

1 Умножение матриц

1.1 Постановка задачи

Реализация алгоритма умножения двух матриц с использованием POSIX threads: необходимо вычислить произведение двух квадратных матриц A и B размера n : в результате чего получается матрица $C = A \times B$, $c_{i,j} = \sum_{k=1}^n a_{i,k} \cdot b_{k,j}$, где $c_{i,j}$ – элементы матрицы C .

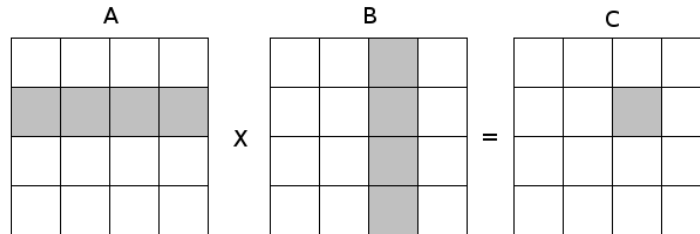
1.2 Подход к распараллеливанию

1.2.1 Простая реализация

Статически распределяем задачу между потоками: при запуске каждый поток в качестве аргумента принимает интервал строк $[start_i; end_i)$ ($i = 0 \dots T-1$, $end_i = start_{i+1}$ для $i = 0 \dots T-2$, где T - количество потоков) результирующей матрицы, которые ему надо посчитать. Т. к. все потоки вычисляют различные элементы матрицы C , то синхронизация не требуется.

1.2.2 Блочное умножение

Матрицы A , B , C разбиваются на одинаковые квадратные (по количеству элементов) блоки размера, кратного размеру матриц. Пусть размер блока равен $bsize$, обозначим $bcount = \frac{n}{bsize}$, тогда каждая матрица разбита на $bcount \cdot bcount$ блоков. Блок (i, j) матрицы C вычисляется: $BC_{i,j} = \sum_{k=1}^{bcount} BA_{i,k} \cdot BB_{k,j}$, где умножение блоков $BA_{i,k} \times BB_{k,j} = BT$, элементы BT : $bt_{i,j} = \sum_{k=1}^{bcount} ba_{i,k} \cdot bb_{k,j}$.



Как и в предыдущем подходе, распределяем между потоками осуществлялось статически: занумеруем все блоки, при запуске каждый поток в качестве аргумента принимает интервал блоков $[start_i; end_i)$ ($i = 0 \dots bcount \cdot bcount - 1$, $end_i = start_{i+1}$ для $i = 0 \dots bcount \cdot bcount - 2$, где T - количество потоков) результирующей матрицы, которые ему надо посчитать. Как и в простой реализации, все потоки вычисляют различные элементы матрицы C и синхронизация не требуется.

1.2.3 Наибольший параллелизм

Из определения умножения следует, что для вычисления матрицы C необходимо произвести n^3 умножений. Распределим умножения между всеми потоками: занумеруем все умножения числами от 0 до $n^3 - 1$ и аналогично предыдущим реализациям будем в каждый поток передавать интервал $[start_i; end_i)$ – номера умножений, которые надо посчитать.

При данном подходе разные потоки могут считать одни элементы матрицы C , поэтому для каждого из них создаём отдельную матрицу C_i , т. о. синхронизация по-прежнему не требуется. Итоговая матрица $C = \sum_{i=0}^{T-1} C_i$.

1.3 Результаты тестирования

Измерение времени осуществлялось с помощью системной команды `time`. Размер матрицы $n = 2000$.

Реализация	Кол-во потоков	Время		
		обычное	блочное (100)	необычное
Однопоточная		19.209	20.520	-
Многопоточная	1	19.086	20.458	19.117
Многопоточная	2	10.656	11.275	10.771
Многопоточная	3	7.887	8.241	8.031
Многопоточная	4	6.913	6.703	7.306
Многопоточная	5	7.858	7.641	8.041
Многопоточная	6	7.174	6.780	7.366
Многопоточная	7	7.171	6.755	7.360
Многопоточная	8	7.164	6.765	7.353
Многопоточная	9	7.277	6.871	7.503
Многопоточная	10	7.169	6.784	7.419

2 LU-разложение

2.1 Постановка задачи

Известно, что квадратную матрицу A размера n можно представить как произведение двух матриц $L \cdot U$ размера $n \times n$, где L – нижняя треугольная матрица, U – верхняя треугольная матрица. Элементы $l_{i,j}$ и $u_{i,j}$ соответственно матриц L и U могут быть вычислены по формулам:

$$u_{0,j} = a_{0,j}, \quad 0 \leq j < n$$

$$u_{i,j} = a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,j}, \quad i \leq j < n$$

$$l_{i,i} = 1$$

$$l_{i,0} = \frac{a_{i,0}}{u_{0,0}}, \quad 1 \leq j < n$$

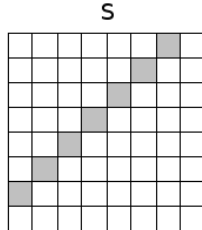
$$l_{i,j} = \frac{1}{u_{j,j}} \left(a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,j} \right), \quad i+1 \leq j < n$$

2.2 Подход к распараллеливанию

$$\text{Рассмотрим матрицу } S = L + U - E = \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & \cdots & u_{0,n-1} \\ l_{1,0} & u_{1,1} & u_{1,2} & \cdots & \vdots \\ l_{2,0} & l_{2,1} & u_{2,2} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{n-2,n-1} \\ l_{n-1,0} & \cdots & \cdots & l_{n-1,n-2} & u_{n-1,n-1} \end{pmatrix} = (s_{i,j}),$$

где E – единичная матрица.

