**Estudiante:** Julio Alejandro Mazo Reyes.

**Materia:** Programación Paralela.

**Parcial 1**

**Video:** <https://youtu.be/Ah-lD1_LopU>

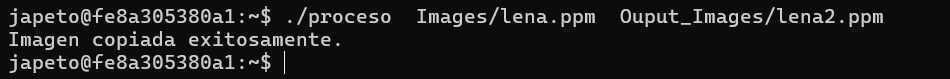
**Repositorio:** <https://github.com/JulioM-108/Parcial_Paralela>

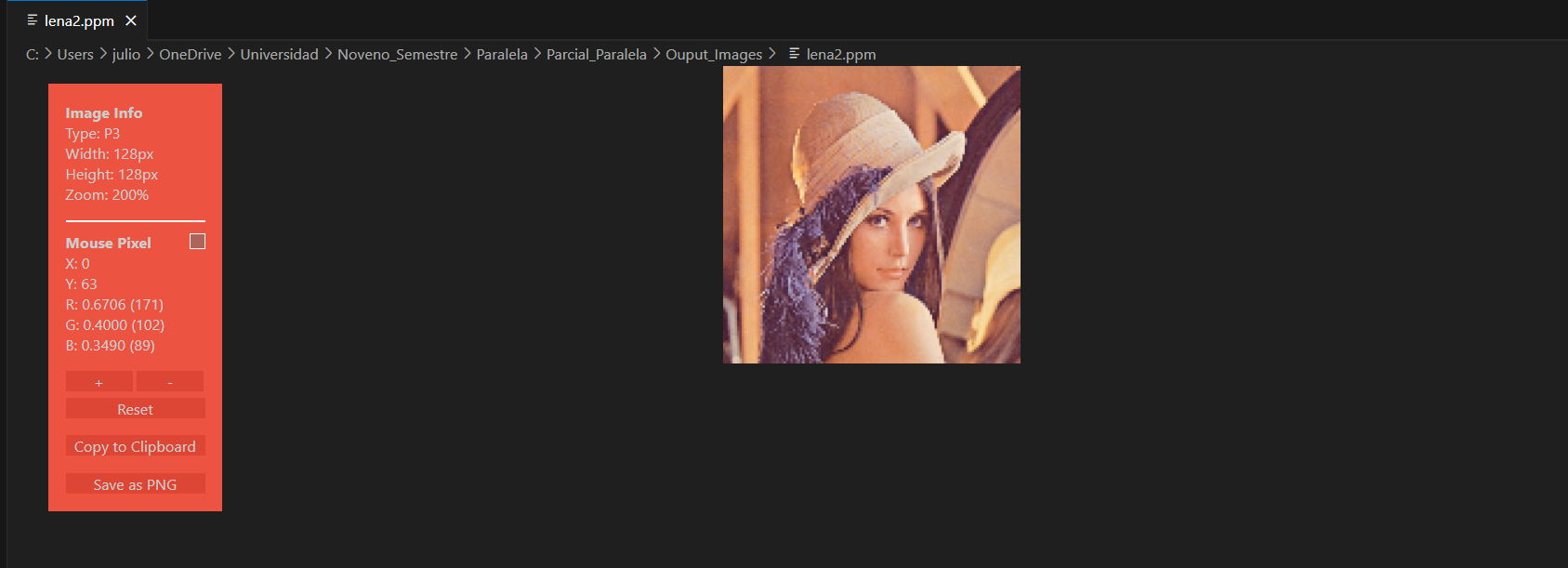
¿Cómo influye la programación paralela, utilizando OpenMP ó Pthreads y MPI (En un entorno simulado con docker), en el tiempo de ejecución del filtrado de imágenes PPM y PGM en comparación con una implementación secuencial?

R: En este punto me pareció que la versión secuencial funciono un poco parecido a la paralela. Por un lado, con Pthreads, los tiempos de CPU andan entre 0.047 y 0.067 segundos, lo cual ya es una mejora notable al dividir el trabajo entre hilos. Por otro lado, con OpenMP, los tiempos para la imagen sulfur.pgm fueron de 0.788 segundos, su gran ventaja (que no fue el tiempo realmente) es que se me hizo más fácil de implementar para paralelizar bucles, lo que agilizo los procesos.

