

实现方式

总共使用了三个 GPU kernel——selfKernel, crossKernel, globalKernel, 分别表示参考优化方法的三个阶段。

三个 kernel 的 Block 大小均为 $b \times b$, 即分块的大小。(此处取 $b = 32$, 即达到刚好最大线程数 1024)

selfKernel 中包含 1×1 个线程块, crossKernel 中包含 $\lceil \frac{n}{b} \rceil \times 2$ 个线程块 (除去与当前块编号相同的, 分别表示十字块的行和列), globalKernel 中包含 $\lceil \frac{n}{b} \rceil \times \lceil \frac{n}{b} \rceil$ 个线程块 (表示剩余块)。

在 selfKernel 中使用一个 $b \times b$ 大小的共享内存数组, 用于存储计算过程中的临时值。

在 crossKernel 中使用两个 $b \times b$ 大小的共享内存数组, 用于临时存储当前十字架中块的值和当前中心块的值。

在 globalKernel 中使用两个 $b \times b$ 大小的共享内存数组, 用于临时存储更新当前块所需两个十字架中块的值。

具体实现如下:

```
1  __global__ void kernel(int n, int k, int *graph) {
2      auto i = blockIdx.y * blockDim.y + threadIdx.y;
3      auto j = blockIdx.x * blockDim.x + threadIdx.x;
4      if (i < n && j < n) {
5          graph[i * n + j] = min(graph[i * n + j], graph[i * n + k] + graph[k * n +
6      j]);
7      }
8  }
9  __global__ void selfKernel(int n, int p, int *graph) {
10     __shared__ int sharedGraph[b][b];
11     const auto i = threadIdx.y;
12     const auto j = threadIdx.x;
13     const auto realI = b * p + i;
14     const auto realJ = b * p + j;
15     const auto id = realI * n + realJ;
16     if (realI < n && realJ < n) {
17         sharedGraph[i][j] = graph[id];
18     } else {
19         sharedGraph[i][j] = 1000000000;
20     }
21     __syncthreads();
22     int newDis;
23     for (int k = 0; k < b; ++k) {
24         newDis = sharedGraph[i][k] + sharedGraph[k][j];
25         __syncthreads();
26         if (newDis < sharedGraph[i][j]) {
```

```

27         sharedGraph[i][j] = newDis;
28     }
29     __syncthreads();
30 }
31 if (realI < n && realJ < n) {
32     graph[id] = sharedGraph[i][j];
33 }
34 }
35
36 __global__ void crossKernel(int n, int p, int *graph) {
37     if (blockIdx.x == p) return;
38     __shared__ int sharedGraph[b][b];
39     const auto i = threadIdx.y;
40     const auto j = threadIdx.x;
41     auto realI1 = b * p + i;
42     auto realJ1 = b * p + j;
43     const auto id1 = realI1 * n + realJ1;
44     if (realI1 < n && realJ1 < n) {
45         sharedGraph[i][j] = graph[id1];
46     } else {
47         sharedGraph[i][j] = 1000000000;
48     }
49     __syncthreads();
50     auto realI2 = realI1;
51     auto realJ2 = realJ1;
52     if (blockIdx.y == 0) {
53         realI2 = b * blockIdx.x + i;
54     } else {
55         realJ2 = b * blockIdx.x + j;
56     }
57     __shared__ int sharedNewGraph[b][b];
58     const auto id2 = realI2 * n + realJ2;
59     int minDis;
60     if (realI2 < n && realJ2 < n) {
61         minDis = sharedNewGraph[i][j] = graph[id2];
62     } else {
63         minDis = sharedNewGraph[i][j] = 1000000000;
64     }
65     __syncthreads();
66     int newDis;
67     if (blockIdx.y == 0) {
68         for (int k = 0; k < b; ++k) {
69             newDis = sharedNewGraph[i][k] + sharedGraph[k][j];
70             if (newDis < minDis) {
71                 minDis = newDis;
72             }
73         }
74         __syncthreads();
75         sharedNewGraph[i][j] = minDis;

```

```

75         __syncthreads();
76     }
77 } else {
78     for (int k = 0; k < b; ++k) {
79         newDis = sharedGraph[i][k] + sharedNewGraph[k][j];
80         if (newDis < minDis) {
81             minDis = newDis;
82         }
83         __syncthreads();
84         sharedNewGraph[i][j] = minDis;
85         __syncthreads();
86     }
87
88 }
89 if (realI2 < n && realJ2 < n) {
90     graph[id2] = sharedNewGraph[i][j];
91 }
92 }
93
94 __global__ void globalKernel(int n, int p, int *graph) {
95     if (blockIdx.x == p || blockIdx.y == p) return;
96     __shared__ int sharedRowGraph[b][b], sharedColGraph[b][b];
97     const auto i = threadIdx.y;
98     const auto j = threadIdx.x;
99     auto realI1 = b * p + i;
100    auto realJ1 = b * p + j;
101    const auto id1 = realI1 * n + realJ1;
102    auto realI2 = b * blockIdx.y + i;
103    auto realJ2 = b * blockIdx.x + j;
104    const auto id2 = realI2 * n + realJ2;
105    if (realI1 < n && realJ2 < n) {
106        sharedRowGraph[i][j] = graph[realI1 * n + realJ2];
107    } else {
108        sharedRowGraph[i][j] = 1000000000;
109    }
110    if (realI2 < n && realJ1 < n) {
111        sharedColGraph[i][j] = graph[realI2 * n + realJ1];
112    } else {
113        sharedColGraph[i][j] = 1000000000;
114    }
115    __syncthreads();
116    if (realI2 < n && realJ2 < n) {
117        int minDis = graph[id2], newDis;
118        for (int k = 0; k < b; ++k) {
119            newDis = sharedColGraph[i][k] + sharedRowGraph[k][j];
120            if (newDis < minDis) {
121                minDis = newDis;
122            }

```

```
123     }
124     graph[id2] = minDis;
125 }
126 }
127
```

不同图规模加速比

图规模 n	朴素实现运行时间 (ms)	运行时间 (ms)	加速比
1000	14.756778	2.970092	4.97
2500	377.112280	25.343865	14.88
5000	2970.998835	158.762596	18.71
7500	10013.331400	515.264827	19.43
10000	22616.127009	1195.623236	18.91