Распараллеливание алгоритма Флойда-Уоршелла для нахождения кратчайших путей в графе

Строгова М.Д.

Научный руководитель – д.т.н., доцент Курносов М.Г.

В работе выполнен анализ параллельного алгоритма Флойда-Уоршелла для нахождения кратчайших путей во взвешенном ориентированном графе с использование стандарта MPI. Выполнены анализы эффективности данного метода и его масштабируемости, проведены эксперименты на вычислительном кластере Jet.

Ключевые слова: параллельное программирование, стандарт MPI, алгоритм Флойда, кратчайший путь, граф.

Введение

Взвешенный ориентированный граф – это граф, каждому ребру которого присвоено некое значение – вес и направление. Задача о кратчайшем пути в таком графе – задача поиска самого короткого пути (цепи) между двумя вершинами на графе, в которой минимизируется сумма весов рёбер, составляющих путь. Данная задача является одной из важнейших классических задач теории графов. Один из множества алгоритмов для ее решения является алгоритм Флойда-Уоршелла, разработанный в 1962 году Р. Флойдом и С. Уоршеллом. Алгоритм применим к графам с произвольным, в том числе с отрицательными, весами, но без отрицательных циклов. Общая суть алгоритма состоит в выборе минимального значения между заданным весом ребра ориентированного графа и суммой двух путей от исходном к конечной вершине через возможные другие [1-3].

1. Алгоритм Флойда-Уоршелла

Веса графа представляются в виде квадратной матрицы $n \times n$, где n – количество вершин. Соответственно, W_{ij} – расстояние между вершинами i и j. При отсутствии ребра между i и j значение $W_{ij} = 0$, также считается что $W_{ii} = 0$.

Пусть введено обозначение – d^k_{ij} для длины кратчайшего пути от i до j, который кроме самих вершин проходит только через вершины $1 \ldots k$. Очевидно, что d^0_{ij} – длина (вес) ребра (i, j), если таковое не существует. Существует два варианта значения d^k_{ij}, $k \in \{1, \ldots, n\}$, n – количество строк/столбцов в матрице весов:

1. Кратчайший путь между i, j не проходит через вершину k, тогда $d^k_{ij} = d^{k-1}_{ij}$;
2. Существует более короткий путь между i, j, проходяший через k, тогда он сначала идет от i до k, а потом от k до j.

Таким образом, для нахождения значения функции достаточно выбрать минимум из двух обозначенных значений. Тогда формула для d^k_{ij} имеет вид:

- d^0_{ij} — отсутствие пути между i и j;
- $d^k_{ij} = \min(d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj})$

Последовательный алгоритм Флойда-Уоршелла вычисляется все значения d^k_{ij}. На каждом шаге алгоритм обновляет значение матрицы весов W, $W_{ij} = d^n_{ij}$. Таким образом, алгоритм имеет три вложенных цикла (рис. 1), то есть алгоритм имеет кубическую сложность – $O(n^3)$.

```
for k = 1 to n
      for i = 1 to n
            for j = 1 to n
                  W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Рис. 1. Псевдокод последовательной реализации алгоритма Флойда-Уоршелла.

2. Параллельный алгоритм

Выполнено распараллеливание алгоритма Флойда-Уоршелла для нахождения кратчайшего пути между вершинами взвешенного ориентированного графа путем одномерной декомпозиции матрицы весов W. Программная реализация метода выполнена на языке C в стандарте MPI. Равномерное распределение строк матрицы по вычислительным процессам выполнено следующим образом: каждому процессу достается целая часть от деления количества строк n на количество процессов p плюс по одной строке от возможного остатка. Пример подобного распределения представлен на рис. 2.

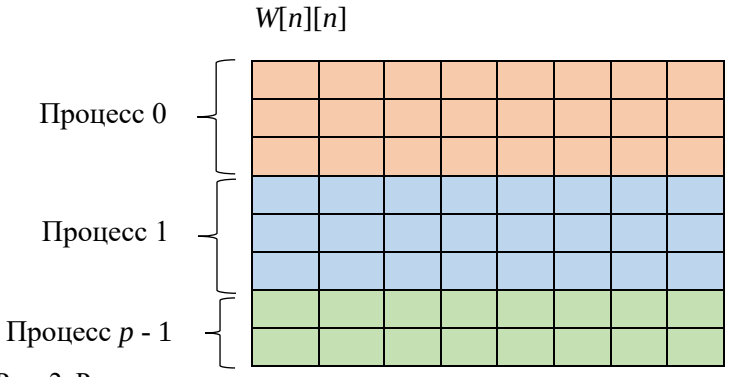

Рис. 2. Равномерное распределение строк матрицы весов по процессам: $n = 8$, $p = 3$.

В параллельной реализации каждому процессу назначается порядка $O(n/p)$ строк матрицы вместо хранения целой матрицы весов в корневом процессе, что сокращает расход памяти и время работы алгоритма из-за ненадобности передачи строк процессам. При начале работы алгоритма запускается цикл, проходящий по всем строкам, через значения которых рассматривается кратчайший путь для сравнения. Далее процессы определяют, кому принадлежит k-ая строка на данной итерации. «Владелец» передает данную строку всем остальным процессам при помощи метода широковещательной рассылки Bcast и вычисляет с ее помощью кратчайшие пути в своей подматрице. Остальные же процессы принимают строку и также делают вычисления (рис. 3).

```
index = 0
recive = 0
for k = 1 to n
     if k >= lb and k <= ub
          index = (rank == 0) ? k : k % lb
          Bcast(m[index], n, rank)
          for i = 1 to nrows
               for j = 1 to n
                    if lb + i != j
                         m[i][j] =
                                 min((m[i][j], m[i][k] + m[index][j])
                    end if
               end for
          end for
     else if k < lb || k > ub
          if k < max_nrows * amount_of_max
               recive = k / max_nrows
          else
               recive = (k - difference * amount_of_max) / min_nrows
          end if
          Bcast(k_row, n, recive)
          for i = 1 to nrows
               for j = 1 to n
                    if lb + i != j
                         m[i][j] = min(m[i][j], m[i][k] + k_row[j])
                    end if
               end for
          end for
     end if
end for
```

Рис. 3. Псевдокод параллельной реализации алгоритма Флойда-Уоршелла.

На рис. 3 представлена реализация определения «владельца» *k*-ой строки и ее широковещательной передачи. Введены следующие обозначения: *max_nrows* – максимальное количество строк, которое может быть назначено процессу; *amount_of_max* – количество процессов, которым назначено *max_nrows* строк; *min_nrows* – минимальное количество строк, которое может быть назначено процессу;

$$difference = max\_nrows - min\_nrows.$$

Таким образом, параллельная реализация характеризуется временем обменов $O(nT_{\text{Bcast}})$, а время вычислений составляет $O(n^3/p)$, что значительно меньше, чем время, работы последовательного алгоритма.

3. Результаты экспериментов

Исследования эффективности параллельной программы проводились на кластере Jet с сетью связи Gigabit Ethernet. На каждом узле 2 процессора Intel Xeon E5420 с оперативной памятью в 8 GB.

Эксперименты проводились на 2, 4, 8, 16, 32, 48, 64 процессах и при размерности матрицы весов равной 3000x3000 элементов. Эксперименты показали, что программа характеризуется ускорением близким к линейному (рис. 4).

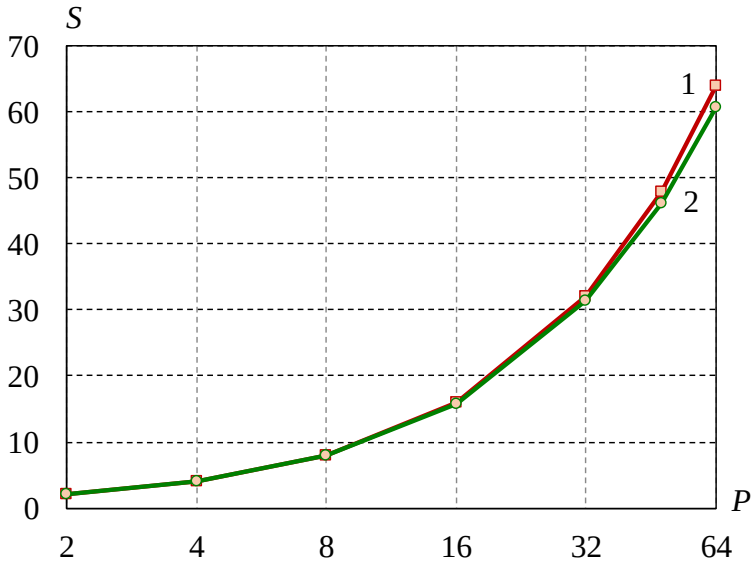

Рис. 4. Зависимость коэффициента ускорения параллельного алгоритма от числа *p* процессов: 1 – линейное ускорение, 2 – ускорение при *n* = 3000.

Литература

1. Эндрюс Г. Основы многопоточного, параллельного и распределенного программирования. – М.: Вильямс, 2003.
2. Кормен, Лайзерсон, Риверст, Штайн. Алгоритмы. Построение и анализ. – М.: Вильямс, 2007.
3. Гергель В. П. Современные языки и технологии параллельного программирования. – М.: Издательство МГУ, 2012.