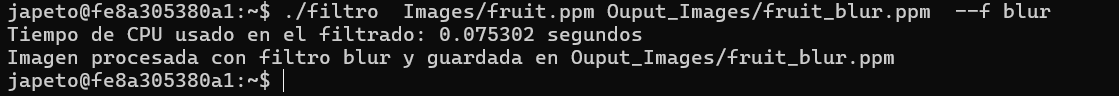
Pero el que realmente me impresionó fue MPI en ese entorno de Docker, con tiempos de 0.023 a 0.024 segundos por nodo. Esto demuestra que al distribuir el trabajo en varios procesos que actúan como si fueran nodos separados, se consigue la mayor reducción de tiempo. En conclusión, todos estos enfoques, Pthreads, OpenMP y MPI, demuestran que la programación paralela es la clave para optimizar el uso de la CPU y la memoria, logrando que el filtrado de imágenes sea mucho más rápido que si se hiciera de forma secuencial.

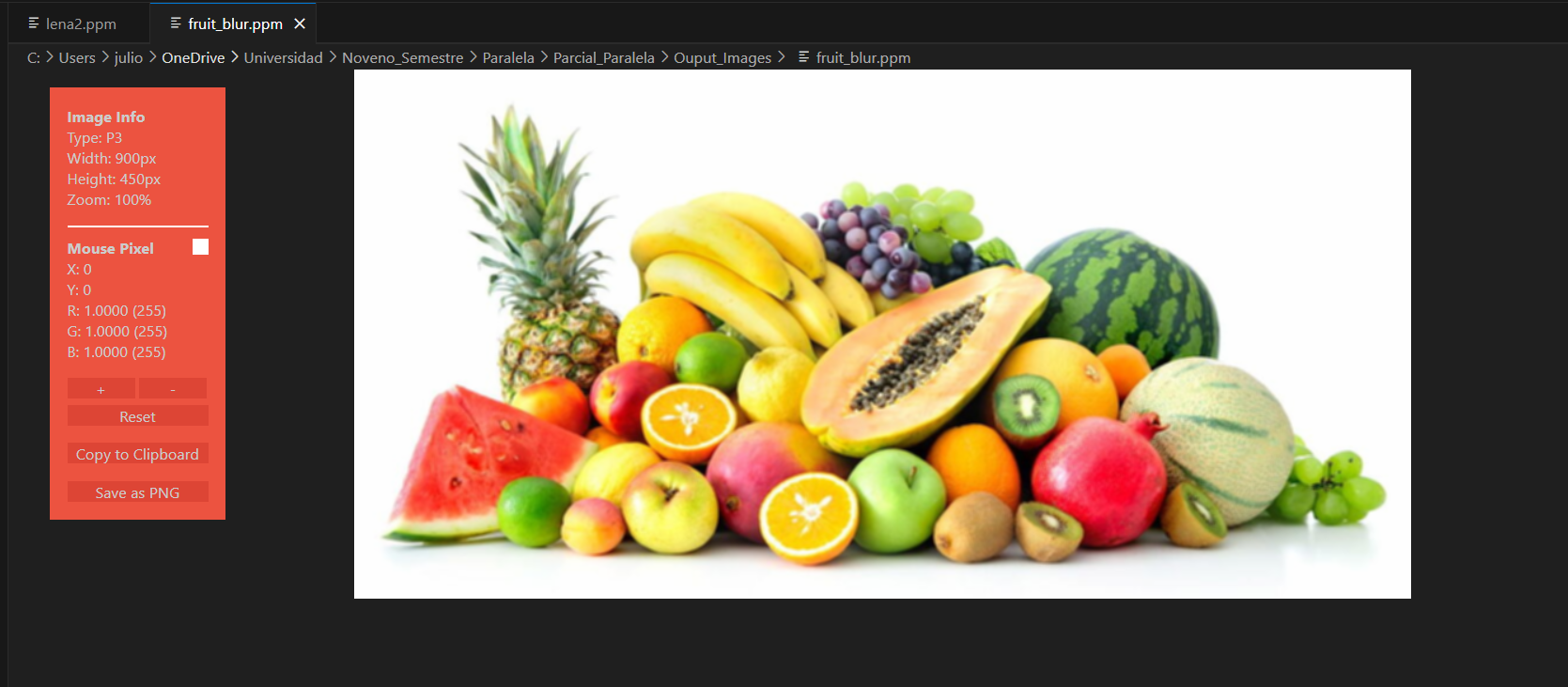
Versión Básica de Procesador:

