draft

August 20, 2023

1 Asynchronous and concurrent execution on GPUs

Melina Abeling, Julian Aeissen, Michele Pagani. Supervised by *Oliver Fuhrer*

GPUs allow for asynchronous (CPU simply launches work on GPU and then continues) and concurrent (multiple tasks are executed in parallel on the GPU) execution. In this project you will implement a simple stencil program and investigate performance using different stencil motifs as a function of grid size and amount of concurrency. It is foreseen to use either CuPy or CUDA for this project.

1.1 Introduction

TODO

1.2 Methods

TODO

[1]:
```
# Imports
import time
import cupy as cp
import numpy as np
import matplotlib.pyplot as plt
import math
```

[2]:
```
# Utils
def update_halo(field, num_halo):
    # Checks
    dim = len(field.shape)
    assert dim in [2,3]

    # 2d case
    if dim == 2:
        # bottom edge (without corners)
        field[:num_halo, num_halo:-num_halo] = field[
            -2 * num_halo : -num_halo, num_halo:-num_halo
```

```
        ]

        # top edge (without corners)
        field[-num_halo:, num_halo:-num_halo] = field[
            num_halo : 2 * num_halo, num_halo:-num_halo
        ]

        # left edge (including corners)
        field[:, :num_halo] = field[:, -2 * num_halo : -num_halo]

        # right edge (including corners)
        field[:, -num_halo:] = field[:, num_halo : 2 * num_halo]

    # 3d case
    elif dim == 3:
        # bottom edge (without corners)
        field[:, :num_halo, num_halo:-num_halo] = field[
            :, -2 * num_halo : -num_halo, num_halo:-num_halo
        ]

        # top edge (without corners)
        field[:, -num_halo:, num_halo:-num_halo] = field[
            :, num_halo : 2 * num_halo, num_halo:-num_halo
        ]

        # left edge (including corners)
        field[:, :, :num_halo] = field[:, :, -2 * num_halo : -num_halo]

        # right edge (including corners)
        field[:, :, -num_halo:] = field[:, :, num_halo : 2 * num_halo]
```

1.2.1 Stencils

TODO

Example

[3]:
```
def step_stencil_2d_example(in_field, out_field, n_halo):
    # Checks
    assert len(in_field.shape) == 2
    assert len(out_field.shape) == 2
    h,w = out_field.shape
    h_in_,w_in_ = in_field.shape
    assert h_in_ == h + 2*n_halo
    assert w_in_ == w + 2*n_halo

    # Example with a simple Gaussian filter
    # IMPORTANT always have an expected halo
```

```python
    assert n_halo == 1

    # Computation
    out_field[:,:] = (
        4.0 * in_field[1:-1, 1:-1]
        + 2.0 * in_field[2:, 1:-1]
        + 2.0 * in_field[:-2, 1:-1]
        + 2.0 * in_field[1:-1, 2:]
        + 2.0 * in_field[1:-1, :-2]
        + in_field[2:, 2:]
        + in_field[2:, :-2]
        + in_field[:-2, 2:]
        + in_field[:-2, :-2]
    ) / 16.0

    # out_field[:,:] = in_field[2:, 1:-1]
```

[4]:
```python
def step_stencil_3d_example(in_field, out_field, n_halo):
    # Checks
    assert len(in_field.shape) == 3
    assert len(out_field.shape) == 3
    _, h,w = out_field.shape
    _, h_in_,w_in_ = in_field.shape
    assert h_in_ == h + 2*n_halo
    assert w_in_ == w + 2*n_halo

    # Example with a simple Gaussian filter
    # IMPORTANT always have an expected halo
    assert n_halo == 1

    # Computation
    out_field[:,:,:] = (
        4.0 * in_field[:, 1:-1, 1:-1]
        + 2.0 * in_field[:, 2:, 1:-1]
        + 2.0 * in_field[:, :-2, 1:-1]
        + 2.0 * in_field[:, 1:-1, 2:]
        + 2.0 * in_field[:, 1:-1, :-2]
        + in_field[:, 2:, 2:]
        + in_field[:, 2:, :-2]
        + in_field[:, :-2, 2:]
        + in_field[:, :-2, :-2]
    ) / 16.0
```

