Assignment 3: OMP

Bar-Ilan CS 89-3312
Parallel Systems Programming

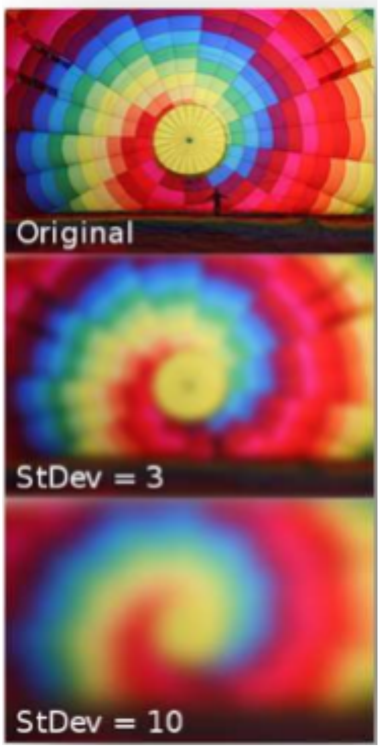
December 25, 2025

1 Gaussian blur algorithm

In this assignment, we will implement a blur algorithm. In particular, we will discuss the Gaussian Blur Algorithm.

Note that the algorithm is well-known and used in real-life; for example, Photoshop offers using Gaussian blur to "polish" photos:
https://www.adobe.com/creativecloud/photography/discover/gaussian-blur.html

In a nutshell, this algorithm blurs an image, using Gaussian function, in order to reduce image noise & details. In fact, the algorithm applies a "Gaussian kernel" to an image. We can control the intensity of the blur, for example:



Without going into too much detail, the algorithm averages the value of each pixel in relation to its neighbors.

You are given a sequential code in C, implementing the Gaussian Blur Algorithm; you are required to parallelize it using OpenMP!

1.1 Notes for parallelization

1. When thinking about how to parallelize the code, bear in mind that it requires thinking about shared resources; in particular, for example, pay attention to accessing the kernel, and maybe synchronization before changing certain pixels?

2. Parallelizing the algorithm doesn't just mean putting pragmas – think about the scheduling, shared resources, false sharing, synchronization, and about any other tools in OpenMP that we have learned in class.

2 Binary Search Tree synchronization

In this part you will implement a small library that provides the user with an integer binary-search tree data structure, with operations that can be used safely in parallel.

You are given the file `binary_tree.h`, and it is your job to implement the functions whose signatures appear in the file. You may choose your own implementation of the `TreeNode` structure, as long as the library functions behave as specified.

All operations always receive the *root* of the tree (i.e., the root of the original tree object) as their first argument. Functions that modify the tree (`insertNode` and `deleteNode`) must return the (possibly updated) root of the tree after the operation.

2.1 `binary_tree.h`

```
#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include <stdbool.h>

// Definition of a binary tree node
typedef struct TreeNode {
    int data;
    struct TreeNode *left;
    struct TreeNode *right;
} TreeNode;

// Function to create a new binary tree node.
// Returns a pointer to the newly allocated node.
TreeNode* createNode(int data);

// Function that insert a new value into the binary search tree.
// Returns a pointer to the new tree.
TreeNode* insertNode(TreeNode* root, int data);

// Function that delete a value from the binary search tree, if it exists.
// Returns a pointer to the new tree.
TreeNode* deleteNode(TreeNode* root, int data);

// Search for a value in the binary search tree.
// Returns true if 'data' is found, false otherwise.
bool searchNode(TreeNode* root, int data);
```