеет вершиной v.

1: *sockets* = *GetTotalSockets*()

2: Разбиение графа, выделение каждому процессору $ns = \frac{n}{sockets}$ вершин.

Разбиения *P* и *Bitmap*, соответствующие разбиению графа: если вершиной *v* владеет сокет *s*, то *P*[*v*] и *Bitmap*[*v*] также принадлежат *s*.

3: **for** *i* = 1, *n* **do** // *in parallel* [в [2] в алгоритме 3 нет *in parallel*]

4: $P[v] \leftarrow \infty$;

5: *Bitmap*[*v*]←0;

endfor

6: $P[v_s] \leftarrow 0$; // здесь можно пометить v_s уровнем: присвоить $L(v_s)=0$

7: **for** *s* = 1 *to sockets* **do**

8: *CQ*[*s*] ← *Ø*;

9: *NQ*[*s*] ← *Ø*;

endfor

10: *CQ*[*DetermineSocket*(v_s)] ← *Enqueue* v_s;

11: fork;

12: *this* = *GetMySockets*();

13: **while** *CQ*[*this*] ≠*Ø* **do** // это цикл, каждая итерация которого есть текущий уровень ***l***

14: **while** *CQ*[*this*] ≠*Ø* **do** // *in parallel*

15: *u* ← *LockedDequeue*(*CQ*[*this*]);

16: **for** *каждой v смежной к u* **do** // *in parallel*

17: *s* ← *DetermineSocket*(*v*);

18: **if** *s* = *this* **then**

19: *a* ←*Bitmap*[*v*];

20: **if** *a* = 0 **then**

21: *prev* = *LockedReadSet*(*Bitmap*[*v*],1);

22: **if** *prev* = 0 **then**

23: *P*[*v*] ← *u*; // здесь можно присвоить *L*(*v*)=*L*(*u*)+1;

24: *LockedEnqueue*(*NQ*[*this*],*v*);

endif

endif

25: **else**

26: *LockedEnqueue*(*SQ*[*s*], (*v*,*u*));

endif

endfor

endwhile

27: Synchronize;

28: **while** *SQ*[*this*] ≠*Ø* **do** // *in parallel*

29: (*v*,*u*) ← *LockedDequeue*(*SQ*[*this*]);

30: *a* ←*Bitmap*[*v*];

31: **if** *a* = 0 **then**

32: *prev* = *LockedReadSet*(*Bitmap*[*v*],1);

33: **if** *prev* = 0 **then**

```
34:              P[v] ← u;  // здесь можно присвоить  L(v)=L(u)+1;
35:              LockedEnqueue(NQ[this],v);
                endif
              endif
            endwhile
36:         Synchronize;
37:         Swap(CQ[this], NQ[this]);
38:         NQ[this] ← Ø;
          endwhile
39:       join;
```

В Алгоритме 3 устанавливается сокет, которому принадлежит дочерняя вершина (строки 17 и 18). Если вершина принадлежит текущему сокету, то действия такие же, как в алгоритме 2; если нет, то вершина с ее родителем заносятся в очередь *SQ*[*s*] сокета *s*, которому принадлежит дочерняя вершина. После синхронизации (строка 27) в каждом сокете проверяется своя очередь обмена *SQ*. В *SQ* хранятся вершины (+ родитель), которые необходимо пометить на текущем уровне. Поэтому на уровне ***l*** повторяется еще один цикл (строки 28–35) как в алгоритме 2, но не для *CQ*, а для *SQ*. В строке 36 проводится последняя синхронизация для текущего уровня, *NQ* содержат все вершины для нового уровня. В строке 37 *CQ* и *NQ* меняются местами и, если *CQ* не пусто, начинается итерация нового уровня.

В работе [2] отмечается, что Алгоритм 3 легко обобщается для реализации на компьютерах с распределенной памятью.

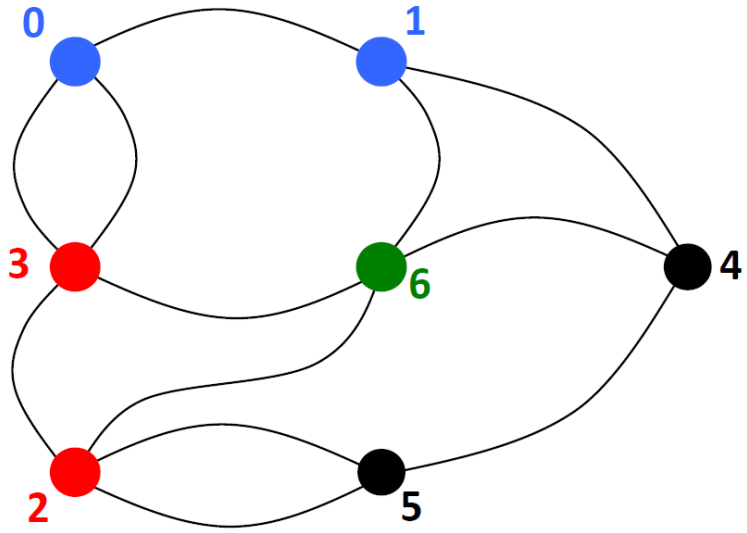
Параллельный 1D алгоритм для реализации на компьютере с распределенной памятью

Поиск по очень большим (миллиарды вершин и ребер) графам вызывает затруднение из-за огромного пространства для поиска: невозможно сохранить настолько большие графы в оперативной памяти одного компьютера. Возникает необходимость в разработке распределенных (т.е. предназначенных для распределенных вычислений) параллельных алгоритмов поиска в ширину.

Рассмотрим распределенный алгоритм поиска в ширину, использующий одномерное (1D) разбиение графа.