A *TODO*

[5]:
```python
def step_stencil_a(field):
    pass
```

B *TODO*

[6]:
```
def step_stencil_b(field):
    pass
```

1.2.2 Initial field

TODO

[7]:
```
def get_initial_field(size, n_halo, value = 1.0) -> cp.ndarray:
    # Check parameters
    assert type(size) == tuple
    dim = len(size)
    assert dim in [2,3]

    # Init
    h, w = size[-2], size[-1]

    # Add halo
    h += 2*n_halo
    w += 2*n_halo

    # 2d
    if dim == 2:
        field = cp.zeros((h, w))
        field[ h//4 : 3*h//4,
               w//4 : 3*w//4 ] = value

    # 3d
    elif dim == 3:
        field = cp.zeros((size[0], h, w))
        field[ :,
               h//4 : 3*h//4,
               w//4 : 3*w//4 ] = value

    return field
```

1.2.3 Sequential

TODO

[8]:
```
def sequential_computation(stencil, field):
    pass
```

1.2.4 Concurrent

TODO

[9]:
```python
def compute_gpu(in_field, stencil, n_stream, n_iter, n_halo, tile=True):
    # Init
    out_field = cp.copy(in_field)

    # Chech n_stream
    if tile:
        assert math.sqrt(n_stream).is_integer()
        stream_per_side = int(math.sqrt(n_stream))
    else:
        stream_per_side = n_stream

    # Check in_field
    dim = len(in_field.shape)
    assert dim in [2, 3]
    h,w = in_field.shape[-2], in_field.shape[-1]
    h -= 2*n_halo
    w -= 2*n_halo
    # assert h % stream_per_side == 0
    # assert w % stream_per_side == 0

    if tile:
        h_stream = h // stream_per_side
        w_stream = w // stream_per_side
    else:
        h_stream = h // stream_per_side
        w_stream = w

    is_3d = dim == 3

    # Create streams
    streams = [ cp.cuda.Stream() for _ in range(n_stream) ]

    for iter in range(n_iter):
        # Init
        e = cp.cuda.Event()
        e.record()

        update_halo(in_field, n_halo)

        # Iterate over streams
        for idx, s in enumerate(streams):
            # Indeces
            if tile:
                i, j = idx // stream_per_side, idx % stream_per_side
            else:
                # If no tiles, divide only the first dimension
                i, j = idx, 0
```

```
            with s:
                # Stencil iteration
                # print(f"i = {i}, j = {j}, len in = ({-i*h_stream + 2*n_halo + (i+1)*h_stream}, {-j*w_stream+ 2*n_halo + (j+1)*w_stream}), len out = {(-(n_halo + i*h_stream) + n_halo + (i+1)*h_stream, -( n_halo + j*w_stream) + n_halo + (j+1)*w_stream)}, pos in = [{(i*h_stream , 2*n_halo + (i+1)*h_stream)}, {(j*w_stream , 2*n_halo + (j+1)*w_stream)}], pos out = [{((n_halo + i*h_stream), n_halo + (i+1)*h_stream)}, { ( n_halo + j*w_stream, n_halo + (j+1)*w_stream) }], h_stream = {h_stream}, w_stream  = {w_stream}, h = {h}")
                if is_3d:
                    stencil(
                        in_field[
                            :,
                            i*h_stream: 2*n_halo + (i+1)*h_stream,
                            j*w_stream: 2*n_halo + (j+1)*w_stream
                        ],
                        out_field[
                            :,
                            n_halo + i*h_stream: n_halo + (i+1)*h_stream,
                            n_halo + j*w_stream: n_halo + (j+1)*w_stream
                        ],
                        n_halo
                    )
                else:
                    stencil(
                        in_field[
                            i*h_stream: 2*n_halo + (i+1)*h_stream,
                            j*w_stream: 2*n_halo + (j+1)*w_stream
                        ],
                        out_field[
                            n_halo + i*h_stream: n_halo + (i+1)*h_stream,
                            n_halo + j*w_stream: n_halo + (j+1)*w_stream
                        ],
                        n_halo
                    )

        # Syncronize all streams
        e.synchronize()

        # Update out_field
        if iter < n_iter - 1:
            in_field, out_field = out_field, in_field

    return out_field
```

[10]:
```
# %%timeit

# initial_field = get_initial_field((16, 1024, 1024), 1);
# compute_gpu( initial_field, step_stencil_3d_example, 4, 1, 1);
# initial_field.shape
```

