Курс kiev-clrs – Лекция 21. Параллельные алгоритмы, часть 2

Иван Веселов

2009 г.

Содержание

1 План лекции

- Параллельное умножение матриц
- Анализ работы и длины критического пути
- Параллельная сортировка слиянием

2 Анализ многопоточных программ

Анализ многопоточных программ часто сводится к решению рекурентностей, что обусловлено “разделяй и властвуй”-природой введённой ранее модели многопоточности. Для решения рекурретностей мы будем активно пользоваться “основной теоремой”.

В качестве примеров алгоритмов, которые мы будем распараллеливать – мы рассмотрим умножение матриц и сортировку слиянием.

3 Умножение матриц

Рассмотрим задачу умножения матриц $n \times n$ рекурсивно – через задачу умножения подматриц $n/2 \times n/2$. Не теряя общности, предположим, что n – точная степень двойки.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix} + \begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

Таким образом, чтобы умножить матрицы нам нужно произвести 8 умножений и 4 сложения подматриц вдвое меньшего размера. Запишем алгоритм вспомогательной процедуры Add:

```
ADD(C, T, n)
 1  if n = 1
 2     then C[1,1] ← C[1,1] + T[1,1]
 3          return
 4  разобьём матрицым C и T на подматрицы размера n/2 × n/2
 5  spawn ADD(C11, T11, n/2)
 6  spawn ADD(C12, T12, n/2)
 7  spawn ADD(C21, T21, n/2)
 8  spawn ADD(C22, T22, n/2)
 9  sync
10  return
```

и используем её в Mult:

```
MULT(C, A, B, n)
 1  if n = 1
 2     then C[1,1] ← A[1,1]Ḃ[1,1]
 3          return
 4  выделим место для матрицы T[1..n, 1..n]
 5  разобьём матрицы C и T на подматрицы размера n/2 × n/2
 6  spawn MULT(C11, A11, B11, n/2)
 7  spawn MULT(C12, A11, B12, n/2)
 8  spawn MULT(C21, A21, B11, n/2)
 9  spawn MULT(C22, A21, B12, n/2)
10  spawn MULT(T11, A12, B21, n/2)
11  spawn MULT(T12, A12, B22, n/2)
12  spawn MULT(T21, A22, B21, n/2)
13  spawn MULT(T22, A22, B22, n/2)
14  sync
15  ADD(C, T, n)
16  return
```

То есть для умножения: мы считаем две промежуточные матрицы C и T, а затем складываем их. Все подматрицы матриц C и T могут вычисляться параллельно. Однако нам необходимо выделение места для матрицы T в каждом из рекурсивных вызовов процедуры $Mult$.

Разбиение матрицы на подматрицы занимает константное время, поскольку требует константное число операций вычислений индексов.

Анализ: пусть $M_p(n)$ – время выполнения умножения на p процессорах, а $A_p(n)$ – время выполнения сложения.

$$\begin{aligned} A_1(n) &= 4A_1(n/2) + \Theta(1) \\ &= \Theta(n^2) \end{aligned}$$

То есть рекурсивное выполняется так же, как обычное сложение с двумя вложенными массивами. Этот показатель никак нельзя улучшить, поскольку нам нужно как минимум считать все входные данные, что уже занимает $\Theta(n^2)$

Поскольку мы можем запустить все рекурсивные вызовы параллельно, в длине критического пути учитывается длина только одного из подпутей (максимального, но т.к. они все равны – то любого).

$$
\begin{aligned}
A_\infty(n) &= A_\infty(n/2) + \Theta(1) \\
&= \Theta(\lg n)
\end{aligned}
$$

Теперь проделаем тот же анализ с процедурой Mult:

$$
\begin{aligned}
M_1(n) &= 8M_1(n/2) + A_1(n) \\
&= 8M_1(n/2) + \Theta(n^2) \\
&= \Theta(n^3) \\
M_\infty(n) &= M_\infty(n/2) + A_\infty(n) \\
&= M_\infty(n/2) + \Theta(\lg n) \\
&= \Theta(\lg^2 n)
\end{aligned}
$$

Таким образом, параллелизм процедуры Mult:

$$
\bar{P} = M_1(n)/M_\infty(n) = \Theta(n^3/\lg^2 n)
$$

При $n = 1000, \bar{P} \approx 1000^3/10^2 = 10^7$, что заведомо превышает количество процессоров в компьютерах.

