Image filtering INF560

Manas UTESHEV
Vladimir OGORODNIKOV

Defense organization

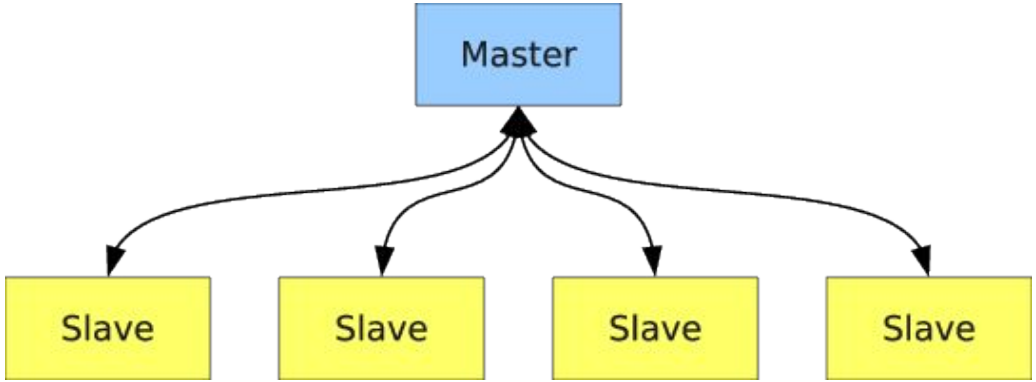
I. Parallelism approaches and choices
II. Algorithms
III. Experimental evaluations

Parallelism approaches and choices

Pure MPI version

1. Master node is a scheduler

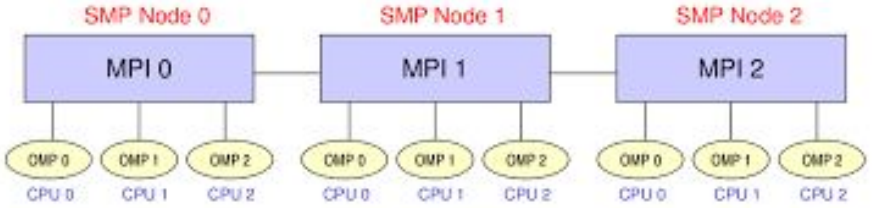
2. Slaves process images



MPI + OpenMP version

I. Each process has its own threadpool

II. OpenMP threads in filters



MPI “striping” mode

I. Use of all available resources
II. Each image is divided in stripes
III. Synchronization of neighbouring stripes
IV. Decision policy for MPI: striping or legacy

I/O optimization

I. Reduce abundant network communications

II. Loading and storing of pixels done by slaves

CUDA implementation

I. Reduce the computation time of blur filter

II. CUDA streams for asynchronous kernel calls

III. Multi-block reduction technique for “end” variable

Algorithms

Legacy MPI

```
def master():
	for i in range(1, world_size - 1):
		Irecv(signal[i], i, req[i])

	while processed_images < N:
		i = Waitany(req)
		if signal[i] >= 0:
			processed[i] = True
			processed_images += 1
		if sent_images < N:
			Send(sent_images, i)
			Irecv(signal[i], i, req[i])
			sent_images += 1
		else:
			Send(-1, i)
			req[i] = MPI_REQUEST_NULL

	for i in range(1, world_size - 1):
		if req[i] != MPI_REQUEST_NULL:
			Wait(req[i])
			Send(-1, i)
```

```
def slave():
	Send(-1, 0)
	while True:
		i = Recv(0)
		if i < 0:
			break
		process(i)
		Send(i, 0)
```

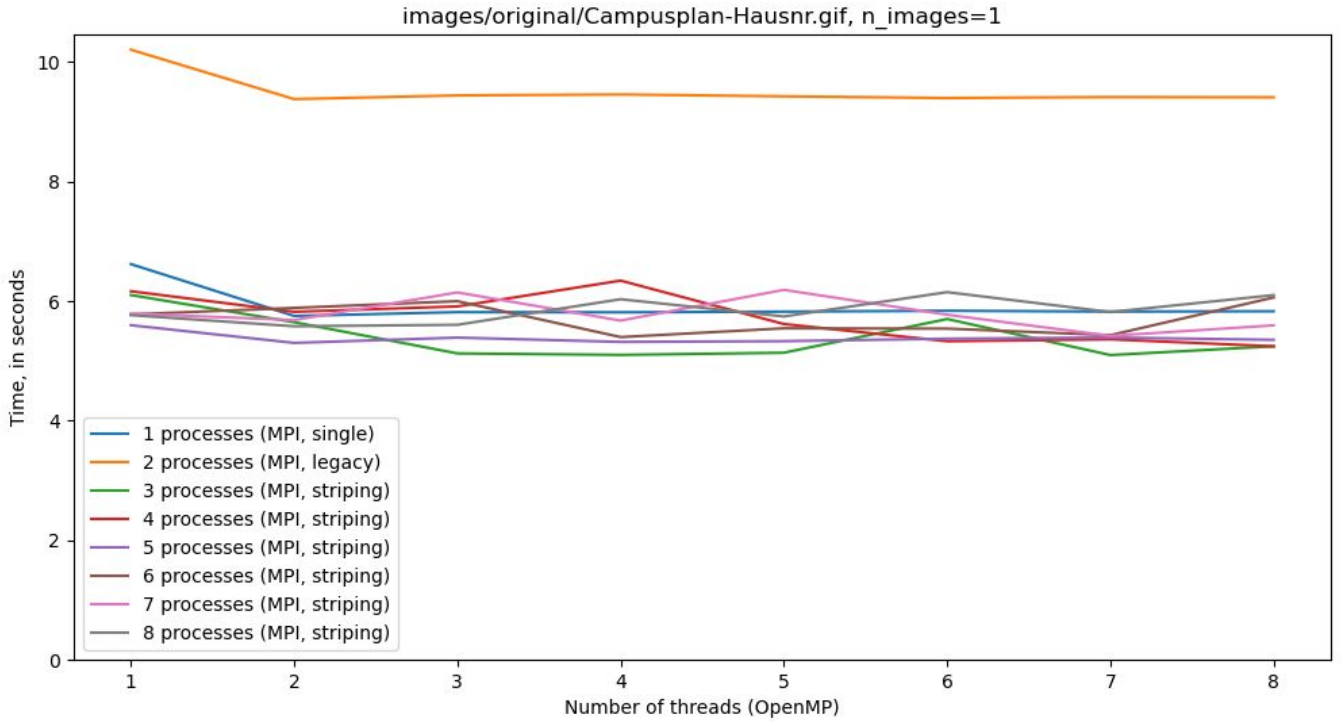
Striping MPI

```
typedef struct {
    int min_row;
    int max_row;
    int single_mode;
    int top_neighbour_id;
    int bottom_neighbour_id;
    int stripe_count;
} striping_info;
```