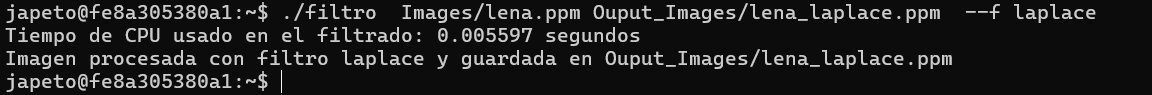


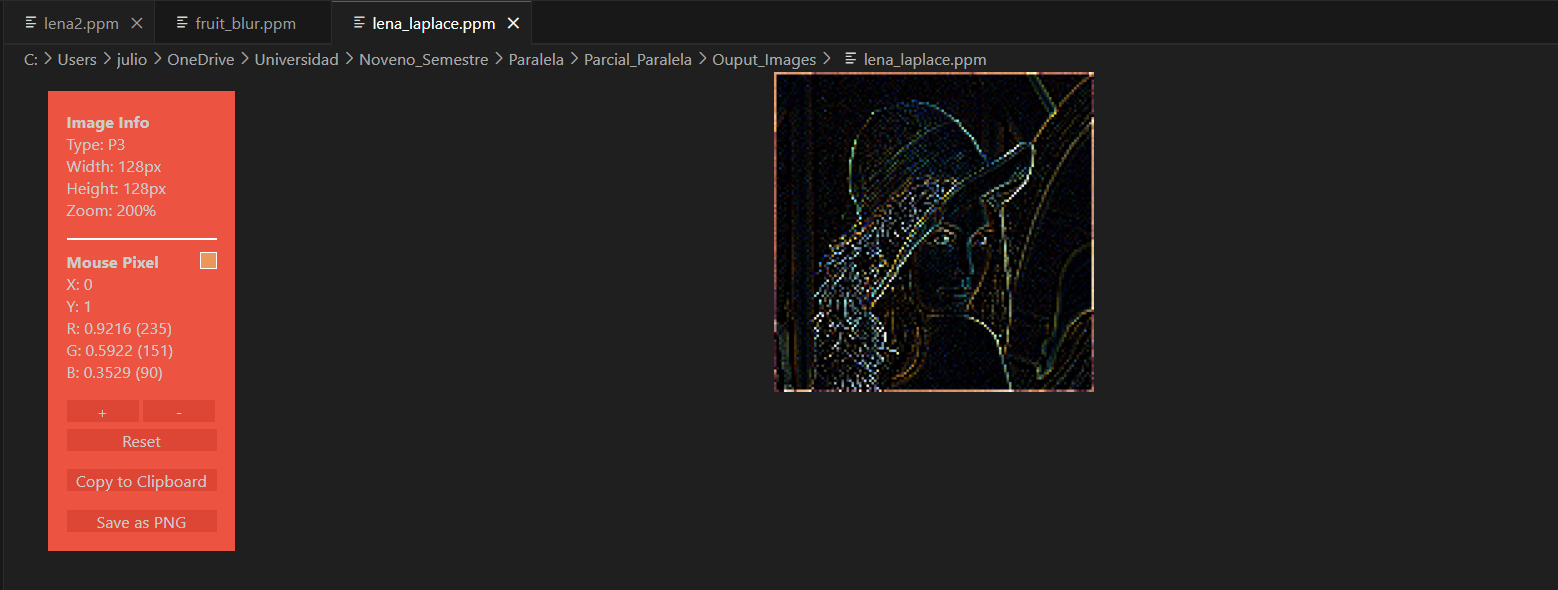


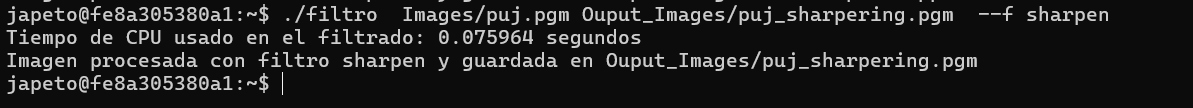
Versión Secuencial:

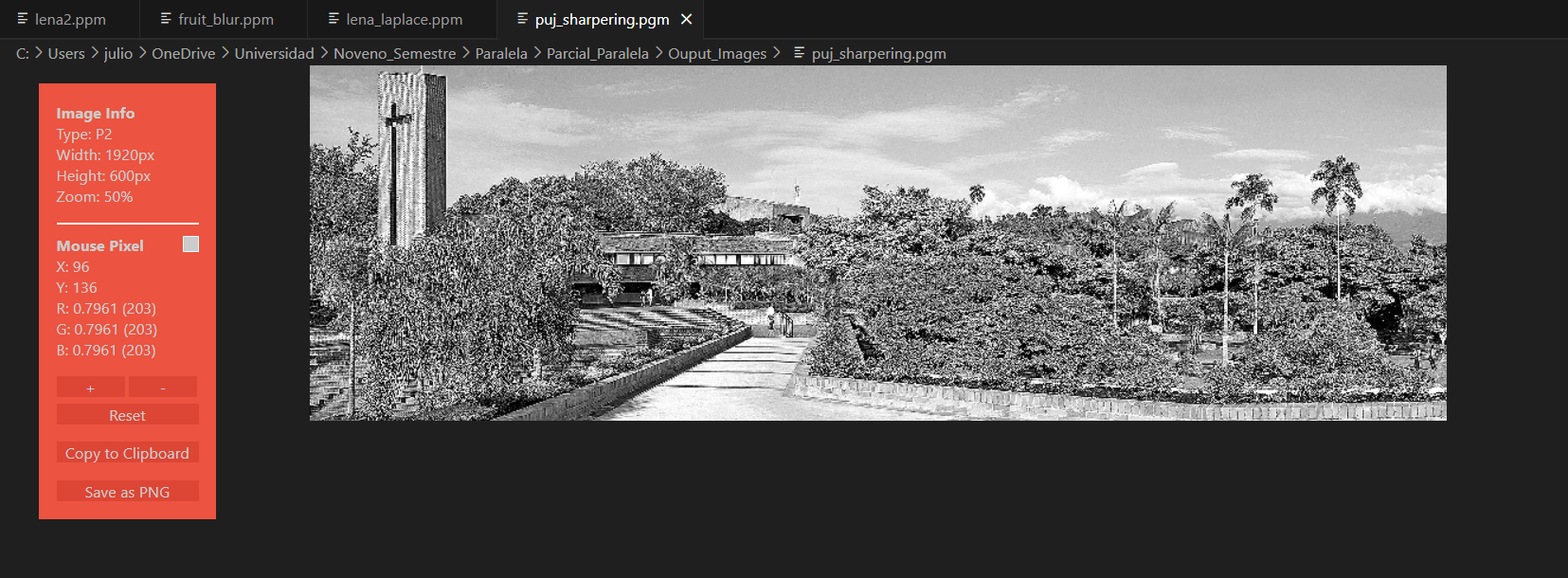




