Умножение матриц. Алгоритм Штрассена

Шипицына Алина, 716 группа

Обычный алгоритм перемножения матриц размера n работает за время $O(n^3)$.
Алгоритм Штрассена умножает матрицы за время $O(n^{log_2 7}) = O(n^{2.81})$

Алгоритм

Пусть A, B — две квадратные матрицы над кольцом R. Матрица *C* получается по формуле:
$C = AB$, $A, B, C \in R^{2^n * 2^n}$

Разделим матрицы *A*, *B* и *C* на равные по размеру блочные матрицы:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

тогда:

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\
\mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\
\mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}
\end{aligned}$$

Однако Количество умножений все-равно 8.

Введем новые элементы:

$$\begin{aligned}
\mathbf{P}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{P}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{P}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{P}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{P}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{P}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{P}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

которые затем используются для выражения $C_{i,j}$. Таким образом, нам нужно всего 7 умножений на каждом этапе рекурсии. Элементы матрицы *C* выражаются из P_k по формулам:

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{P}_1 + \mathbf{P}_4 - \mathbf{P}_5 + \mathbf{P}_7 \\
\mathbf{C}_{1,2} &= \mathbf{P}_3 + \mathbf{P}_5 \\
\mathbf{C}_{2,1} &= \mathbf{P}_2 + \mathbf{P}_4 \\
\mathbf{C}_{2,2} &= \mathbf{P}_1 - \mathbf{P}_2 + \mathbf{P}_3 + \mathbf{P}_6
\end{aligned}$$

Рекурсивный процесс продолжается *n* раз, до тех пор пока размер матриц $C_{i,j}$ не станет достаточно малым, далее используют обычный метод умножения матриц. Это делают из-за того, что алгоритм Штрассена теряет эффективность по сравнению с обычным на малых матрицах в

силу большего числа сложений. Оптимальный размер матриц для перехода к обычному умножению зависит от характеристик процессора и языка программирования и на практике лежит в пределах от 32 до 128. В своей работе я использовала размерность 32 при переходе к обычному умножению.

Замер времени для сравнения метода Штрассена и обычного метода без распараллеливания

Размер матриц $2^{10} = 1024$

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel normal method: 12.931174
Time for non-parallel: 6.601889
```

Видно, что метод Штрассена работает быстрее

Алгоритм распараллеливания метода Штрассена

Я выделила 4 группы операций, в каждой из которых все операции могут быть выполнены параллельно:

$$
\begin{aligned}
T_1 &= A_{0,0} + A_{1,1} \\
T_2 &= B_{0,0} + B_{1,1} \\
T_3 &= A_{1,0} + A_{1,1} \\
T_4 &= B_{0,1} - B_{1,1} \\
T_5 &= B_{1,0} - B_{0,0} \\
T_6 &= A_{0,0} + A_{0,1} \\
T_7 &= A_{1,0} - A_{0,0} \\
T_8 &= B_{0,0} + B_{0,1} \\
T_9 &= A_{0,1} - A_{1,1} \\
T_{10} &= B_{1,0} + B_{1,1}
\end{aligned}
$$

$$
\begin{aligned}
S_1 &= T_1 \cdot T_2 \\
S_2 &= T_3 \cdot B_{0,0} \\
S_3 &= A_{0,0} \cdot T_4 \\
S_4 &= A_{1,1} \cdot T_5 \\
S_5 &= T_6 \cdot B_{1,1} \\
S_6 &= T_7 \cdot T_8 \\
S_7 &= T_9 \cdot T_{10}
\end{aligned}
$$

$$
\begin{aligned}
T_{11} &= S_1 + S_4 \\
T_{12} &= S_2 + S_4 \\
T_{13} &= S_3 + S_6 \\
T_{14} &= S_7 - S_5 \\
T_{16} &= S_3 + S_5 \\
T_{17} &= S_1 - S_2
\end{aligned}
$$

$$
\begin{aligned}
T_{15} &= T_{11} + T_{14} \\
T_{18} &= T_{13} + T_{17}
\end{aligned}
$$

Для реализации распараллеливания я использовала механизм пула задач (omp task), а синхронизацию проводила с помощью omp taskwait. Так как это долгая операция, я объединила первые 2 группы в одну, чтобы было меньше точек синхронизации.

Результаты

• Сначала посмотрим на зависимость скорости от количества потоков. Размер матриц $2^{10} = 1024$.

Num_threads = 2

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel normal method: 14.642783
Time for non-parallel: 6.584451
Time for parallel: 3.864555
```