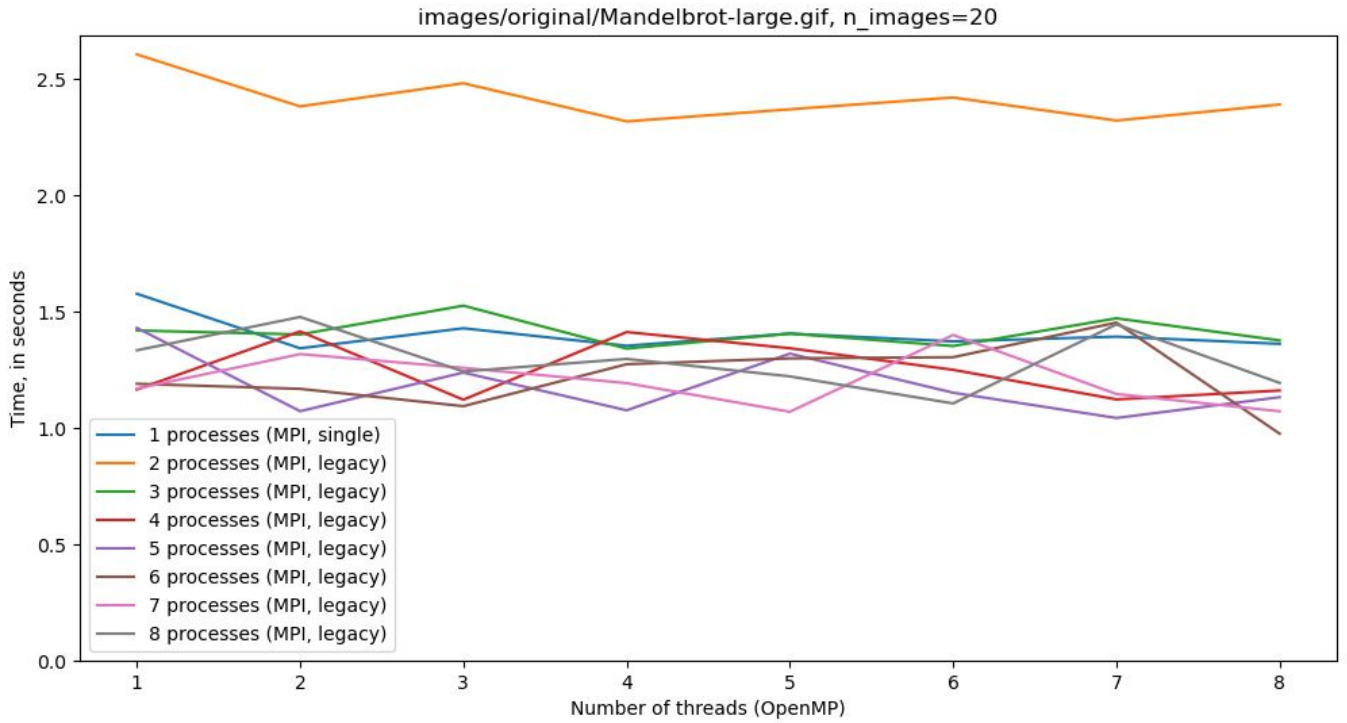
```
def blur_filter():
    while True:
        end = 1
        if not s_info.single_mode:
            synchronize_rows(s_info)

        # OpenMP stuff, setting up 'end'

        if not s_info.single_mode:
            end = synchronize_bool_and(
                                s_info, end)
```