1D разбиение графа – такое разбиение, при котором ребра, выходящие из некоторой вершины, принадлежат тому же процессу, что и данная вершина. Вершины, принадлежащие некоторому процессу, называются его локальными вершинами.

Приведем пример 1D разбиения графа, изображенного на рисунке, для четырех процессов. Пусть вершины 0 и 1 принадлежат нулевому процессу, вершины 2 и 3 принадлежат первому процессу, вершины 4 и 5 принадлежат второму процессу, вершина 6 принадлежит третьему процессу.



Тогда ребра делятся между процессами следующим образом:

(0,1) (0,3) (0,3) (1,0) (1,4) (1,6)
(2,3) (2,5) (2,5) (2,6) (3,0) (3,0) (3,2) (3,6)
(4,1) (4,5) (4,6) (5,2) (5,2) (5,4)
(6,1) (6,2) (6,3) (6,4)

Пусть некоторый граф задан матрицей смежности A. Будем считать, не нарушая общности, что строки и столбцы матрицы A симметрично переупорядочены так, чтобы вершины, принадлежащие одному процессу, были приписаны соседним строкам и столбцам матрицы (например, строки 1, 2 и столбцы 1, 2 соответствуют вершинам процесса 0, а строки 3, 4 и столбцы 3, 4 – вершинам процесса 1). Тогда матрица смежности A имеет вид

$$\begin{pmatrix} A_1 \\ A_2 \\ \dots \\ A_p \end{pmatrix},$$

где A_i есть блок из строк матрицы смежности, принадлежащих процессу i–1.

Ребра, исходящие из вершины v формируют строку с номером v матрицы смежности; строка v формирует список смежности вершины v. Ребра, входящие в этот список, принадлежат тому же процессу, что и вершина v. Для балансировки разбиения, каждый процесс должен владеть примерно одинаковыми количествами вершин и исходящих из них ребер.

Распределенный поиск в ширину с 1D разбиением выполняется следующим образом. На итерации l алгоритма просматриваются вершины уровня l; помечаются вершины уровня l на предыдущей, (l–1)-й итерации (стартовая вершина помечается при инициализации). Пусть F – множество вершин уровня l, принадлежащих данному процессу. Из вершин, смежных вершинам из F, формируют множество N соседних вершин. Некоторые из этих вершин могут принадлежать данному процессу, а некоторые – не принадлежать. Во втором случае эти вершины посылаются соответствующим процессам (чтобы быть помеченными и оказаться во множествах F этих процессов на следующем уровне). Каждый процесс получает эти (посланные процессу из множеств N других процессов) вершины; уже помеченные вершины далее не рассматривает. Каждая вершина v помечается уровнем $L(v)$, который обозначает расстояние между вершинами v и v_s.

Структура данных $L(v)$ (и структура данных $P(v)$, если находить и родителей вершин) распределена: каждый процесс хранит данные лишь о своих вершинах. Информация о распределении вершин между процессами и о номере стартовой вершины хранится каждым процессом.

Алгоритм 4 (параллельный алгоритм BFS для распределенных вычислений с 1D разбиением графа) [3].

Вход: матрица смежности A, стартовая вершина v_s.

Выход: $L(v)$ – массив уровней вершин // $P(v)$ – массив родителей вершин.

Для каждого процесса с номером i:

1: Инициализация: $L(v)=0$, если $v=v_s$; $L(v)=\infty$, если $v\neq v_s$

2: **for** l = 0 to ∞ **do**

3: **Сформировать** $F=\{u \mid L(u)=l\}$ // множество локальных вершин уровня l (вершины, помеченные на предыдущей итерации на шаге 13 или стартовая вершина на нулевой итерации)

4: **if** для всех процессов $F=\varnothing$ **then**

5: завершить все вычислительные процессы

endif

6: **Сформировать** N – множество соседей вершин из F, пользуясь своими строками из матрицы A, т.е. строками блоков A_{i+1}

// для каждого v из N можно указывать родителя,
т.е. N формировать из пар (v,u), где u – родитель v
(из пар (v,u_1) и (v,u_2) включается пара с меньшим значением u)

7: **for all** processes q **do**

8: **Сформировать** N_q – множество вершин из N, распределенных процессу q // N_q могут быть пустыми

9: **Send** N_q to process q

10: **Receive** $\overline{N}_q$ from process q // $\overline{N}_q$ – вершины текущего процесса, принадлежащие множеству N процесса q

endfor

11: **Сформировать** множество (из локальных вершин) $\overline{N}=\bigcup_q \overline{N}_q$

// $\overline{N}_q$ могут пересекаться: ребра в разных процессах могут привести в одинаковые вершины; если $\overline{N}$ формируется из пар (v,u), то из пар (v,u_1) и (v,u_2) включается пара с меньшим значением u; $\overline{N}_q$ могут иметь помеченные вершины

12: **for** $v\in\overline{N}$ and $L(v)=\infty$ **do** // цикл по всем непомеченным вершинам из вновь сформированного множества локальных вершин

13: $L(v)=l+1$ // если N и $\overline{N}$ формировались из пар (v,u), то $P(v)=u$

endfor

endfor(l)

Пример применения параллельного Алгоритма 4. Пусть граф задан матрицей смежности:

$$A=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Граф с такой матрицей смежности является одним из вариантов ориентированного графа, полученного на основе рассмотренного ранее не ориентированного графа.

Пусть v_S=0, число процессов – 4, вершины 0 и 1 принадлежат нулевому процессу, вершины 2 и 3 принадлежат первому процессу, вершины 4 и 5 принадлежат второму процессу, вершина 6 принадлежит третьему процессу.

Разбиение опишем схемой:

		Pr_0	Pr_1	Pr_2	Pr_3
v		0 1	2 3	4 5	6
Pr_0	0	0 1 0 1 0 0 0			
	1	0 0 0 0 1 0 1			
Pr_1	2		0 0 0 0 0 1 1		
	3		1 0 1 0 0 0 1		
Pr_2	4			0 0 0 0 0 1 0	
	5			0 0 1 0 0 0 0	
Pr_3	6				0 0 0 0 1 0 0

Применим параллельный Алгоритм 4.

Инициализация:

Pr_0 : $L(v_S)=L(0)=0$, $L(1)=\infty$,

Pr_1 : $L(2)=\infty$, $L(3)=\infty$,

Pr_2 : $L(4)=\infty$, $L(5)=\infty$,

Pr_3 : $L(6)=\infty$.

Итерация l = 0:

Множества F:

Pr_0 : $F=\{0\}$,

Pr_1 , Pr_2 , Pr_3 : $F=\varnothing$.

Множества N (N формируем из пар (v,u), где u – родитель v), N_q (N_q указаны только не пустые):

Pr_0 : $N=\{(1,0),(3,0)\}$, $N_0=\{(1,0)\}$, $N_1=\{(3,0)\}$,

Pr_1, Pr_2, Pr_3 : $N=\varnothing$.

Результаты обмена множествами N_q:

Pr_0 : $\overline{N}_0=\{(1,0)\}$, $\overline{N}=\{(1,0)\}$,

Pr_1 : $\overline{N}_0=\{(3,0)\}$, $\overline{N}=\{(3,0)\}$,

Pr_2, Pr_3 : $\overline{N}=\varnothing$.

Метки $L(v)$ и $P(v)$ для $v\in\overline{N}$ (если $L(v)=\infty$):

Pr_0 : $L(1)=1$, $P(1)=0$,

Pr_1 : $L(3)=1$, $P(3)=0$,

Pr_2, Pr_3 : – .

Итерация l = 1:

Множества F:

Pr_0 : $F=\{1\}$,

Pr_1 : $F=\{3\}$,

Pr_2, Pr_3 : $F=\varnothing$.

Множества N, N_q:

Pr_0 : $N=\{(4,1),(6,1)\}$, $N_2=\{(4,1)\}$, $N_3=\{(6,1)\}$,

Pr_1 : $N=\{(0,3),(2,3),(6,3)\}$, $N_0=\{(0,3)\}$, $N_1=\{(2,3)\}$, $N_3=\{(6,3)\}$,

Pr_2, Pr_3 : $N=\varnothing$.

Результаты обмена множествами N_q:

Pr_0 : $\overline{N}_1=\{(0,3)\}$, $\overline{N}=\{(0,3)\}$,

Pr_1 : $\overline{N}_1=\{(2,3)\}$, $\overline{N}=\{(2,3)\}$,

Pr_2 : $\overline{N}_0=\{(4,1)\}$, $\overline{N}=\{(4,1)\}$,

Pr_3 : $\overline{N}_0=\{(6,1)\}$, $\overline{N}_1=\{(6,3)\}$, $\overline{N}=\{(6,1)\}$, ((v,u)=(6,3) в $\overline{N}$ не вошло, так как v=6 уже присутствует в паре (v,u)=(6,1).

Метки $L(v)$ и $P(v)$ для $v\in\overline{N}$ (если было $L(v)=\infty$):

Pr_0 : – (вершина 0 уже помечена: $L(0)=0$),

Pr_1 : $L(2)=2$, $P(2)=3$,

Pr_2 : $L(4)=2$, $P(4)=1$,

Pr_3 : $L(6)=2$, $P(6)=1$.

Итерация l = 2:

Множества F:

Pr_0 : $F=\varnothing$,

Pr_1 : $F=\{2\}$,

Pr_2 : $F=\{4\}$,

Pr_3 : $F=\{6\}$.

Множества N, N_q:

Pr_0 : $N=\varnothing$,

Pr_1 : $N=\{(6,2),(5,2)\}$, $N_2=\{(5,2)\}$, $N_3=\{(6,2)\}$,

Pr_2 : $N=\{(5,4)\}$, $N_2=\{(5,4)\}$,

Pr_3 : $N=\{(4,6)\}$, $N_2=\{(4,6)\}$.

Результаты обмена множествами N_q:

Pr_0 : $\overline{N}=\varnothing$,

Pr_1 : $\overline{N}=\varnothing$,

Pr_2 : $\overline{N}_1=\{(5,2)\}$, $\overline{N}_2=\{(5,4)\}$, $\overline{N}_3=\{(4,6)\}$, $\overline{N}=\{(5,2),(4,6)\}$,

((v,u)= (5,4) в $\overline{N}$ не вошло, так как v=5 уже присутствует в паре (v,u)=(5,2)),

Pr_3 : $\overline{N}_1=\{(6,2)\}$, $\overline{N}=\{(6,2)\}$.

Метки $L(v)$ и $P(v)$ для $v\in\overline{N}$ (если $L(v)=\infty$):

Pr_0 : – ,

Pr_1 : – ,

Pr_2 : $L(5)$=3, $P(5)$=2, (вершина 4 уже помечена),

Pr_3 : – (вершина 6 уже помечена).

Итерация l = 3:

Множества F:

Pr_0 : $F=\varnothing$,

Pr_1 : $F=\varnothing$,

Pr_2 : $F=\{5\}$,

Pr_3 : $F=\varnothing$.

Множества N, N_q:

Pr_0 : $N=\varnothing$,

Pr_1 : $N=\varnothing$,

Pr_2 : $N=\{(2,5)\}$, $N_1=\{(2,5)\}$,

Pr_3 : $N=\varnothing$.

Результаты обмена множествами N_q:

Pr_0 : $\overline{N} = \varnothing$,

Pr_1 : $\overline{N}_2 = \{(2,5)\}$, $\overline{N} = \{(2,5)\}$,

Pr_2 : $\overline{N} = \varnothing$,

Pr_3 : $\overline{N} = \varnothing$.

