Assignment 3: OMP

Bar-Ilan CS 89-3312
Parallel Systems Programming

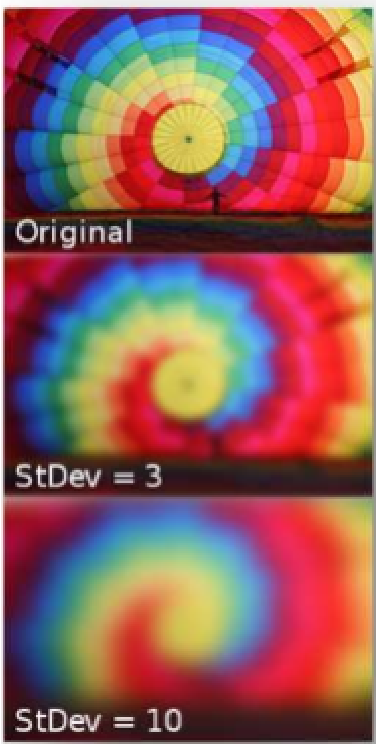
December 11, 2025

EX1 – 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!

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.

EX2 – Binary Search Tree synchronization

In this part you will be implementing a small library that provides the user 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's your job to implement the functions whose signature is in the file. You may choose your own implementation of the `TreeNode` structure, as long as the library functions work the way they are expected.

`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
TreeNode* createNode(int data);

// Function to insert a new node into the binary tree
TreeNode* insertNode(TreeNode* root, int data);

// Function to delete a node from the binary tree
TreeNode* deleteNode(TreeNode* root, int data);

// Function to search for a node in the binary tree
bool searchNode(TreeNode* root, int data);

// Function to find the minimum value node in the binary tree
TreeNode* findMin(TreeNode* root);

// Function to perform an in-order traversal of the binary tree
void inorderTraversal(TreeNode* root);

// Function to perform a pre-order traversal of the binary tree
void preorderTraversal(TreeNode* root);
```

```
// Function to perform a post-order traversal of the binary tree
void postorderTraversal(TreeNode* root);

// Function to free all nodes in the binary tree
void freeTree(TreeNode* root);

#endif // BINARY_TREE_H
```

Parallelism requirements

1. Those functions should not be parallel but should support parallelism aka – synchronized.
2. You need to assume different functions might be triggered on the same tree structures from different threads.
3. For example: `thread1` calls `insert` while `thread2` calls `delete`.
4. Don't just lock the entire functions as `critical`; think which functions depend on each other so that the data they return will be valid.
5. All of the work will be on **OMP**

Example

For example consider the binary tree:

```
        10
       /  \
      5    15
     / \   / \
    3   7 12  18
```

The serial flow will return `true` for `findMin()->value == 2` if:

```
insert(2)
findMin()
// findMin()->value == 2 will return true
```

If thread1 will call `insert(2)` and then another thread will trigger `findMin()` the result should be the same (but not the other way around of course).

The same "edge cases" can happen with `insert` and `delete`.

Submission Guidelines

To reduce likelihood of misunderstandings, please follow these guidelines:

1. Work can either be done individually or in pairs.
2. Submission is through the submit.
3. Make sure that your solution compiles and runs without any errors and warnings on BIU servers.
4. In the first line of every file you submit, write in a comment your id and full name. For example: `/* 123456789 Israela Israeli */`.
5. Not using openMP in the task will result in an automatic 0, sequential code is unacceptable.

Environment & Submission

For this assignment, you must submit your code in a file named `ass3.zip`. This zip file should contain the files `binary_tree.h`, `binary_tree.c`, and the parallel version of the file provided for the Gaussian blur (without the `main` function), i.e., `guassonFilter.c`.

Note: Please submit the files with their exact names:

- `binary_tree.h`
- `binary_tree.c`
- `guassonFilter.c`

If you use different file names, you will receive an automatic grade of 0. If you choose to appeal this and send correct files, 10 points will automatically be deducted from your final score and you will start with 90.