Universitat de Lleida

MEINF

High Performance Computing

HPC Project: Heat Diffusion Equation

First Delivery (OpenMP)

Marc Gaspà Joval
Raul Hidalgo Caballero

April 10, 2025

Contents

1 Main Decisions Taken to Parallelize the Problem

We encountered we SMPD (Single Program Multiple Data) problem which we could parallelize using OpenMP.

This section details each part of the serial code that was identified as suitable for parallelization, along with the strategies employed:

1.1 Grid Initialization (initialize_grid)

1.1.1 Parallelizable Section

The initialization loop sets the temperature for each cell independently.

```
for (i = 0; i < nx; i++) {
        for (j = 0; j < ny; j++) {
                int inyj = i * ny + j;
                if (i == j) { grid[inyj] = 1500.0; }
                else if (i == nx - 1 - j) { grid[inyj] = 1500.0; }
                else { grid[inyj] = 0.0; }
        }
}
```

1.1.2 Strategy

In this case as there aren't any dependencies we used `#pragma omp parallel for` with the `collapse(2)` clause on the nested loops to distribute the iterations.

1.2 Computation of the Heat Equation (solve_heat_equation)

1.2.1 Parallelizable Section

In this case, as one step is dependent of the previous one we cannot parallelize the outer loop. However, the inner loop can parallelize two parts, the first one is the aplication of the heat equation:

```
for (i = 1; i < nx - 1; i++) {
        for (j = 1; j < ny - 1; j++) {
                int inyj = i * ny + j;
                new_grid[inyj] = grid[inyj]
                + r * (grid[(i + 1) * ny + j] + grid[(i - 1) * ny + j] - 2 * grid[inyj])
                + r * (grid[inyj + 1] + grid[inyj - 1] - 2 * grid[inyj]);
        }
}
```

The second one is the application of the boundary conditions:

```
for (i = 0; i < nx; i++) {
                new_grid[0 * ny + i] = 0.0;
                new_grid[ny * (nx - 1) + i] = 0.0;
}

for (j = 0; j < ny; j++) {
                new_grid[0 + j * nx] = 0.0;
                new_grid[(ny - 1) + j * nx] = 0.0;
}
```

1.2.2 Strategy

In this case we really used a similar solution for both cases, as each cell can be calculated independently from the others.
We used `#pragma omp parallel for` with the `collapse(2)` clause on the heat equation loop to distribute the iterations. We defined the variables `i` and `j` as private.

For the boundary conditions we added `#pragma omp parallel for` to the loops that set the values of the boundaries. We also defined the variable `i` as private in the first loop and `j` in the second one.

1.3 Data Dependency Management

1.3.1 Overall Approach

A two-buffer scheme is employed throughout the computational phase. While one buffer holds the original data (read-only), the other is used for storing computed updates.

1.3.2 Synchronization

The swapping of pointers between `grid` and `new_grid` is performed outside parallel regions, ensuring data integrity for the subsequent computation cycle.

1.4 Serial Output Operation

1.4.1 Analysis

The generation and writing of the BMP file require a strict order of operations (bottom-to-top row order and specific padding for each row), which is highly sequential.

1.4.2 Decision

The output phase remains serial due to the order dependency and the relatively small impact on overall performance.

2 Scalability of the Program

2.1 Performance Measurement Methodology

Execution times are measured using `omp_get_wtime()`. Testing is conducted using various grid sizes (matrix sizes) and numbers of iteration steps.

2.2 Performance Metrics and Their Formulas

Matrix Size	**Steps**			
	100	**1000**	**10000**	**100000**
100x100				
1000x1000				
2000x2000				

Table 1: Execution times of the `heat_serial` program.

Matrix Size	**Steps**			
	100	**1000**	**10000**	**100000**
100x100				
1000x1000				
2000x2000				

Table 2: Execution times of the `heat_serial` parallel program.