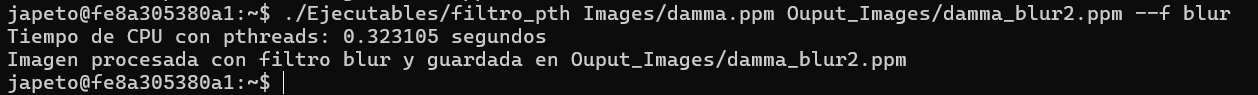


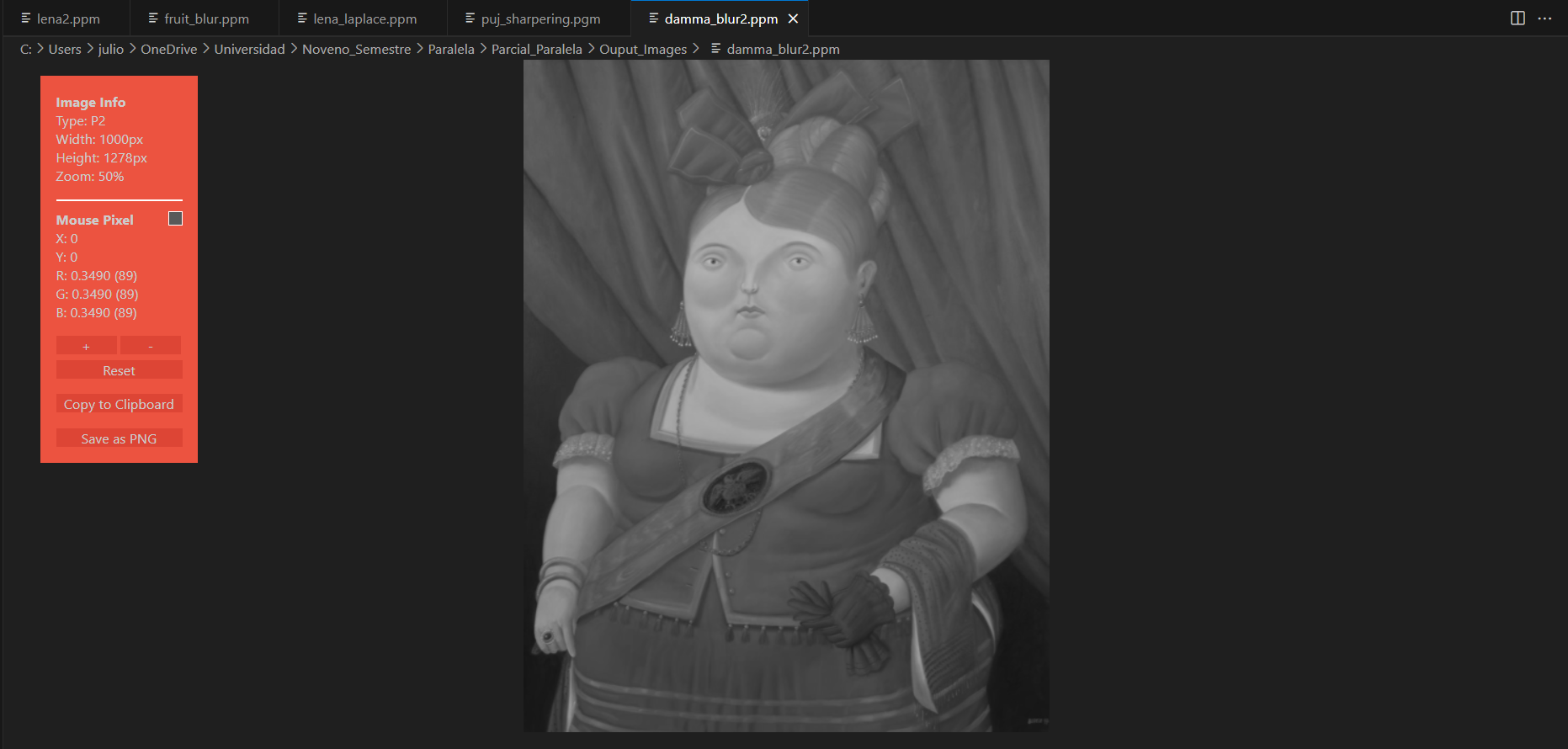