Experimental evaluations

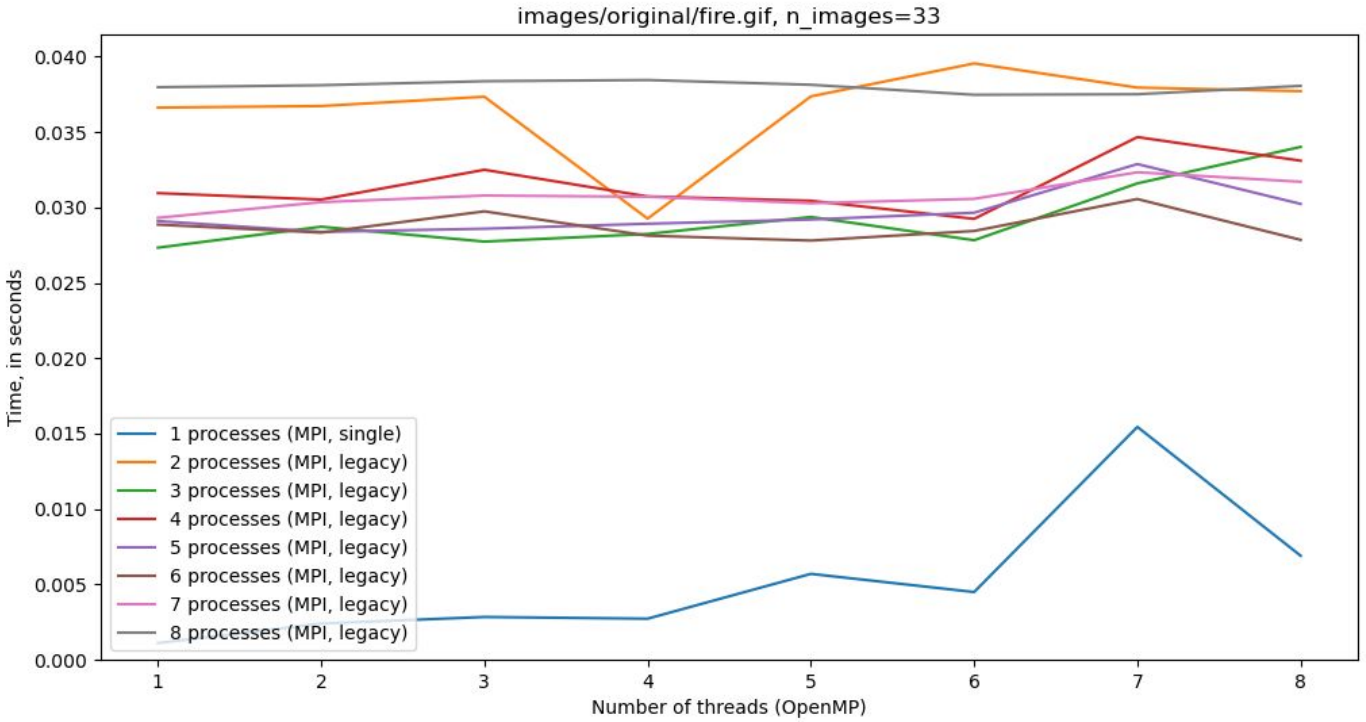
Before IO optimization



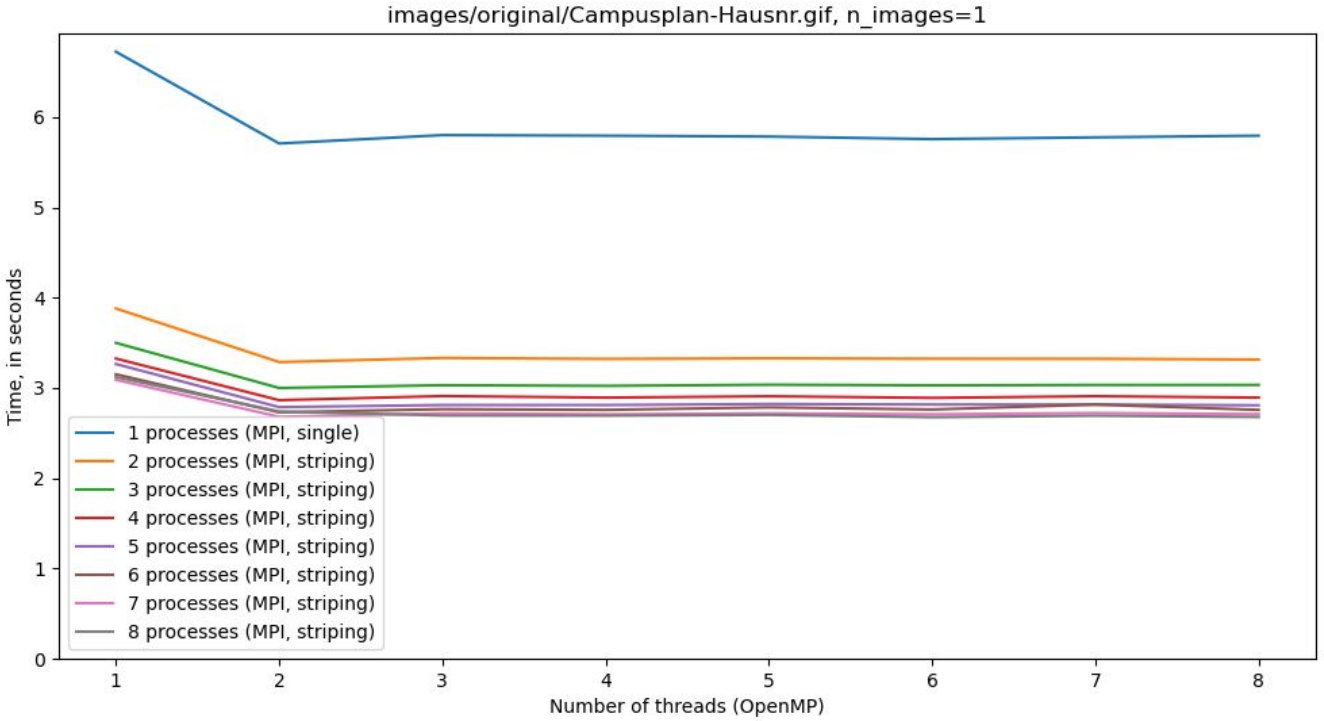
Before IO optimization



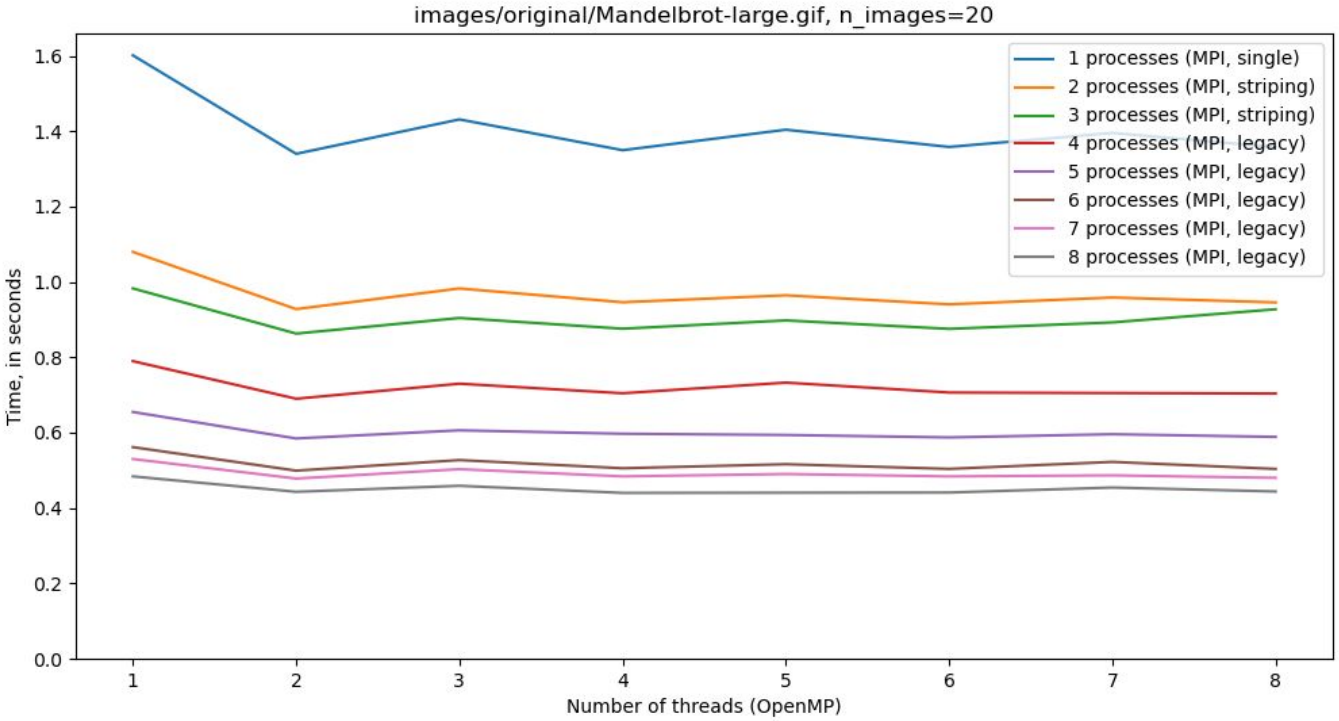
Before IO optimization



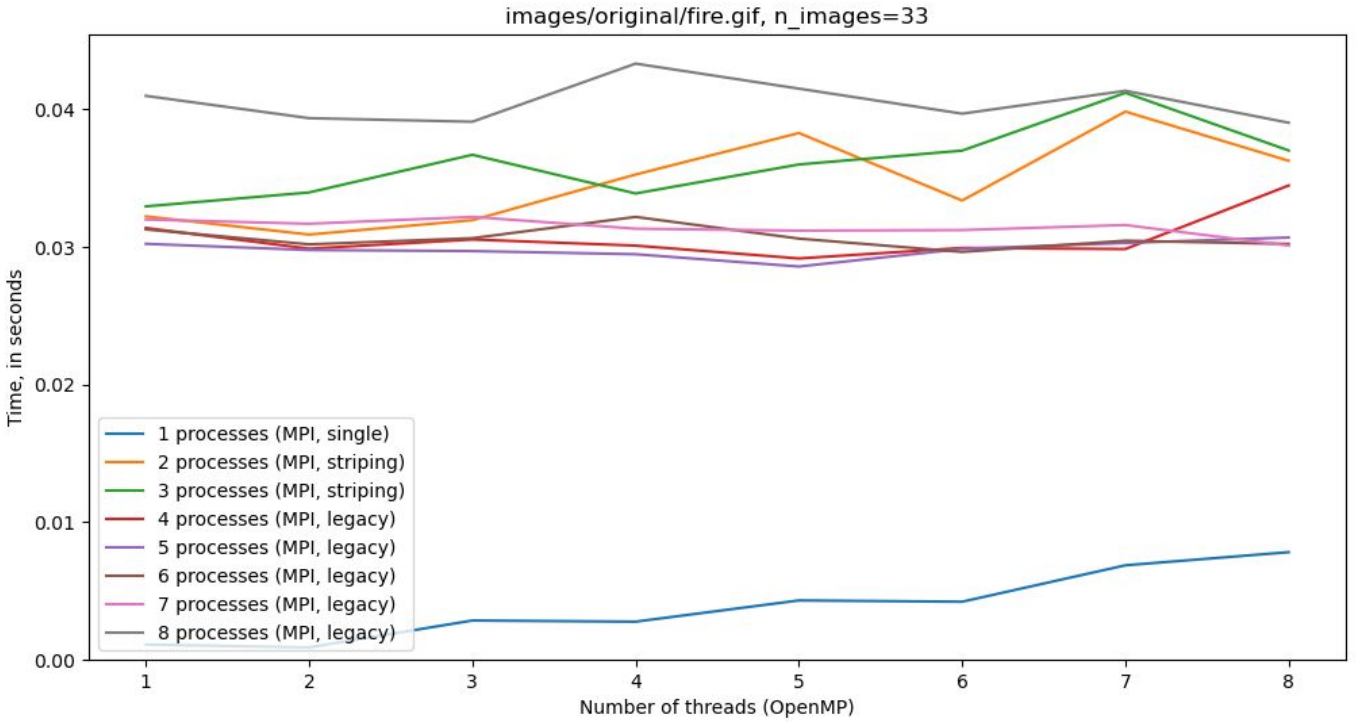
After IO optimization



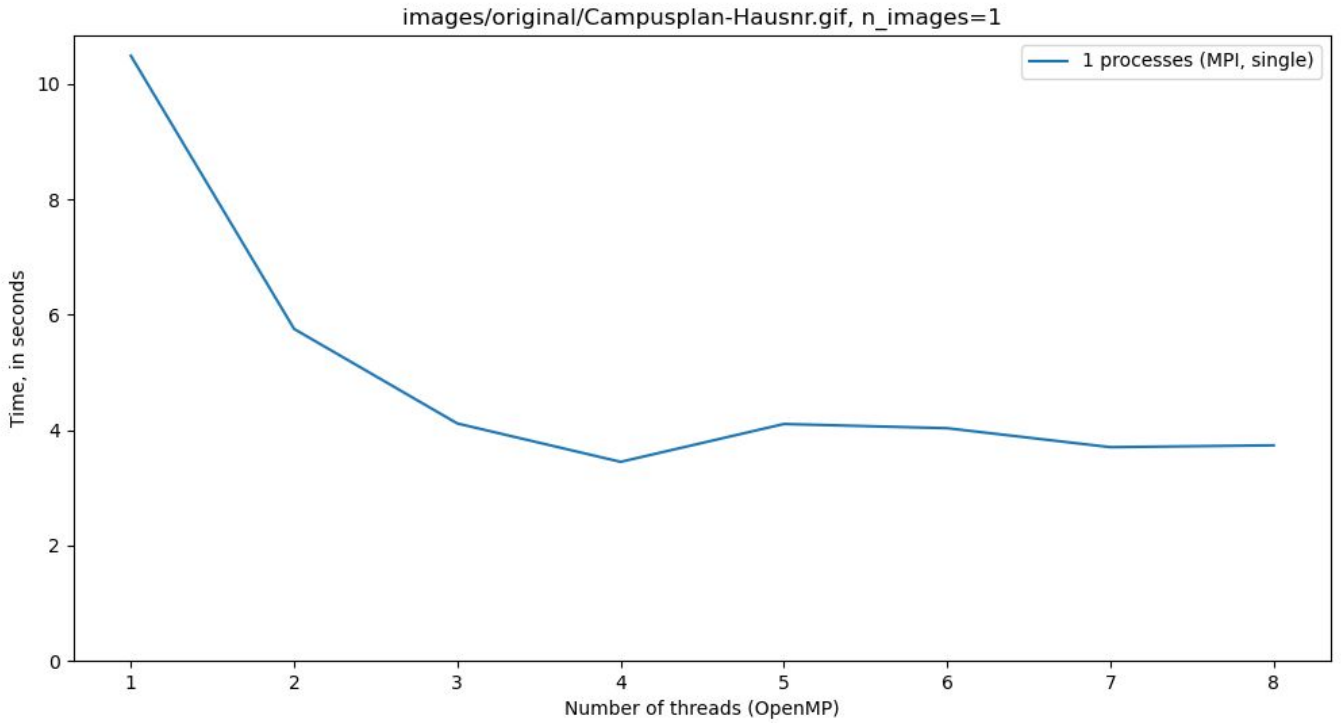
After IO optimization



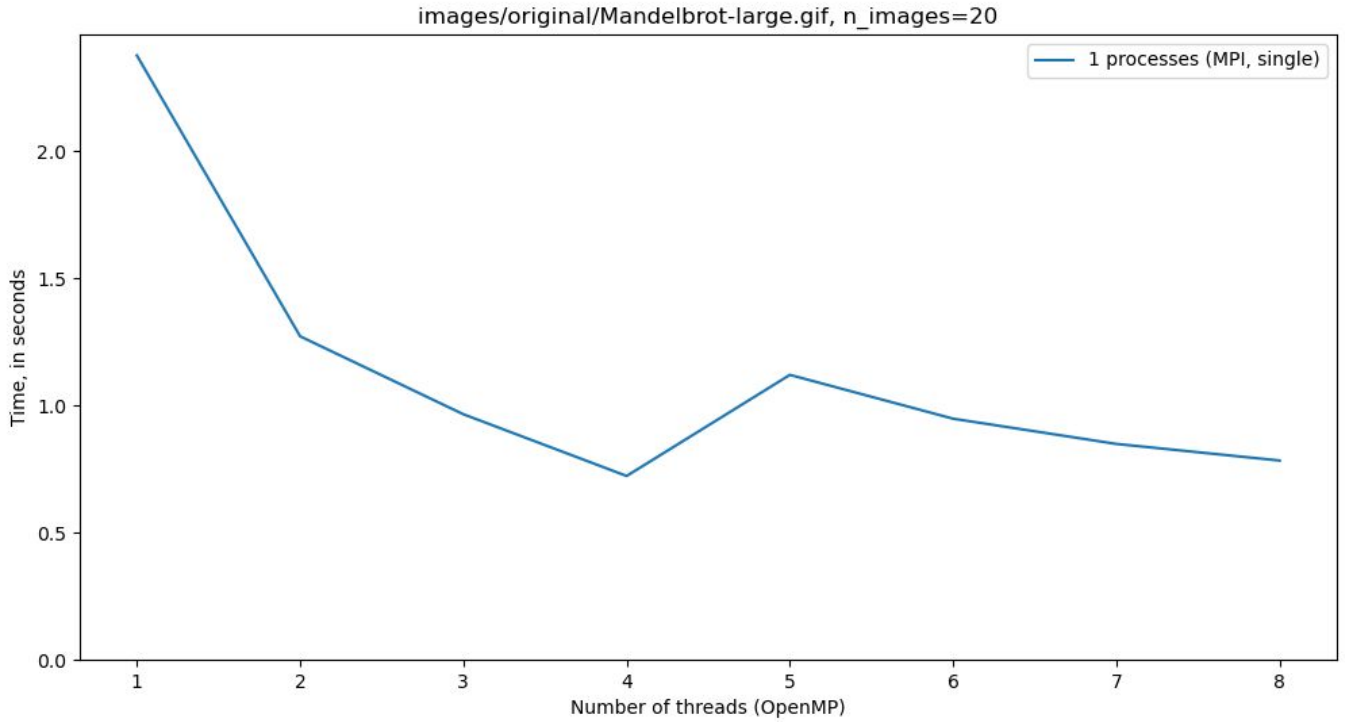
After IO optimization



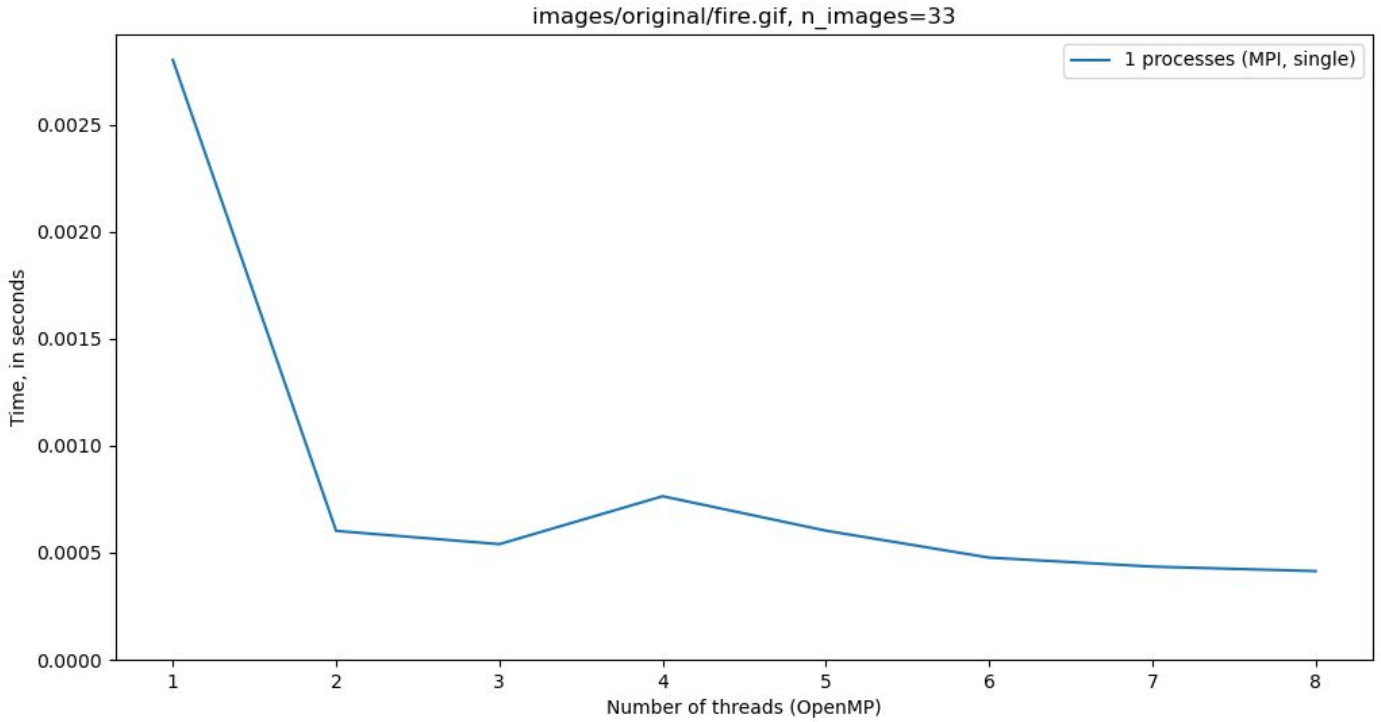
Multithreading



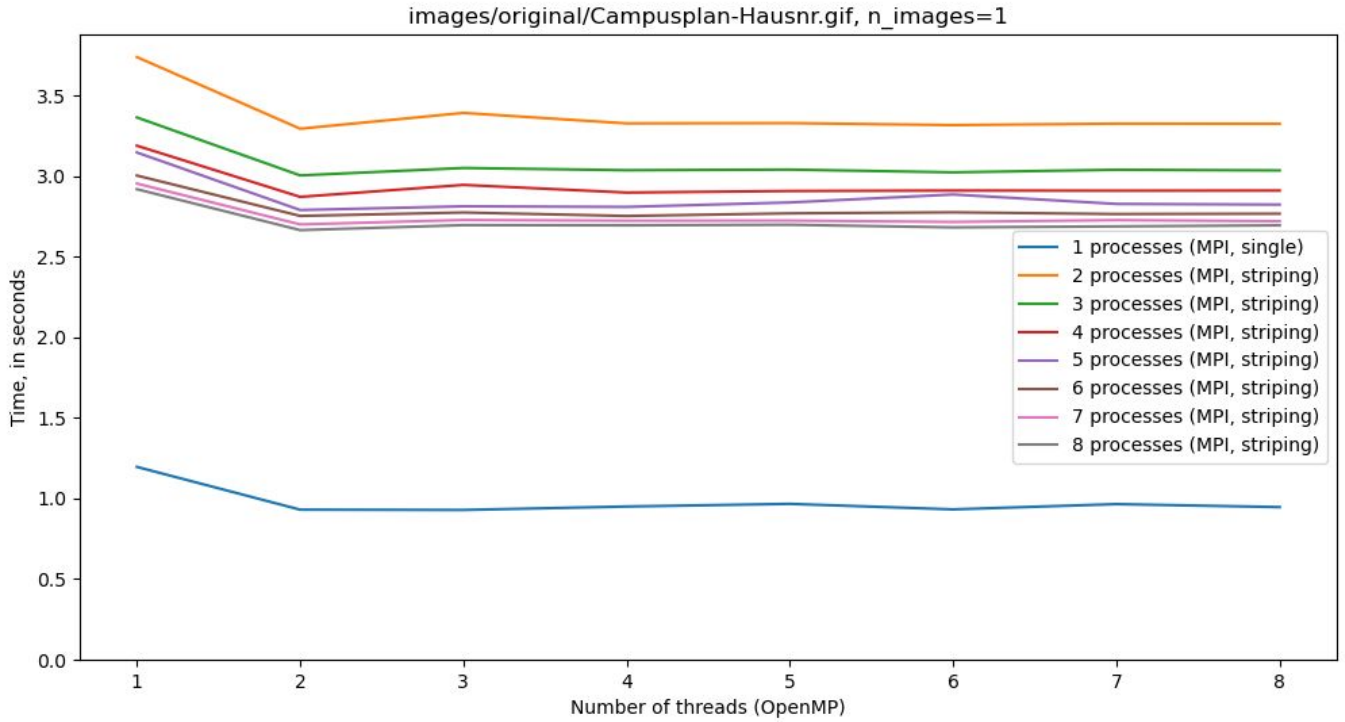
Multithreading



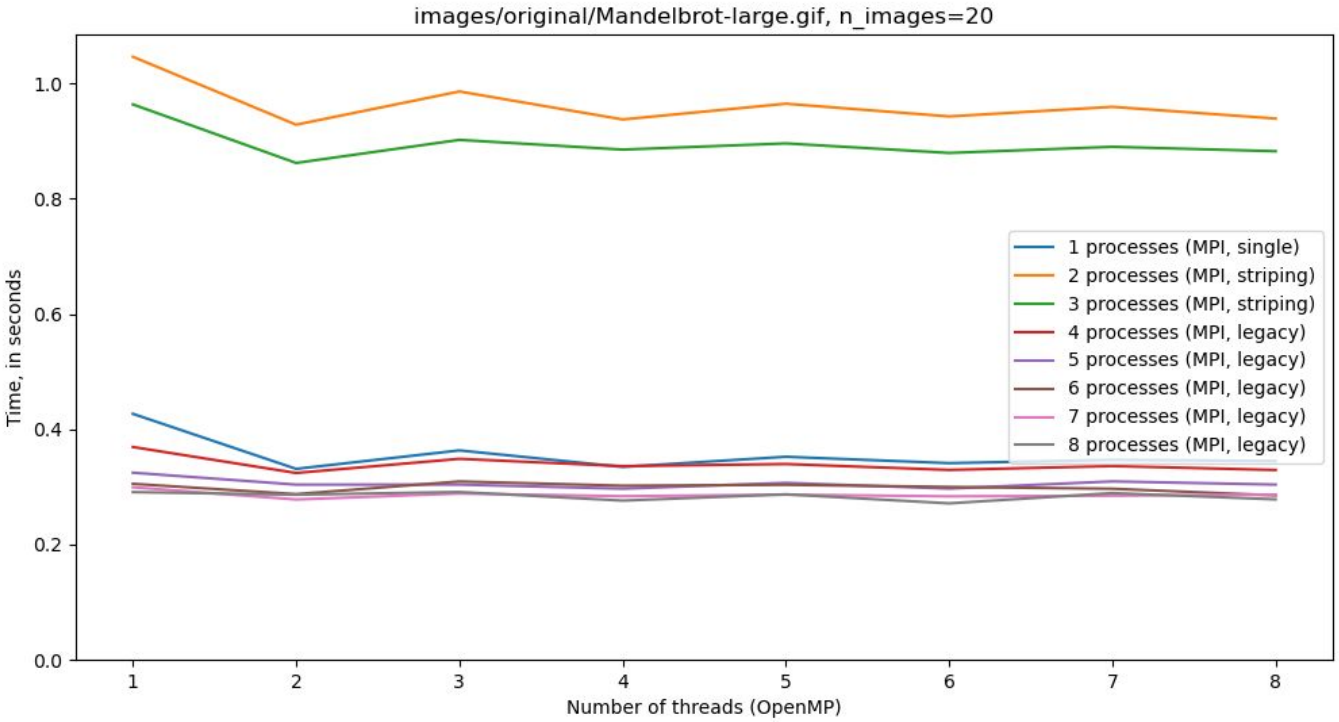
Multithreading