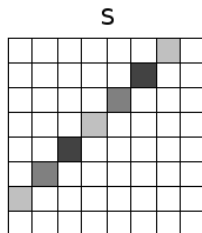
Для вычисления $s_{i,j}$ требуются элементы $s_{u,j}$, $0 \leq u < \min(i,j)$ и $s_{i,v}$, $0 \leq v < \min(i,j)$, т. е. $s_{i,j}$ зависит только от тех элементов, которые находятся выше или левее в матрице S . Элементы $s_{i,k-i}$, расположенные на диагонали, не зависят друг от друга:



Идея алгоритма распараллеливания заключается в том, чтобы последовательно вычислять диагонали. При вычислении очередной диагонали каждый поток:

1. На основании `thread_id` определяет, какие элементы он будет вычислять.
2. Вычисляет их.
3. Ожидает, когда другие потоки закончат вычисление своих элементов данной диагонали.

Элементы диагонали, которые будет вычислять, поток, определяются чередованием: при $tcount$ потоков на k -й диагонали поток tid будет вычислять элементы $s_{i-tid-i \cdot tcount, k+tid+i \cdot tcount}$, $i \leq 0$:



Синхронизация осуществляется с помощью `pthread_barrier`.

2.3 Результаты тестирования

Измерение времени осуществлялось с помощью встроенной в `bash` утилиты `time`. Размер матрицы $n = 2000$.

Однопоточная		54.457
Многopоточная	1	63.709
Многopоточная	2	36.795
Многopоточная	3	28.863
Многopоточная	4	25.877
Многopоточная	5	36.095
Многopоточная	6	34.460
Многopоточная	7	35.333
Многopоточная	8	35.748
Многopоточная	9	41.486
Многopоточная	10	42.567

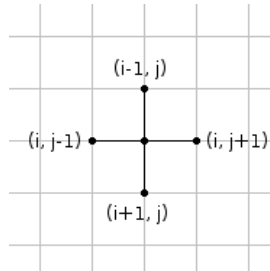
3 Решение уравнения Пуассона методом “крест”

3.1 Постановка задачи

Рассмотрим двумерную сетку, значение в узле с координатами (i, j) обозначим через $a_{i,j}$.

На каждой итерации пересчитываем значение в узлах по формуле:

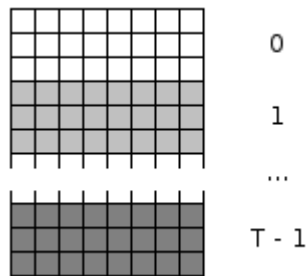
$$a_{i,j}^{k+1} = \frac{a_{i-1,j}^k + a_{i,j+1}^k + a_{i+1,j}^k + a_{i,j-1}^k + a_{i,j}^k}{5}$$



3.2 Подход к распараллеливанию

3.2.1 Деление по строкам

Статически распределяем задачу между потоками: при запуске каждый поток в качестве аргумента принимает интервал строк $[start_i; end_i)$ ($i = 0 \dots T-1, end_i = start_{i+1}$ для $i = 0 \dots T-2$, где T - количество потоков) матрицы, которые ему надо посчитать. Т. к. каждому потоку для вычисления $(k+1)$ -й итерации необходимо иметь значения, полученные соседними потоками после k -й итерации, то синхронизация осуществляется после каждой итерации: с помощью `pthread_barrier` все потоки дожидаются друг друга.



3.2.2 Очередь элементов

Рассмотрим i -ю строку: для вычисления $(k+1)$ -й итерации необходимо иметь значения для $(i-1)$ -й и $(i+1)$ -й строк, полученных на k -й итерации, при этом от остальных строк i -ая никак не зависит.

Воспользуемся этим фактом, будем хранить список строк, для которых может быть посчитана очередная итерация. Поток, посчитав очередную строку, определяет новые строки, которые теперь могут быть посчитаны, и добавляет их в список.

Список строк просто реализуется с помощью очереди: новые строки добавляем в конец, строки для вычисления берём из начала очереди. Синхронизация требуется только при доступе к очереди и может быть реализована с помощью `pthread_mutex`.

3.3 Результаты тестирования

Измерение времени осуществлялось с помощью системной команды `time`. Размер матрицы $n = 2000$, количество итераций – 1000.

Время работы однопоточной реализации: 160 с.

Реализация	Кол-во потоков	Время (в секундах)	
		по строкам	очередь
Многопоточная	1	160	161
Многопоточная	2	87	88
Многопоточная	3	62	64
Многопоточная	4	51	52
Многопоточная	5	65	52
Многопоточная	6	58	52
Многопоточная	7	56	52
Многопоточная	8	52	52
Многопоточная	9	58	52
Многопоточная	10	56	52