Метки $L(v)$ и $P(v)$ для $v \in \overline{N}$ (если $L(v) = \infty$):

Pr_0 : – ,

Pr_1 : – (вершина 2 уже помечена),

Pr_2 : – ,

Pr_3 : – .

Итерация l = 4:

Множества F: для всех процессов $F = \varnothing$,

все вычислительные процессы завершаются.

Параллельный 2D алгоритм для реализации на компьютере с распределенной памятью

Для оптимизации использования памяти и межпроцессорных коммуникаций вместо 1D разбиения графа используется двумерное (2D) разбиение. При 1D разбиении количество процессов, участвующих в групповых коммуникациях, равно $O(P)$, где P – общее количество процессов. При 2D разбиении количество процессов, вовлеченных в групповые коммуникации, равно $O\left(\sqrt{P}\right)$.

2D разбиение принято описывать посредством разбиения транспонированной матрицы смежности A^T на блоки. Блоки распределяются между процессами, так что каждое ребро принадлежит только одному процессу; кроме того, вершины тоже распределены между процессами.

Будем считать, что P процессов логически организованы в виде двумерного массива процессов размера $R \times C$. Процесс с координатами (i,j) обозначим $\mathrm{Pr}_{i,j}$ (нумерация i и j начинается с единицы). Пусть число $\dfrac{n_{adj}}{R \cdot C}$ является целым, где n_{adj} есть порядок матрицы смежности; если это не так, то можно добавить в граф фиктивные вершины.

Разбиение, которое мы будем использовать, опишем следующей схемой:

$$
\begin{array}{c}
\\
\mathrm{Pr}_{1,1}\\
\mathrm{Pr}_{2,1}\\
\vdots\\
\mathrm{Pr}_{R,1}\\
\cdots\\
\mathrm{Pr}_{1,C}\\
\mathrm{Pr}_{2,C}\\
\vdots\\
\mathrm{Pr}_{R,C}
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
\mathrm{Pr}_{1,1}-\mathrm{Pr}_{R,1} & & \mathrm{Pr}_{1,2}-\mathrm{Pr}_{R,2} & & \cdots & & \mathrm{Pr}_{1,C}-\mathrm{Pr}_{R,C}
\end{array}\\
\left(\begin{array}{cccccccc}
A_{1,1}^{(1)} & & A_{1,2}^{(1)} & & \cdots & & A_{1,C}^{(1)}\\
A_{2,1}^{(1)} & & A_{2,2}^{(1)} & & \cdots & & A_{2,C}^{(1)}\\
\vdots & & \vdots & & \vdots & & \vdots\\
A_{R,1}^{(1)} & & A_{R,2}^{(1)} & & \cdots & & A_{R,C}^{(1)}\\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots\\
A_{1,1}^{(C)} & & A_{1,2}^{(C)} & & \cdots & & A_{1,C}^{(C)}\\
A_{2,1}^{(C)} & & A_{2,2}^{(C)} & & \cdots & & A_{2,C}^{(C)}\\
\vdots & & \vdots & & \vdots & & \vdots\\
A_{R,1}^{(C)} & & A_{R,2}^{(C)} & & \cdots & & A_{R,C}^{(C)}
\end{array}\right)
\end{array}.
$$

Транспонированная матрица смежности A^T разбивается на $R \times C$ блочных строк и C блочных столбцов. Блок, обозначаемый $A_{i,j}^{(*)}$, принадлежит процессу $\mathrm{Pr}_{i,j}$. В каждом блоке $\frac{n_{adj}}{R \cdot C}$ строк и $\frac{n_{adj}}{C}$ столбцов. Один процесс владеет C блоками. Владение процесса блоком означает владение ребрами, принадлежащими этому блоку, т.е. ребрами, которые соответствуют ненулевым элементам этого блока транспонированной матрицы смежности.

Таким образом, каждый процесс владеет частичными списками смежности вершин, соответствующих входящим в блоки столбцам матрицы смежности. Суммарно процессы одного столбца владеют полными списками смежности принадлежащих им вершин.

Разбиения сбалансировано, если процессы владеют примерно одинаковым количеством ребер.

Вершины распределяются между процессами следующим образом: процесс $\mathrm{Pr}_{i,j}$ владеет вершинами, соответствующими блочной строке с номером $(j-1)R+i$. Каждый процесс $\mathrm{Pr}_{i,j}$ обладает информацией о распределении вершин между процессами строки i и о номере стартовой вершины.

Распределенный поиск в ширину, использующий описанное 2D разбиение, выполняется следующим образом. Как и в случае использования 1D разбиения, на итерации l алгоритма просматриваются вершины уровня l (помечаются вершины уровня l на предыдущей, $(l-1)$-й итерации). Пусть F – множество вершин уровня l, принадлежащих данному процессу. Рассмотрим вершину u из множества F.

Процесс-владелец вершины u посылает процессам из этого же столбца массива процессов сообщение о том, что вершина u является вершиной текущего уровня. Каждый из этих процессов содержит свою часть списков смежности для этой вершины. Назовем эту операцию *expand*. Каждый из процессов в этом столбце процессов в результате таких коммуникационных операций получает множество вершин, находит для этих вершин соседние

вершины с использованием своих списков смежности и формирует множество N потенциальных вершин следующего уровня.

Вершины из N затем отправляются их владельцам, которые могут находиться только в той же строке массива процессов; эту операцию назовем *fold*. Каждый процесс получает посланные ему вершины из множеств N других процессов; уже помеченные вершины далее не рассматривают (заметим, что операции 8–11 Алгоритма 4 соответствуют операции *fold*). Каждая вершина v помечается уровнем $L(v)$, который обозначает расстояние между вершинами v и v_s.