Num_threads = 3

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 6.555515
Time for parallel: 2.932458
```

Num_threads = 4

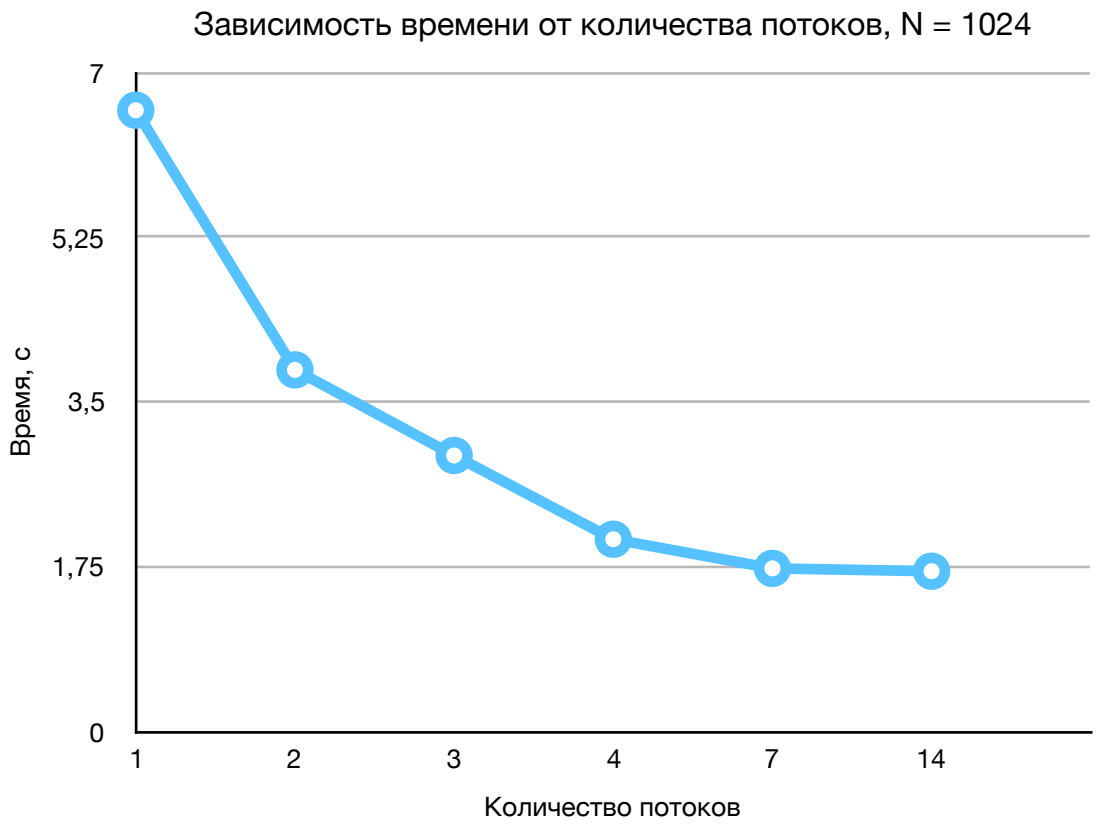
```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 6.655621
Time for parallel: 2.047013
```

Num_threads = 7

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 6.582964
Time for parallel: 1.734784
```

Num_threads = 14

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 6.658475
Time for parallel: 1.736449
```



• Посмотрим на зависимость времени от размера матриц для 7 потоков

N = 128

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 0.005790
Time for parallel: 0.002020
```

N = 256

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 0.041199
Time for parallel: 0.012195
```

N = 512

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 0.291364
Time for parallel: 0.080202
```

N = 1024

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 2.036309
Time for parallel: 0.543608
```

N = 2048

```
(base) alina@MacBook-Pro-Alina Парпрог % ./matrix_omp
Time for non-parallel: 14.477698
Time for parallel: 3.683835
```