1.3 Results

TODO

1.3.1 Performance over concurrency

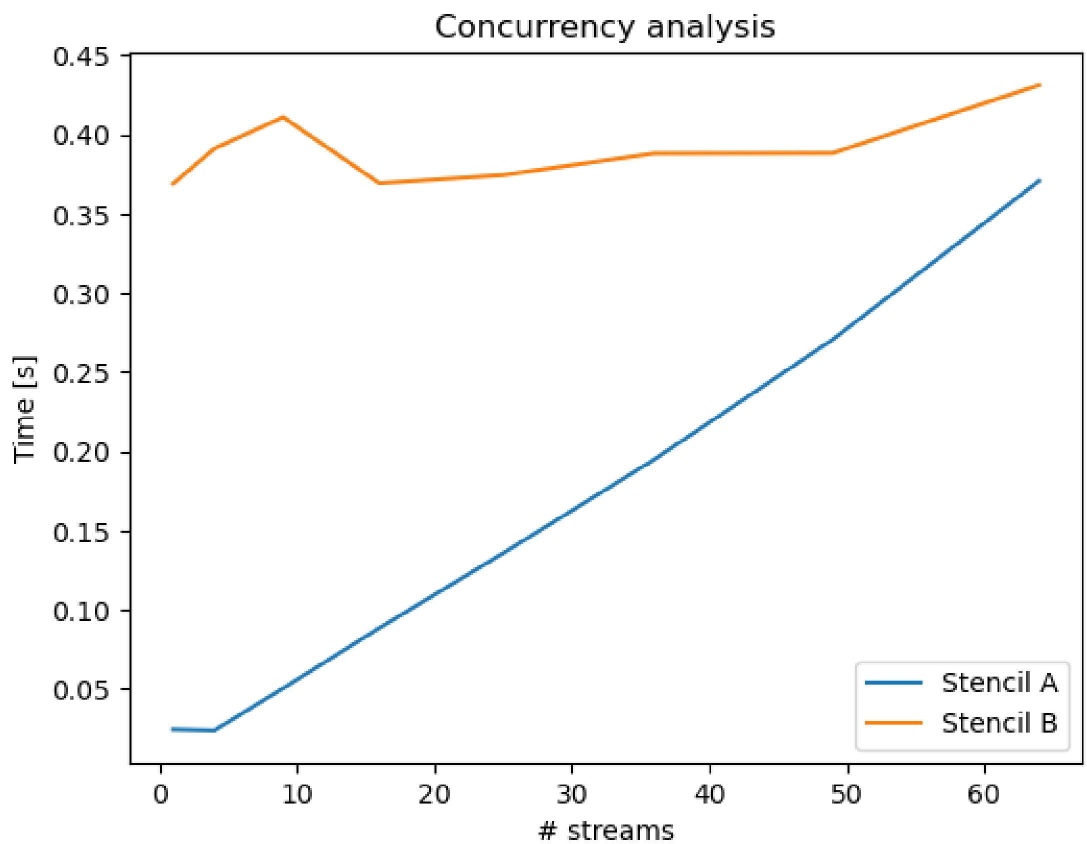
TODO

[11]:
```
### TODO DELETE
step_stencil_a = step_stencil_2d_example
step_stencil_b = step_stencil_3d_example

# Settings
steps_concurrency = [1, 4, 9, 16, 25, 36, 49, 64]
field_size_a = (1024, 1024)
field_size_b = (16, 1024, 1024)
n_iter = 20
n_iter_stats = 20
n_halo = 1

# Setup
input_field_concur_a = get_initial_field(field_size_a, n_halo)
input_field_concur_b = get_initial_field(field_size_b, n_halo)
output_fields_a_concur = []
output_fields_b_concur = []
times_a_concur = []
times_b_concur = []
times_transfer_a_concur = []
times_transfer_b_concur = []


for concurrency in steps_concurrency:

    # Compute for stencil A
    temp_times_a = []
    temp_times_transfer_a = []

    # Iterate for time statistics
    for _ in range(n_iter_stats):
        # Compute
```

```
        tic = time.perf_counter()
        temp_output_field_a_gpu = compute_gpu(cp.copy(input_field_concur_a),␣
↪step_stencil_a, concurrency, n_iter, n_halo, tile=False)
        toc = time.perf_counter()
        temp_times_a.append(toc - tic)

        # Get results from GPU
        tic = time.perf_counter()
        temp_output_field_a = temp_output_field_a_gpu.get()
        toc = time.perf_counter()
        temp_times_transfer_a.append(toc-tic)
        output_fields_a_concur.append(temp_output_field_a)
    times_a_concur.append(temp_times_a)
    times_a_concur.append(temp_times_a)

    # Compute for stencil B
    temp_times_b = []
    temp_times_transfer_b = []

    # Iterate for time statistics
    for _ in range(n_iter_stats):
        # Compute
        tic = time.perf_counter()
        temp_output_field_b_gpu = compute_gpu(cp.copy(input_field_concur_b),␣
↪step_stencil_b, concurrency, n_iter, n_halo, tile = False)
        toc = time.perf_counter()
        temp_times_b.append(toc - tic)
        # Get results from GPU
        tic = time.perf_counter()
        temp_output_field_b = temp_output_field_b_gpu.get()
        toc = time.perf_counter()
        temp_times_transfer_b.append(toc-tic)
        output_fields_b_concur.append(temp_output_field_b)
    times_b_concur.append(temp_times_b)
```

[12]:

```
# Plots
plt.title("Concurrency analysis")

plt.plot(steps_concurrency, np.mean(times_a_concur, axis = 1), label = "Stencil␣
↪A")
plt.plot(steps_concurrency, np.mean(times_b_concur, axis = 1), label = "Stencil␣
↪B")

plt.xlabel('# streams')
plt.ylabel('Time [s]')
plt.legend()
plt.ticklabel_format(axis='y', useOffset=False)
```

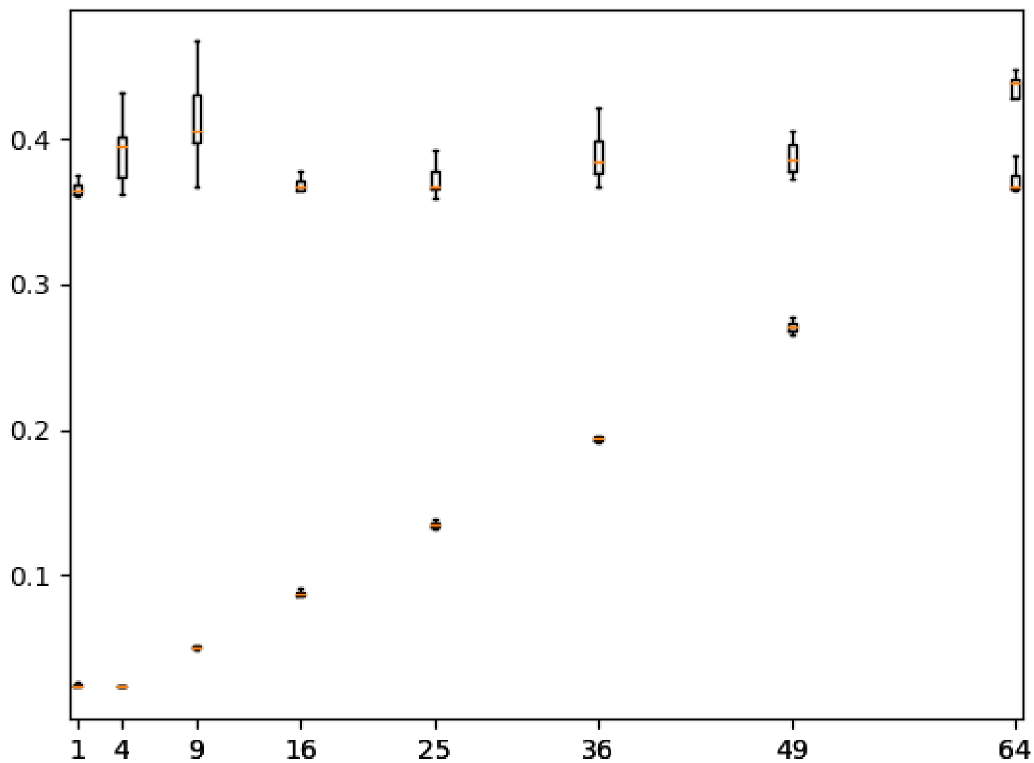


[ ]:

```

```

[24]:

```
# plt.boxplot(np.array(times_a_concur).transpose())
plt.boxplot(np.array(times_a_concur).transpose(), positions =␣
 ↪steps_concurrency, showfliers = False);
plt.boxplot(np.array(times_b_concur).transpose(), positions =␣
 ↪steps_concurrency, showfliers = False);
```



[15]:
```
np.array(times_a_concur).shape
len(steps_concurrency)
```

[15]: 8

[ ]:
```
# img = output_fields_a_concur[2].get()
# img = input_field_concur_a.get()
img = output_fields_b_concur[4].get()[5,:,:]
plt.imshow(img)
print(cp.max(img))
```

1.3.2 Performance over grid size

TODO

[ ]:
```
# TODO
```

[ ]:
```
# Plots
```

1.4 Discussion

TODO

1.5 Conclusion

TODO

```
[ ]:
```