В операциях *expand* и *fold* заключается оптимизация по сравнению с 1D разбиением: в каждой операции группового обмена участвует лишь строка или столбец массива процессов, а не все процессы.

Структура данных $L(v)$ (и структура данных $P(v)$, если находить и родителей вершин) распределена: каждый процесс хранит данные лишь о своих вершинах. Каждый процесс обладает также информацией о распределении вершин между процессами своей процессорной строки и о номере стартовой вершины.

Алгоритм 5 (параллельный алгоритм BFS для распределенных вычислений с 2D разбиением графа) [3].

Вход: транспонированная матрица смежности A^T, стартовая вершина v_s.

Выход: $L(v)$ – массив уровней вершин // $P(v)$ – массив родителей вершин.

Для каждого процесса $\mathrm{Pr}_{i,j}$:

1: Инициализация: $L(v)=0$, если $v=v_s$; $L(v)=\infty$, если $v\neq v_s$

2: **for** l = 0 to ∞ **do**

3: **Сформировать** $F=\{u \mid L(u)=l\}$ // множество локальных вершин уровня l (вершины, помеченные на предыдущей итерации на шаге 17 или стартовая вершина на нулевой итерации)

4: **if** для всех процессов $F=\varnothing$ **then**

5: завершить все вычислительные процессы

endif

6: **for all** processes q in this processes column **do**

7: **Send** F to process q

8: **Receive** $\overline{F}_q$ from process q // $\overline{F}_q$ – это множество F процесса q

endfor

9: **Сформировать** множество $\overline{F}=\bigcup_q \overline{F}_q$

10: **Сформировать** N – множество соседей вершин из $\overline{F}$, пользуясь своими списками смежности
// для каждого v из N можно указывать родителя,
т.е. N формировать из пар (v,u), где u – родитель v
(из пар (v,u_1) и (v,u_2) включается пара с меньшим значением u)

11: **for all** processes q in this processes row **do**

12: **Сформировать** N_q – множество вершин из N,

распределенных процессу q // N_q могут быть пустыми

13: **Send** N_q to process q

14: **Receive** $\overline{N}_q$ from process q // $\overline{N}_q$ – вершины текущего процесса, принадлежащие множеству N процесса q

endfor

15: **Сформировать** множество (из локальных вершин) $\overline{N} = \bigcup_q \overline{N}_q$

// $\overline{N}_q$ могут пересекаться: ребра в разных процессах могут привести в одинаковые вершины; если $\overline{N}$ формируется из пар (v,u), то из пар (v,u_1) и (v,u_2) включается пара с меньшим значением u; $\overline{N}_q$ могут иметь помеченные вершины

16: **for** $v \in \overline{N}$ and $L(v) = \infty$ **do** // цикл по всем непомеченным вершинам из вновь сформированного множества локальных вершин

17: $L(v)=l+1$ // если N и $\overline{N}$ формировались из пар (v,u), то $P(v)=u$

endfor

endfor(l)

Операции 6–9 Алгоритма 5 соответствуют операции *expand*, операции 11–15 соответствуют операции *fold*.

Отметим, что каждое множество F посылается всем процессам столбца массива процессов, а множество N_q – конкретному процессу строки массива.

Пример применения параллельного Алгоритма 5. Рассмотрим граф с матрицей смежности, приведенной в примере применения параллельного Алгоритма 4, v_S=0. Пусть 4 процесса составляют двумерный массив размера 2×2. Имеем: n_{adj}=7, R=2, C=2. Для того чтобы число $\frac{n_{adj}}{R \cdot C}$ было целым, добавить в граф фиктивную вершину; тогда n_{adj}=8, $\frac{n_{adj}}{R \cdot C}$=2. Рассмотрим транспонированную матрицу смежности:

$$A^T = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Разбиение опишем схемой:

$$
\begin{array}{c} \\ \mathrm{Pr}_{1,1} \\ \mathrm{Pr}_{2,1} \\ \mathrm{Pr}_{1,2} \\ \mathrm{Pr}_{2,2} \end{array}
\begin{array}{c}
\begin{array}{cc} \mathrm{Pr}_{1,1}, \mathrm{Pr}_{2,1} & \mathrm{Pr}_{1,2}, \mathrm{Pr}_{2,2} \end{array} \\
\begin{pmatrix} A_{1,1}^{(1)} & A_{1,2}^{(1)} \\ A_{2,1}^{(1)} & A_{2,2}^{(1)} \\ A_{1,1}^{(2)} & A_{1,2}^{(2)} \\ A_{2,1}^{(2)} & A_{2,2}^{(2)} \end{pmatrix}
\end{array}
$$

или, в развернутом виде,

$$
\begin{array}{cc} & \begin{array}{cccc} \mathrm{Pr}_{1,1} & & \mathrm{Pr}_{2,1} & \end{array} \quad \begin{array}{cccc} \mathrm{Pr}_{1,2} & & \mathrm{Pr}_{2,2} & \end{array} \\ v & \boxed{0}\ \boxed{1} & \boxed{2}\ \boxed{3} & \boxed{4}\ \boxed{5} & \boxed{6}\ \boxed{7} \end{array}
$$

$$
\begin{array}{c}
\\
\mathrm{Pr}_{1,1}\ \boxed{0}\ \boxed{1} \\
\mathrm{Pr}_{2,1}\ \boxed{2}\ \boxed{3} \\
\mathrm{Pr}_{1,2}\ \boxed{4}\ \boxed{5} \\
\mathrm{Pr}_{2,2}\ \boxed{6}\ \boxed{7}
\end{array}
\begin{pmatrix}
A_{1,1}^{(1)} \begin{array}{|cccc|}\hline 0&0&0&1 \\ 1&0&0&0 \\ \hline \end{array} & A_{1,2}^{(1)} \begin{array}{|cccc|}\hline 0&0&0&0 \\ 0&0&0&0 \\ \hline \end{array} \\
A_{2,1}^{(1)} \begin{array}{|cccc|}\hline 0&0&0&1 \\ 1&0&0&0 \\ \hline \end{array} & A_{2,2}^{(1)} \begin{array}{|cccc|}\hline 0&1&0&0 \\ 0&0&0&0 \\ \hline \end{array} \\
A_{1,1}^{(2)} \begin{array}{|cccc|}\hline 0&1&0&0 \\ 0&0&1&0 \\ \hline \end{array} & A_{1,2}^{(2)} \begin{array}{|cccc|}\hline 0&0&1&0 \\ 1&0&0&0 \\ \hline \end{array} \\
A_{2,1}^{(2)} \begin{array}{|cccc|}\hline 0&1&1&1 \\ 0&0&0&0 \\ \hline \end{array} & A_{2,2}^{(2)} \begin{array}{|cccc|}\hline 0&0&0&0 \\ 0&0&0&0 \\ \hline \end{array}
\end{pmatrix}.
$$

Транспонированная матрица смежности A^T разбивается на четыре ($R \times C$=4) блочные строки и два (C=2) блочных столбца; блоки $A_{i,j}^{(1)}$ и $A_{i,j}^{(2)}$ принадлежат процессу $\mathrm{Pr}_{i,j}$. В каждом блоке две строки ($\frac{n_{adj}}{R \cdot C}$=2) и четыре столбца ($\frac{n_{adj}}{C}$=4). Один процесс владеет двумя блоками, т.е. владеет информацией о ребрах, которые соответствуют ненулевым элементам (если такие элементы есть) этих блоков транспонированной матрицы смежности. Распределение вершин: процесс $\mathrm{Pr}_{i,j}$ владеет вершинами, соответствующими блочной строке с номером $(j-1)2+i$.

Перейдем непосредственно к применению параллельного Алгоритма 5.

Инициализация:

$\mathrm{Pr}_{1,1}$: $L(v_s)=L(0)=0,\ L(1)=\infty$,

$\mathrm{Pr}_{2,1}$: $L(2)=\infty,\ L(3)=\infty$,

$\mathrm{Pr}_{1,2}$: $L(4)=\infty,\ L(5)=\infty$,

$\mathrm{Pr}_{2,2}$: $L(6)=\infty,\ L(7)=\infty$.

Итерация l = 0:

Множества F:

$\mathrm{Pr}_{1,1}$: $F = \{0\}$,

$\mathrm{Pr}_{2,1}$, $\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $F=\varnothing$.

Результаты обмена множествами F:

между процессами столбца 1:

$\mathrm{Pr}_{1,1}$, $\mathrm{Pr}_{2,1}$: $\overline{F} = \{0\}$,

между процессами столбца 2:

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $\overline{F} =\varnothing$.

Множества N (N формируем из пар (v,u), где u – родитель v), N_q (N_q указаны только не пустые):

$\mathrm{Pr}_{1,1}$: $N=\{(1,0)\}$, $N_{1,1} = \{(1,0)\}$,

$\mathrm{Pr}_{2,1}$: $N=\{(3,0)\}$, $N_{2,1} = \{(3,0)\}$,

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $N=\varnothing$.

Результаты обмена множествами N_q:

между процессами строки 1:

$\mathrm{Pr}_{1,1}$: $\overline{N}_{1,1} = \{(1,0)\}$, $\overline{N} = \{(1,0)\}$,

$\mathrm{Pr}_{1,2}$: $\overline{N} =\varnothing$,

между процессами строки 2:

$\mathrm{Pr}_{2,1}$: $\overline{N}_{2,1} = \{(3,0)\}$, $\overline{N} = \{(3,0)\}$,

$\mathrm{Pr}_{2,2}$: $\overline{N} =\varnothing$

(фактически обменов в данном случае не произошло).

Метки $L(v)$ и $P(v)$ для $v \in \overline{N}$ (если $L(v) = \infty$):

$\mathrm{Pr}_{1,1}$: $L(1)=1$, $P(1)=0$,

$\mathrm{Pr}_{2,1}$: $L(3)=1$, $P(3)=0$,

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: – .

Итерация l = 1:

Множества F:

$\mathrm{Pr}_{1,1}$: $F = \{1\}$,

$\mathrm{Pr}_{2,1}$: $F = \{3\}$,

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $F=\varnothing$.

Результаты обмена множествами F:

между процессами столбца 1:

$\mathrm{Pr}_{1,1}$, $\mathrm{Pr}_{2,1}$: $\overline{F} = \{1, 3\}$,

между процессами столбца 2:

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $\overline{F} = \varnothing$.

Множества N, N_q:

$\mathrm{Pr}_{1,1}$: $N = \{(4,1), (0,3)\}$, $N_{1,2} = \{(4,1)\}$, $N_{1,1} = \{(0,3)\}$,

$\mathrm{Pr}_{2,1}$: $N = \{(6,1), (2,3)\}$ ((v,u)=(6,3) в N не вошло, так как v=6 уже присутствует в паре (v,u)=(6,1)), $N_{2,2} = \{(6,1)\}$, $N_{2,1} = \{(2,3)\}$,

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $N = \varnothing$.

Результаты обмена множествами N_q:

между процессами строки 1:

$\mathrm{Pr}_{1,1}$: $\overline{N}_{1,1} = \{(0,3)\}$, $\overline{N} = \{(0,3)\}$,

$\mathrm{Pr}_{1,2}$: $\overline{N}_{1,1} = \{(4,1)\}$, $\overline{N} = \{(4,1)\}$,

между процессами строки 2:

$\mathrm{Pr}_{2,1}$: $\overline{N}_{2,1} = \{(2,3)\}$, $\overline{N} = \{(2,3)\}$,

$\mathrm{Pr}_{2,2}$: $\overline{N}_{2,1} = \{(6,1)\}$, $\overline{N} = \{(6,1)\}$.

Метки $L(v)$ и $P(v)$ для $v \in \overline{N}$ (если $L(v) = \infty$):

$\mathrm{Pr}_{1,1}$: – (вершина 0 уже помечена: было $L(0)$=0),

$\mathrm{Pr}_{2,1}$: $L(2)$=2, $P(2)$=3,

$\mathrm{Pr}_{1,2}$: $L(4)$=2, $P(4)$=1,

$\mathrm{Pr}_{2,2}$: $L(6)$=2, $P(6)$=1.

Итерация l = 2:

Множества F:

$\mathrm{Pr}_{1,1}$: $F = \varnothing$,

$\mathrm{Pr}_{2,1}$: $F = \{2\}$,

$\mathrm{Pr}_{1,2}$: $F = \{4\}$,

$\mathrm{Pr}_{2,2}$: $F = \{6\}$.

Результаты обмена множествами F:

между процессами столбца 1:

$\mathrm{Pr}_{1,1}$, $\mathrm{Pr}_{2,1}$: $\overline{F} = \{2\}$,

между процессами столбца 2:

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $\overline{F} = \{4, 6\}$.

Множества N, N_q:

$\mathrm{Pr}_{1,1}$: $N = \{(5,2)\}$, $N_{1,2} = \{(5,2)\}$,

$\mathrm{Pr}_{2,1}$: $N = \{(6,2)\}$, $N_{2,2} = \{(6,2)\}$,

$\mathrm{Pr}_{1,2}$: $N = \{(5,4), (4,6)\}$, $N_{1,2} = \{(5,4), (4,6)\}$,

$\mathrm{Pr}_{2,2}$: $N=\varnothing$.

Результаты обмена множествами N_q:

между процессами строки 1:

$\mathrm{Pr}_{1,1}$: $\overline{N}=\varnothing$,

$\mathrm{Pr}_{1,2}$: $\overline{N}_{1,1}=\{(5,2)\}$, $\overline{N}_{1,2}=\{(5,4),(4,6)\}$, $\overline{N}=\{(5,2),(4,6)\}$

((v,u)= (5,4) в $\overline{N}$ не вошло, так как v=5 уже присутствует в паре (v,u)=(5,2)),

между процессами строки 2:

$\mathrm{Pr}_{2,1}$: $\overline{N}=\varnothing$,

$\mathrm{Pr}_{2,2}$: $\overline{N}_{2,1}=\{(6,2)\}$, $\overline{N}=\{(6,2)\}$.

Метки $L(v)$ и $P(v)$ для $v\in\overline{N}$ (если $L(v)=\infty$):

$\mathrm{Pr}_{1,1}$: – ,

$\mathrm{Pr}_{2,1}$: – ,

$\mathrm{Pr}_{1,2}$: $L(5)=3$, $P(5)=2$, (вершина 4 уже помечена),

$\mathrm{Pr}_{2,2}$: – (вершина 6 уже помечена).

Итерация l = 3:

Множества F:

$\mathrm{Pr}_{1,1}$: $F=\varnothing$,

$\mathrm{Pr}_{2,1}$: $F=\varnothing$,

$\mathrm{Pr}_{1,2}$: $F=\{5\}$,

$\mathrm{Pr}_{2,2}$: $F=\varnothing$.

Результаты обмена множествами F:

между процессами столбца 1:

$\mathrm{Pr}_{1,1}$, $\mathrm{Pr}_{2,1}$: $\overline{F}=\varnothing$,

между процессами столбца 2:

$\mathrm{Pr}_{1,2}$, $\mathrm{Pr}_{2,2}$: $\overline{F}=\{5\}$.

Множества N, N_q:

$\mathrm{Pr}_{1,1}$: $N=\varnothing$,

$\mathrm{Pr}_{2,1}$: $N=\varnothing$,

$\mathrm{Pr}_{1,2}$: $N=\varnothing$,

$\mathrm{Pr}_{2,2}$: $N=\{(2,5)\}$, $N_{2,1}=\{(2,5)\}$.

Результаты обмена множествами N_q:

между процессами строки 1:

$\mathrm{Pr}_{1,1}$: $\overline{N}=\varnothing$,

$\mathrm{Pr}_{1,2}$: $\overline{N} = \varnothing$,

между процессами строки 2:

$\mathrm{Pr}_{2,1}$: $\overline{N}_{2,2} = \{(2,5)\}$, $\overline{N} = \{(2,5)\}$,

$\mathrm{Pr}_{2,2}$: $\overline{N} = \varnothing$.

Метки $L(v)$ и $P(v)$ для $v \in \overline{N}$ (если $L(v) = \infty$):

$\mathrm{Pr}_{1,1}$: – ,

$\mathrm{Pr}_{2,1}$: – (вершина 2 уже помечена),

$\mathrm{Pr}_{1,2}$: – ,

$\mathrm{Pr}_{2,2}$: – .

Итерация l = 4:

Множества F: для всех процессов $F = \varnothing$,

все вычислительные процессы завершаются.

Другие параллельные алгоритмы

Параллельный 2D алгоритм, использующий умножение матрицы смежности на вектор (параллельный алгебраический алгоритм поиска в ширину в графе) [4, 5]**.** Параллельный 2D алгоритм работы [3] ориентирован на графы малого диаметра с однородной (равномерной) степенью распределения (low-diameter graphs with uniform degree distribution). Параллельный 2D алгоритм, предложенный в работах [4, 5], ориентирован на графы малого диаметра с неравномерной степенью распределения (low-diameter graphs with skewed degree distribution) (именно такие графы рассматриваются в тесте BFS проекта Graph 500). Параллельный 2D алгоритм [4, 5] для реализации на компьютере с распределенной памятью основан на так называемом «шахматном» разбиении матрицы смежности.