Мы можем улучшить нашу процедуру, уменьшив её параллелизм, но убрав временную матрицу и выделение памяти на неё. То есть мы пожертвуем части параллелизма для уменьшения выделяемой памяти.

```
MULT-ADD(C, A, B, n)
 1  if n = 1
 2     then C[1,1] ← C[1,1] + A[1,1]Ḃ[1,1]
 3          return
 4  разобьём матрицы A, B и C на подматрицы размера n/2 × n/2
 5  spawn MULT-ADD(C11, A11, B11, n/2)
 6  spawn MULT-ADD(C12, A11, B12, n/2)
 7  spawn MULT-ADD(C21, A21, B11, n/2)
 8  spawn MULT-ADD(C22, A21, B12, n/2)
 9  sync
10  spawn MULT-ADD(C11, A12, B21, n/2)
11  spawn MULT-ADD(C12, A12, B22, n/2)
12  spawn MULT-ADD(C21, A22, B21, n/2)
13  spawn MULT-ADD(C22, A22, B22, n/2)
14  sync
15  return
```

Таким образом мы два раза используем матрицу C для аккумулирования результата. При этом нужна синхронизирующая операция между двумя обновлениями.

$$\begin{aligned} MA_1(n) &= \Theta(n^3) \\ MA_\infty(n) &= 2MA_\infty(n/2) + \Theta(1) \\ &= \Theta(n) \end{aligned}$$

$$\bar{P} = MA_1(n)/MA_\infty(n) = \Theta(n^3/n) = \Theta(n^2)$$

При $n = 1000, \bar{P} = 10^6$, однако этого достаточно, к тому же алгоритм часто работает быстрее предыдущего, т.к. не тратик так много времени на выделение памяти.

Оказывается, что можно добиться сокращения длины критического пути до $\lg n$, но мы оставим рассмотрение этого вопроса в качестве упражнения.

4 Параллельная сортировка слиянием

Хорошо поддаются распараллеливанию быстрая сортировка и сортировка слиянием. Мы рассмотрим сортировку слиянием из-за её более простого анализа. Псевдокод:

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          spawn MERGE-SORT(A, p, q)
4          spawn MERGE-SORT(A, q + 1, r)
5  sync
6  MERGE(A, p, q, r)
7  return
```

Анализ:

$$\begin{aligned} M_1(n) &= 2M_1(n/2) + \Theta(n) = \Theta(n \lg n) \\ M_\infty(n) &= M_\infty(n/2) + \Theta(n) = \Theta(n) \\ \bar{P} &= \Theta(n \lg n/n) = \Theta(\lg n) \end{aligned}$$

Небольшой параллелизм. Проблема в монолитной функции Merge, которую тоже следет распараллелить.

Идея распараллеливания показана на рисунке.

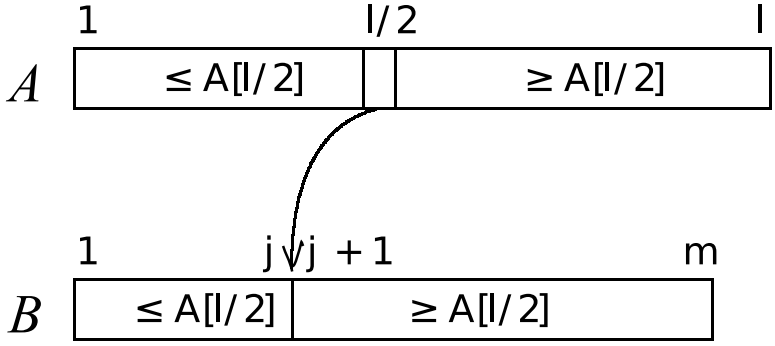

Рис. 1: Распараллеливание Merge

Для объединения двух массивов (они могут быть даже разной длины), берём больший из них, находим его медиану (т.к. массив отсортирован – это будет простым вычислением индекса, константная операция). Затем с помощью бинарного поиска ищем во втором (меньшем) массиве то место, на котором должна находиться найденная медиана. После этого мы получим во втором массиве слева – элементы меньшие медианы, справа – большие. Соответственно теперь задачу можно разделить на две параллельные составляющие – объединение левых частей массивов и правых. А затем просто сконкатенировать результаты.

P-MERGE($A[1\,..\,l], B[1\,..\,m], C[1\,..\,n]$

1 //объединяем A и B в C, длина $n = l + m$
2 //не теряя общности предполагаем, что $l > m$
3 рассматриваем базовый случай
4 ищем такое j, что $B[j] \leqslant A[l/2] \leqslant B[j+1]$ двоичным поиском
5 **spawn** P-MERGE($A[1\,..\,(l/2)], B[1\,..\,j], C[1\,..\,(l/2+j)]$))
6 **spawn** P-MERGE($A[(l/2+1)\,..\,l], B[(j+1)\,..\,m], C[(l/2+j+1)\,..\,n]$)
7 **sync**
8 **return**

Анализ:

Оценим в начале длину критического пути. Поскольку мы используем больший массив для нахождения медианы, получается, что как минимум четверть элементов будет находиться в меньшей из подзадач. Таким образом мы достигаем константного, ограниченного снизу разделения на подзадачи, что позволяет нам сделать оценки. Таким образом,

размер большей подзадачи ограничен $3n/4$, что мы и используем в рекуррентности. Бинарный поиск ограничен $\Theta(\lg n)$:

$$\begin{aligned} PM_\infty &= PM_\infty(3n/4) + \Theta(\lg n) \\ &= \Theta(\lg^2 n) \end{aligned}$$

Теперь оценим работу. Каждый раз задача разбивается на две подзадачи, пусть коэффициент разбиения равен α, тогда их размер равен αn и $(1-\alpha)n$, где $1/4 \leqslant \alpha \leqslant 3/4$. Тогда работа равна:

$$PM_1 = PM_1(\alpha n) + PM_1((1-\alpha)n) + \Theta(\lg n)$$

Покажем, что работа $PM_1 = \Theta(n)$ с помощью метода подстановки.

Индуктивно предположим, что $PM_1(n) \leqslant an - b\lg n$ для неких констант $a, b > 0$.

$$\begin{aligned} PM_1(n) &\leqslant a\alpha n - b\lg(\alpha n) + \alpha(1-\alpha)n - b\lg((1-\alpha)n) + \Theta(\lg n) \\ &= an - b(\lg(\alpha n) + \lg((1-\alpha)n)) + \Theta(\lg n) \\ &= an - b(\lg\alpha + \lg n + \lg(1-\alpha) + \lg n) + \Theta(n) \\ &= an - b\lg n - (b(\lg n + \lg(\alpha(1-\alpha))) - \Theta(\lg n)) \\ &\leqslant an - b\lg n \end{aligned}$$

Так как мы можем выбрать b достаточно большим для того, чтобы $b(\lg n + \lg(\alpha(1-\alpha)))$ доминировало надо $\Theta(\lg n)$. Также мы можем выбрать a достаточно большим, чтобы удовлетворить неравенство в базовом случае. Таким образом, получили, что работа линейна, как и в случае с непараллельным алгоритмом (однако константы, несомненно, больше).

Проведём анализ сортировки слиянием с использованием процедуры параллельного слияния:

$$\begin{aligned} M_\infty(n) &= M_\infty(n/2) + \Theta(\lg^2 n) = \Theta(\lg^3 n) \\ \bar{P} &= \Theta(n\lg n/\lg^3 n) = \Theta(n/\lg^2 n) \end{aligned}$$

что значительно лучше предыдущего алгоритма.

Наилучшее же на сегодня решение обладает параллелизмом $\Theta(n/\lg n)$