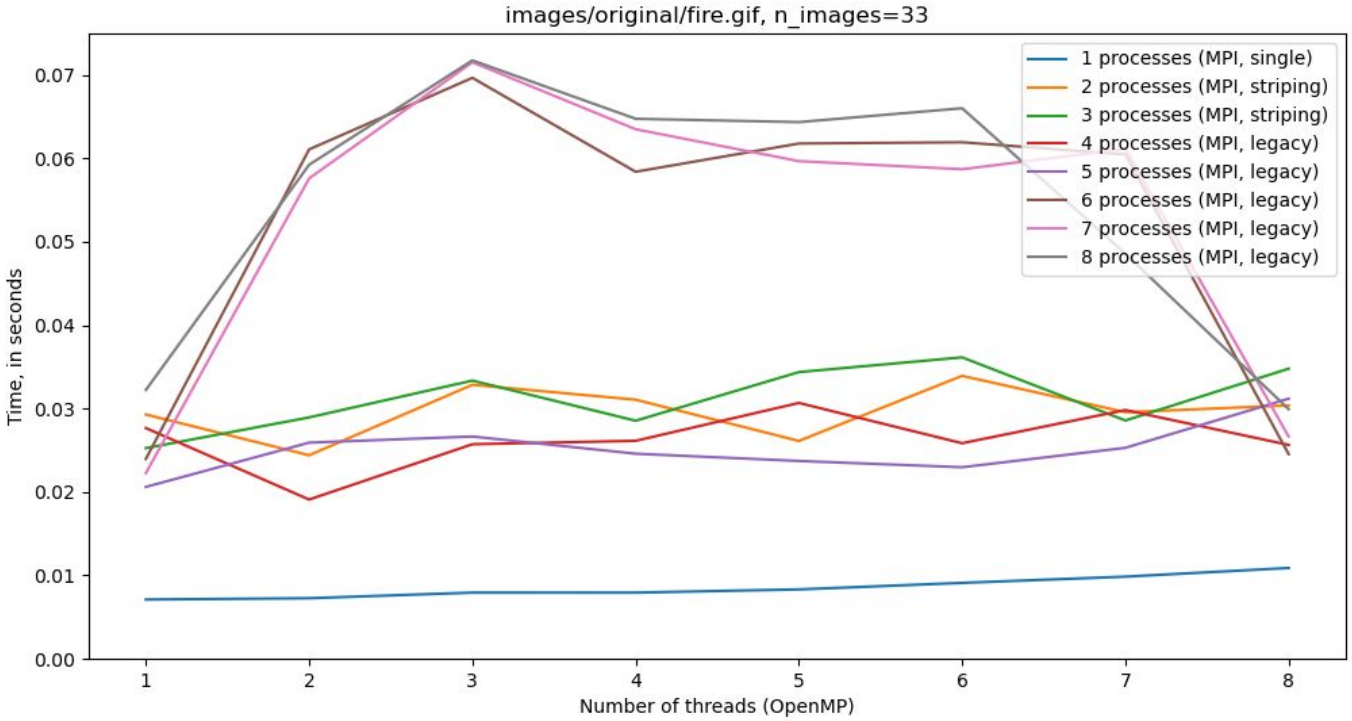
CUDA


images/original/Campusplan-Hausnr.gif, n_images=1
Time, in seconds
Number of threads (OpenMP)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1
2
3
4
5
6
7
8
1 processes (MPI, single)
2 processes (MPI, striping)
3 processes (MPI, striping)
4 processes (MPI, striping)
5 processes (MPI, striping)
6 processes (MPI, striping)
7 processes (MPI, striping)
8 processes (MPI, striping)

CUDA


images/original/Mandelbrot-large.gif, n_images=20
Time, in seconds
Number of threads (OpenMP)
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5
6
7
8
1 processes (MPI, single)
2 processes (MPI, striping)
3 processes (MPI, striping)
4 processes (MPI, legacy)
5 processes (MPI, legacy)
6 processes (MPI, legacy)
7 processes (MPI, legacy)
8 processes (MPI, legacy)

CUDA


images/original/fire.gif, n_images=33
Time, in seconds
Number of threads (OpenMP)
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
1
2
3
4
5
6
7
8
1 processes (MPI, single)
2 processes (MPI, striping)
3 processes (MPI, striping)
4 processes (MPI, legacy)
5 processes (MPI, legacy)
6 processes (MPI, legacy)
7 processes (MPI, legacy)
8 processes (MPI, legacy)