C++ free style assignment: Bee tracker

Sylvain Noiry

Winter semester 2022/2023

Karlsruher Institut für Technologie

Table of Contents

Introduction

Beekeepers and searchers try to better understand the behavior of bees to prevent the mortality of colonies.

What can be collected for now ?

- Beehive weight
- Inside and outside temperature
- Humidity
- Sound
- ...

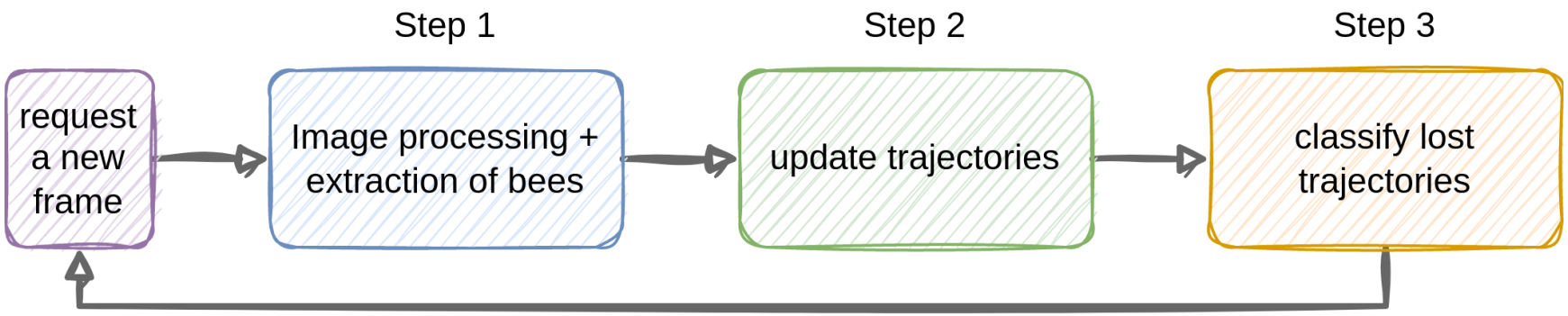
$\Rightarrow$ Lack of data from the activity at the beehive entrance

Introduction

- Put a camera on top of the beehive
- Real time video processing and paths tracking
- Low power budget (solar powered system)

Typical Hardware: Raspberry pi zero 2w (4x Cortex-A53) + camera $\Rightarrow$ Presented benchmarks have been done on a Raspberry pi 3B (roughly the same SoC)