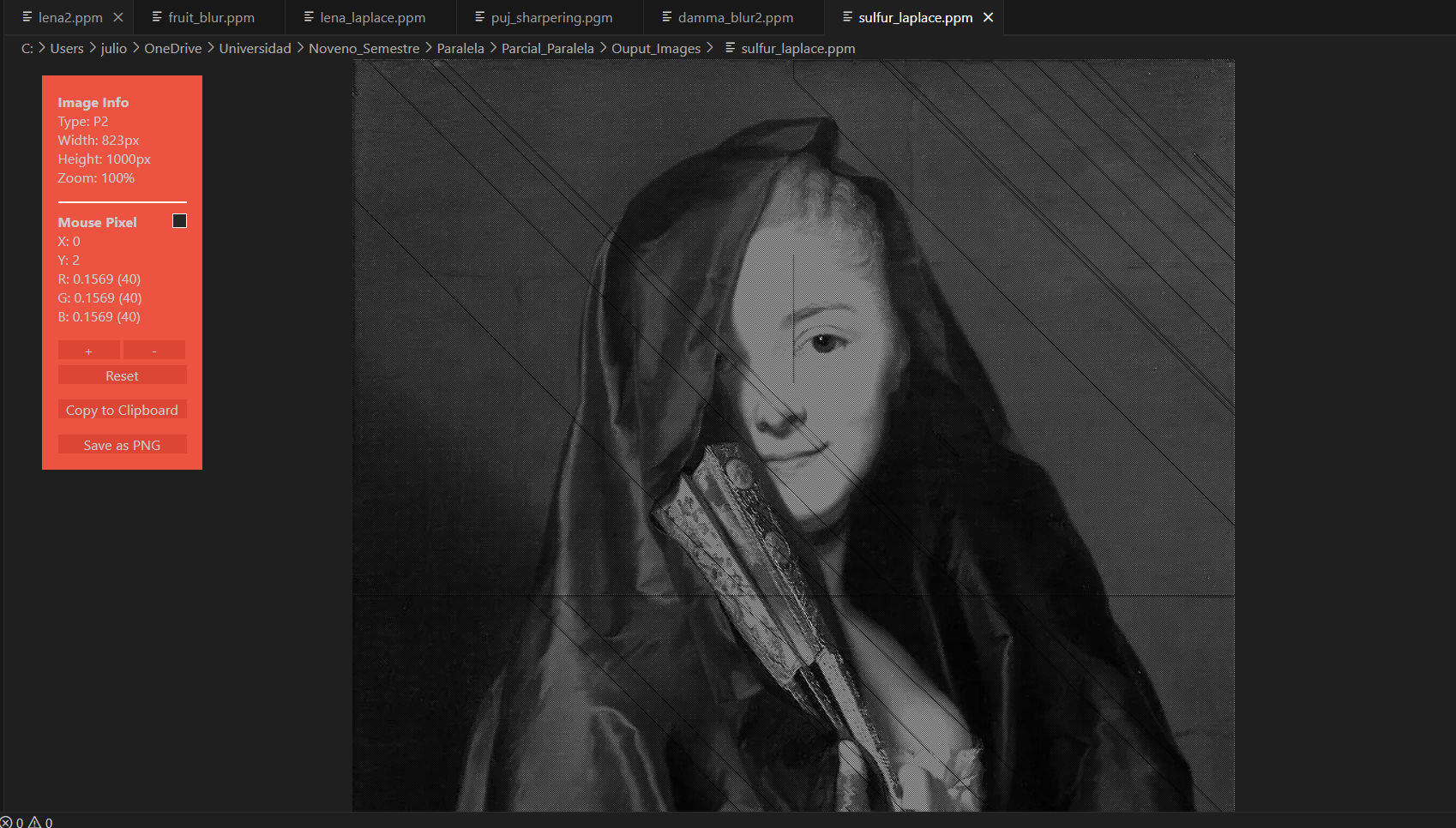


Phtreads:



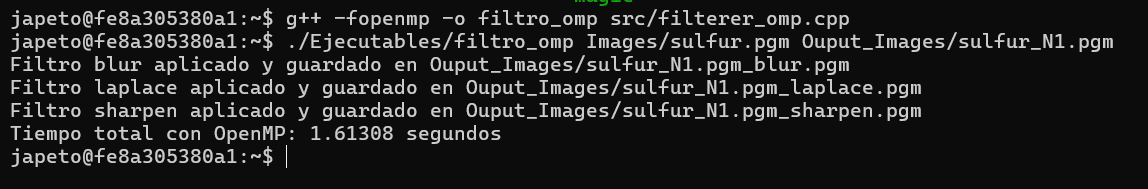


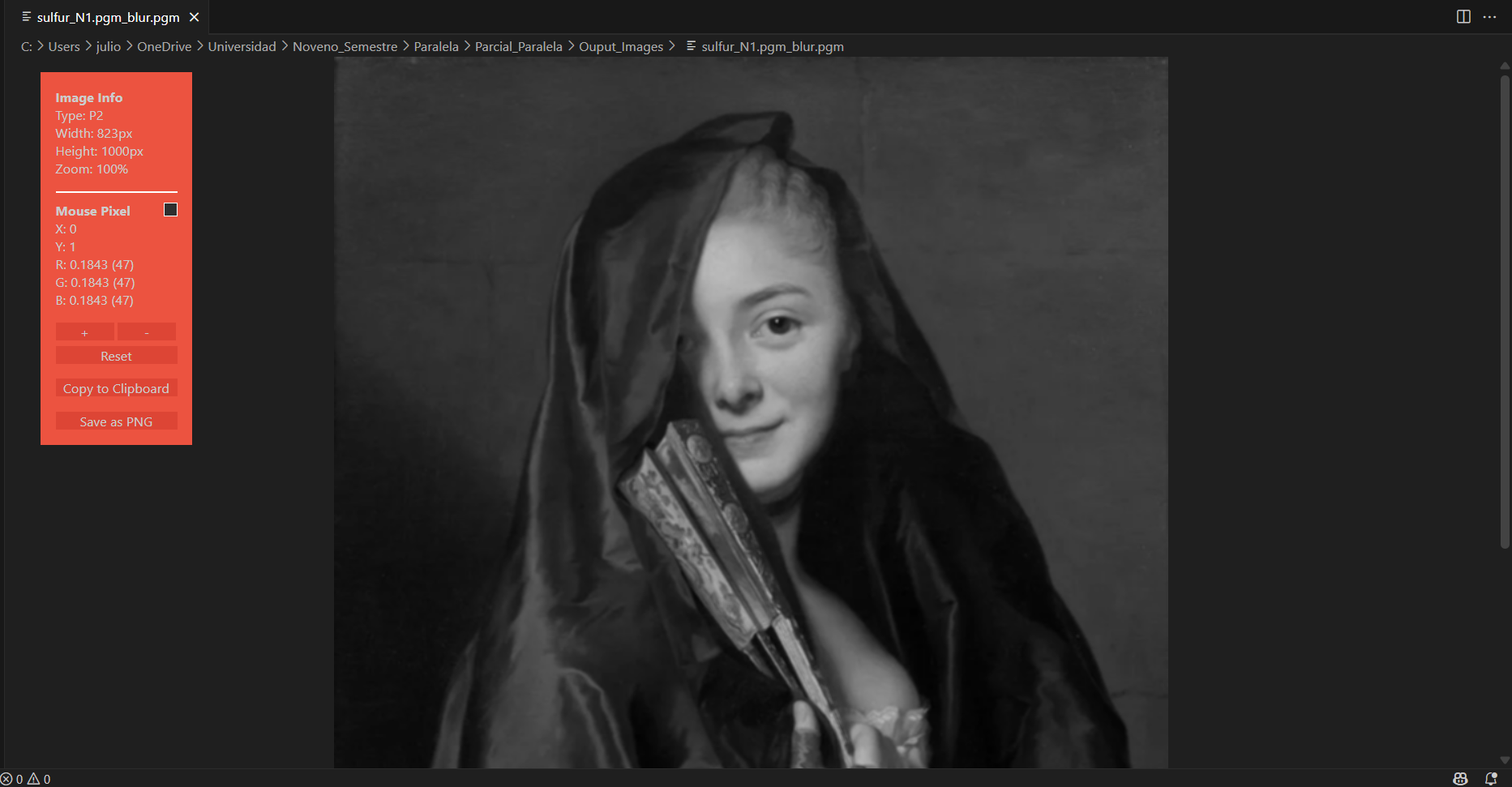




OMP:

**Nota:** Se corrigió el formato y se arreglo la salida de los archivos.





MPI:

