**PerfLab实验报告**

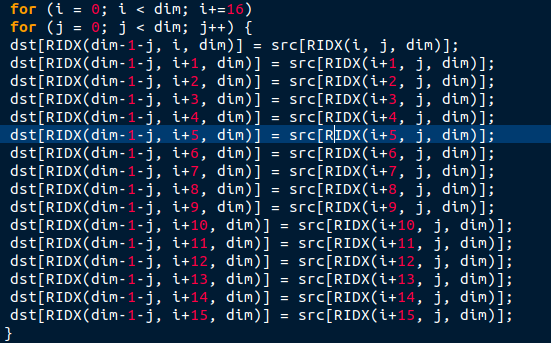
**原来的分数：**



**优化思路：**

Rotate函数：

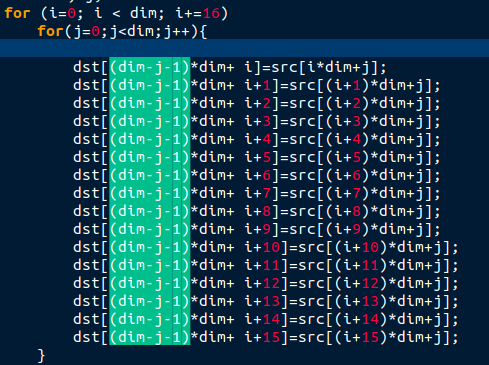
第一步：将函数1×16循环展开，将i展开，即每次循环处理16排：





函数性能大幅度提升。

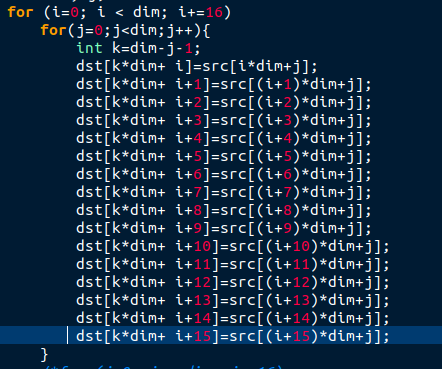
第二步：发现多次大量调用RIDX函数，将该函数展开：





函数性能略微提升。

第三步：函数中多次计算dim-j-1，将其用变量k替换：





函数性能不变，得到最终版本。

思考，如果将j展开，即每次循环处理16列会如何：





发现优化结果不如将i展开好。

Smooth函数：

第一步：由于原版本计算每个元素时都要重新计算该元素所在九宫格的像素值之和，计算量很大，所以改写为处理元素（i，j）时，记录下元素（i，j）所在九宫格的像素值之和为sum，然后在处理元素（i，j+1）时，将sum减去（i-1，j-1）、（i，j-1）、（i+1，j-1）的像素值，然后加上（i-1，j+2）、（i，j+2）、（i+1，j+2）的像素值，这样每次计算能减少3次加法。这样做可以将结果优化到33左右（此版本的代码忘记保存了）

第二步：发现函数中多次调用min、max和RIDX函数，通过改写函数、专门处理边界特殊情况的方法将这三个函数消去，最后的结果提升到了51左右。（此版本的代码忘记保存了）

第三步：发现在循环中会大量调用计算像素和、计算平均值的函数，于是将这些函数全部展开，得到最终版本（见源文件）。