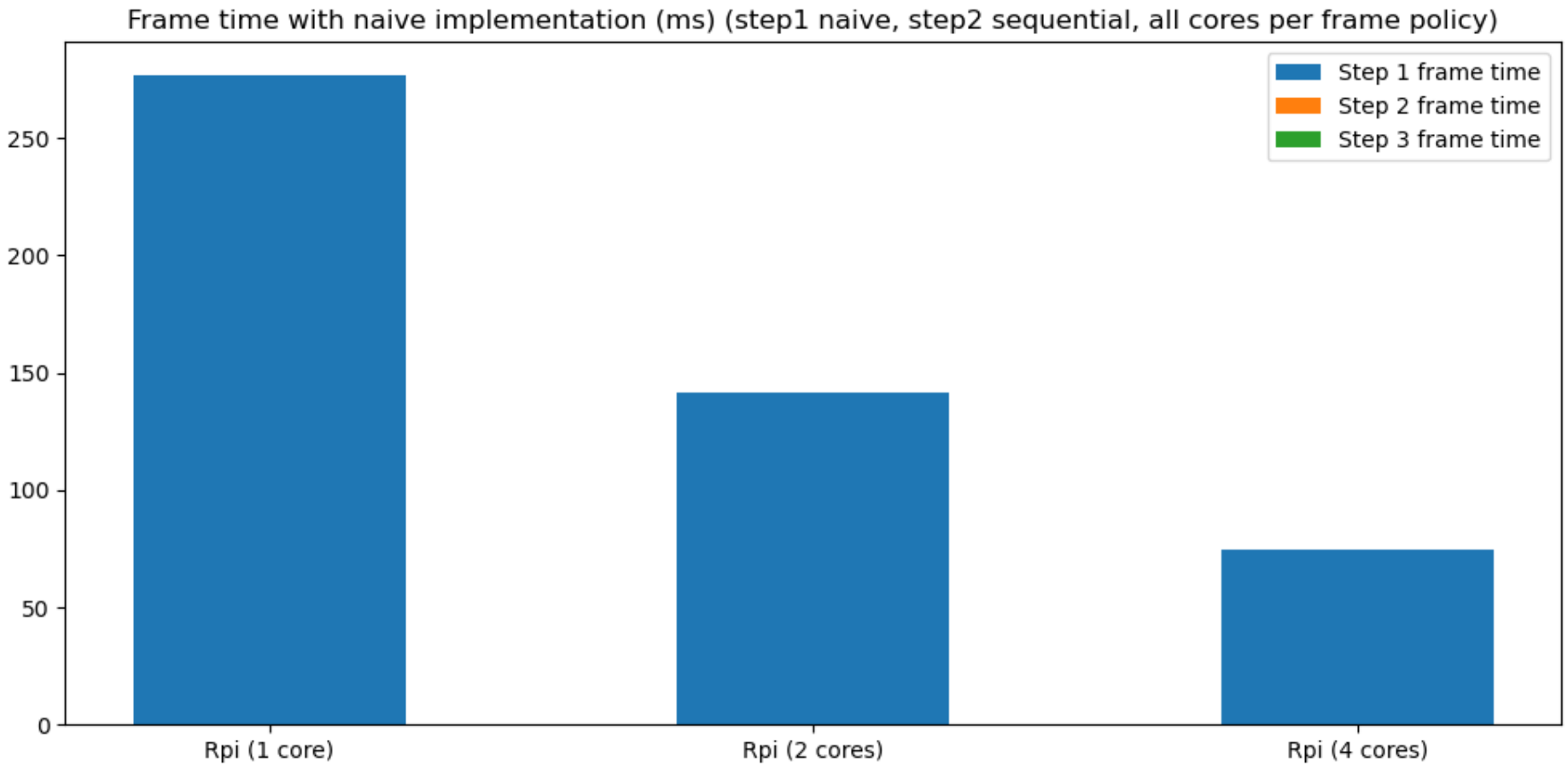
Software architecture overview



Approach based on two recent papers

- **Bee hive traffic monitoring by tracking bee flight paths**, *Baptiste Magnier, Gaëtan Ekszterowicz, Joseph Laurent, Matthias Rival, François Pfister* (2018)
- **Multiple honey bees tracking and trajectory modeling**, *Baptiste Magnier, Eliyahou Gabbay, Faysal Bougamale, Behrang Moradi, François Pfister, Pierre Slangen* (2019)