Параллельные алгоритмы для реализации на GPU, на комбинировании GPU и многоядерного CPU [6–8]**.** Рассматриваются, в частности, оптимизации для графов с разреженными матрицами смежности и графов малого диаметра.

Сверху-вниз и снизу-вверх стратегии поиска. Рассмотренный алгоритм поиска в ширину реализует традиционный подход, так называемую сверху-вниз стратегию поиска (top-down search strategy): на очередной итерации ищутся соседи каждой просматриваемой на этой итерации вершины. Существует еще снизу-вверх стратегия поиска (bottom-up search strategy): на очередной итерации каждая непомеченная вершина проверяет вершины, из которых она доступна, пытаясь найти среди них родителя, т.е. вершину, помеченную на предыдущей итерации.

Снизу-вверх стратегия эффективна только тогда, когда на итерациях просматривается много вершин. Поэтому в начале и конце алгоритма поиска в ширину следует применять сверху-вниз стратегию, а для средних итераций

следует применять снизу-вверх стратегию. Параллельный алгоритм, реализующий такой подход, предложены в работах [9, 10]. Вычислительные эксперименты показали, что гибридное применение двух стратегий дает значительное повышение производительности.

В работе [4] представлены результаты вычислительных экспериментов, в которых сравнивались параллельные алгоритмы с применением 1D и 2D разбиения и сверху-вниз стратегией поиска, а также модифицированные (не просто MPI-реализации, а MPI+OpenMP-реализации) алгоритмы. Эксперименты показали, что параллельные алгоритмы с применением 2D разбиения допускают, в зависимости от используемого суперкомпьютера, получение как лучшей, так и худшей, по сравнению со случаем 1D разбиения, производительности (время на коммуникации всегда лучше).

Строчный формат хранения разреженных матриц

В рассмотренных примерах фигурировали плотные матрицы и векторы. В действительности они являются разреженными и на практике используется разреженный строчный формат (CSR – Compressed Sparse Rows) хранения разреженных матриц [11]. CSR предъявляет минимальные требования к памяти и обеспечивает эффективный доступ к строкам матрицы; доступ к столбцам затруднен, поэтому предпочтительно использовать эту схему хранения в тех алгоритмах, в которых преобладают строчные операции.

В соответствии с CSR-схемой, для хранения матрицы A требуется три одномерных массива:

массив ненулевых элементов матрицы A, в котором они перечислены по строкам от первой до последней (обозначим его как values);

массив номеров столбцов для соответствующих элементов массива values (обозначим его как cols);

массив указателей позиций, с которых начинается описание очередной строки (обозначим его pointer). Описание k-й строки хранится в позициях с pointer[k]-й по (pointer[k+1]–1)-ю массивов values и cols. Если pointer[k]=pointer[k+1], то k-я строка пустая. Если матрица A состоит из n строк, то длина массива pointer будет n +1, причем (n +1)-й элемент массива на 1 превышает количество элементов массива values (и массива cols).

Рассмотрим представление в разреженном строчном формате матрицы A из примеров применения параллельных алгоритмов BFS:

values=(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1);

cols=(4, 1, 4, 6, 1, 2, 7, 3, 5, 2, 3, 4);

pointer=(1, 3, 5, 7, 10, 11, 12, 13).

СПИСОК ЛИТЕРАТУРЫ

1. Котов В.М., Соболевская Е.П., Толстиков А.А. Алгоритмы и структуры данных. Учебное пособие. Минск: БГУ, 2011. 267 с.

2. Agarwal V., Petrini F., Pasetto D., Bader D. Scalable graph exploration on multicore processors // Proceedings of the ACM/IEEE supercomputing 2010 conference. New Orleans, LA, November 13–19, 2010.

3. Yoo A., Chow E., Henderson K., McLendon W., Hendrickson B., Catalyurek U.V: A scalable distributed parallel breadth-first search algorithm on BlueGene/L // Proceedings of the ACM/IEEE supercomputing 2005 conference. November 2005.

4. Buluc A., Madduri K. Parallel breadth-first search on distributed memory systems // Proceedings of the ACM/IEEE supercomputing 2011 conference. November 2011.

5. Buluc A., Madduri K. Graph partitioning for scalable distributed graph computations // 10th DIMACS implementation challenge, graph partitioning and graph clustering. February 13–14, 2012 Atlanta, Georgia.

6. Luo L., Wong M., Hwu W. An effective GPU implementation of breadth-first search // Proceedings of the design automation 2010 conference. June 2010.

7. Hong S., Kim S., Oguntebi T., Olukotun K. Accelerating CUDA graph algorithms at maximum warp // Proceedings of the 16th ACM symposium on principles and practice of parallel programming, 2011.

8. Hong S., Kim S., Oguntebi T., Olukotun K. Efficient parallel graph exploration on multi-core CPU and GPU // Proceedings of the parallel architectures and compilation techniques 2011 conference.

9. Beamer S., Buluc A., Asanovic K., Patterson D. Distributed memory breadth-first search revisited: enabling bottom-up search // IEEE international symposium on parallel and distributed processing. May 20–24, 2013. Cambridge, MA, USA.

10. Черноскутов М.А. Параллельная высокопроизводительная обработка графов // Международная научная конференция «Параллельные вычислительные технологии» (ПаВТ’2016): Труды международной научной конференции (Архангельск, 28 марта – 1 апреля 2016 г.). С. 736–742. (Доклад в электронном виде)

11. Писсанецки С. Технология разреженных матриц. М.: Мир. 1988. 411 с.