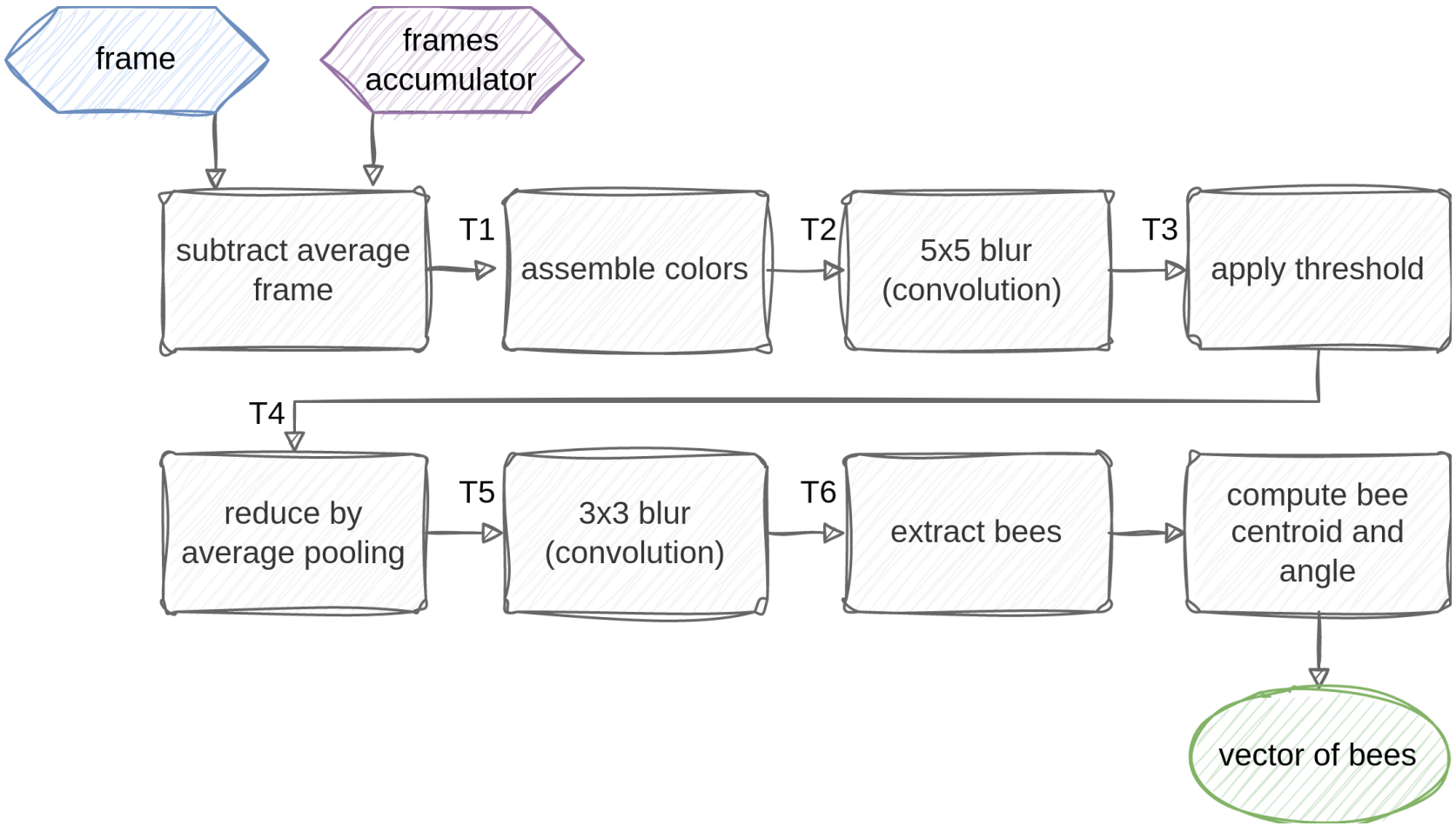
Early benchmark: < 15 *fps*



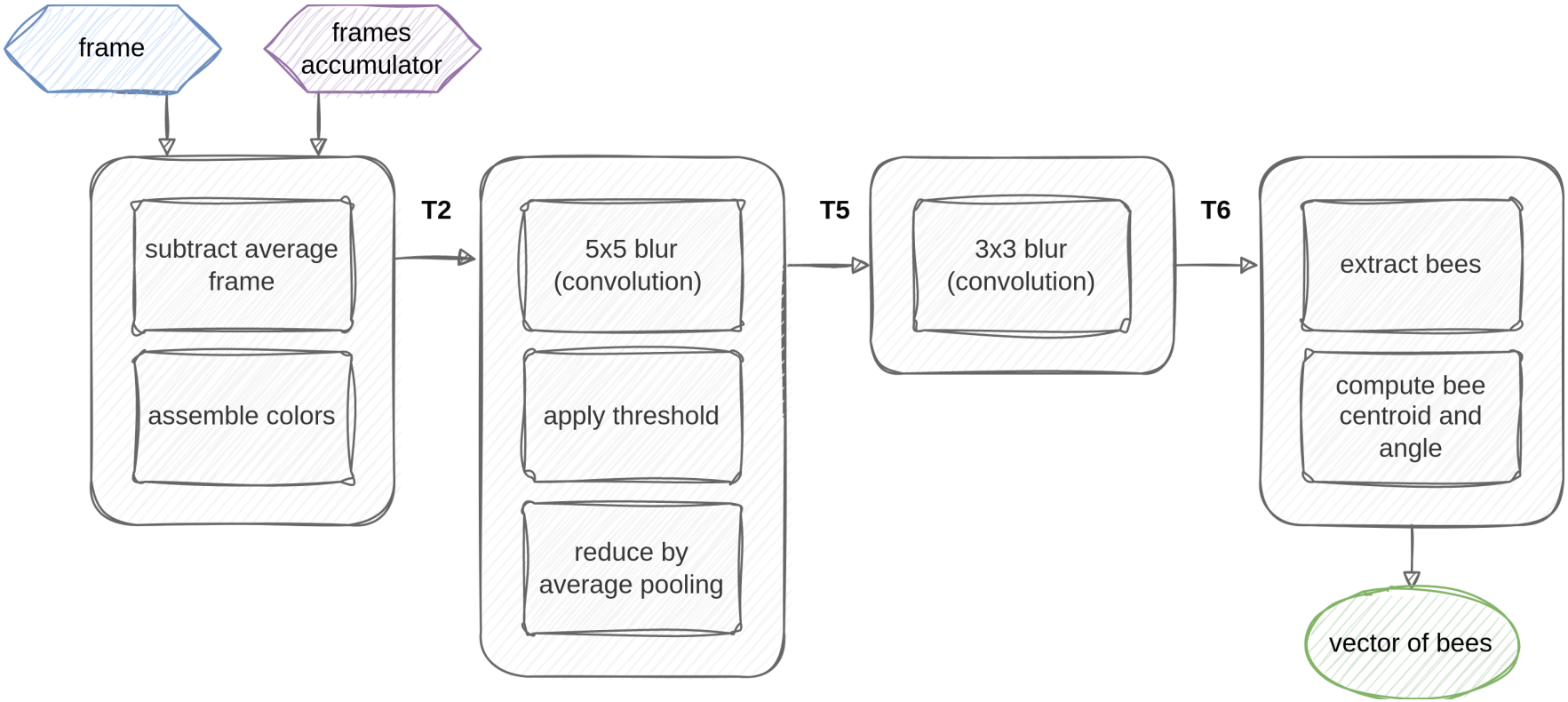
Step 1 is the bottleneck

Optimization will mostly focus on step 1

Step 1 naive architecture



Fuse operations when possible



Optimization of convolution

Switch from Gaussian blur to simple blur:

$$\begin{pmatrix} 1 & 4 & 7 & 1 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 16 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} x \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Optimization of convolution

Switch from Gaussian blur to simple blur:

$$\begin{pmatrix} 1 & 4 & 7 & 1 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 16 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} x \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Reduction of the number of operations:

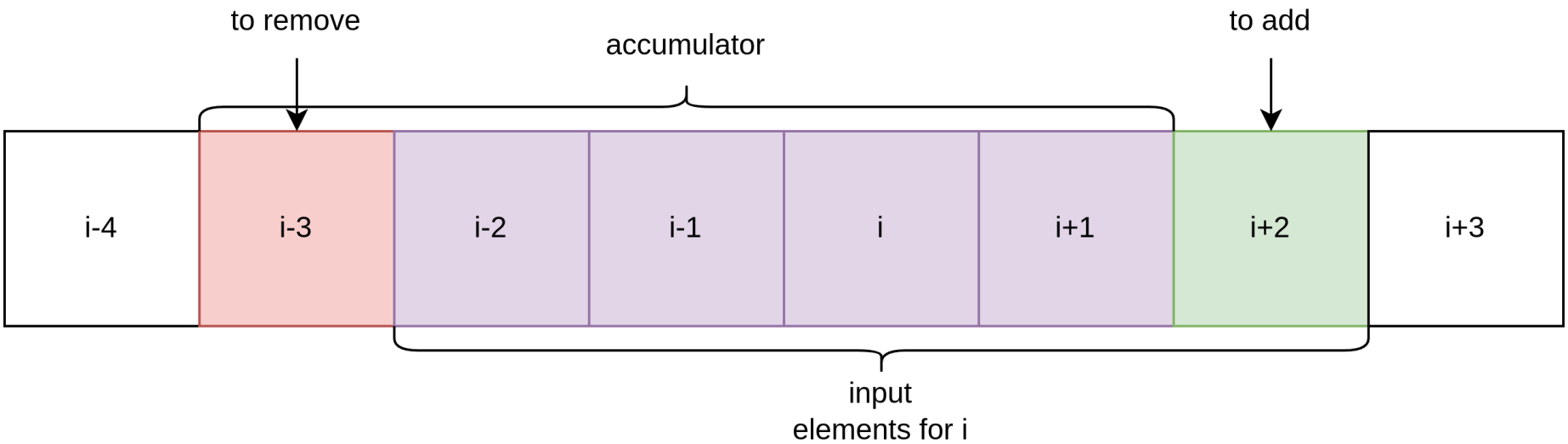
- 5x5 kernel: from 25 to 10: -60%
- 3x3 kernel: from 9 to 6: -33%

Optimization of convolution

The two kernels are only composed of ones:

Common operations between elements

An accumulator and a sliding windows can be used instead of classical convolution



Manual Vectorization

Arm Neon / x86 SSE

- 128-bit vectors
- 8 x `(u)int16_t`

Use of intrinsics instead of assembly:

```
// get input
int16x8_t input = vreinterpretq_s16_u16(inputBloc[inputBlocIndex++]);
// add to accumulator
accumulator = vaddq_s16(accumulator, input);
// put result
outputBloc[outputBlocIndex++] = vreinterpretq_u16_s16(vshrq_n_s16(accumulator, 2));
// update accumulator
accumulator = vsubq_s16(accumulator, previous[0]);
```

```
// get input
__m128i input = inputBloc[inputBlocIndex++];
// add to accumulator
accumulator = _mm_add_epi16(accumulator, input);
// put result / 4 in output bloc
outputBloc[outputBlocIndex++] = _mm_srai_epi16(accumulator, 2);
// update accumulator
accumulator = _mm_sub_epi16(accumulator, previous[0]);
```

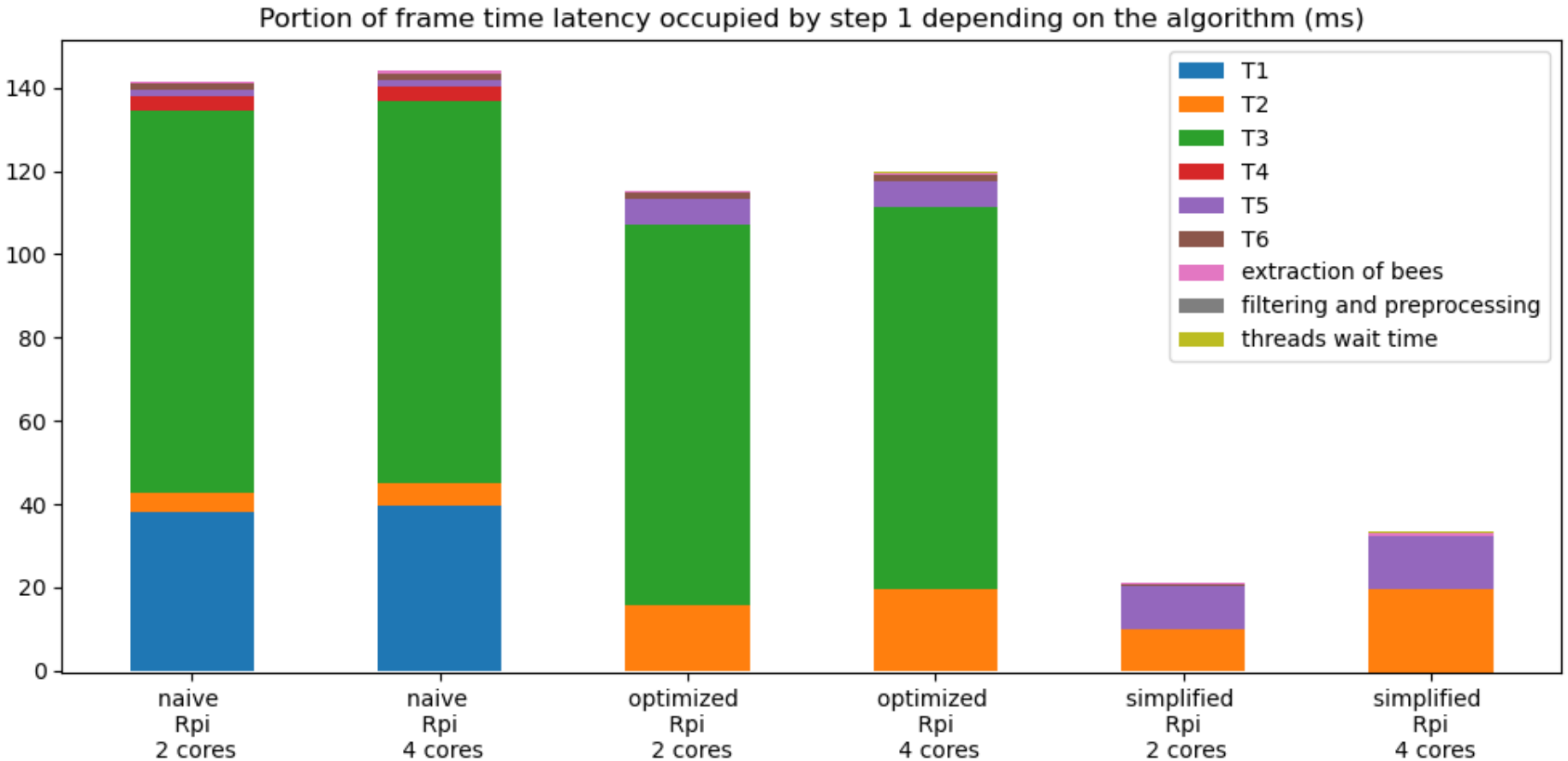
Manual Vectorization challenge

Biggest challenges:

- Intrinsics are not easy to learn
- 8 elements at a time: need to handle corner cases
- Need to load contiguous 128-BIT data for better performance

- De-interleave RGB images
  - Load with de-interleave on Neon, shuffle on SSE
  - Do the arithmetic for each channel
  - Store frames accumulator directly de-interleaved
- Convolution with horizontal kernel:
  - Load a 8x8 bloc (8x 128-BIT loads)
  - Transpose the input bloc
  - Do the arithmetic
  - Transpose the output bloc
  - Store a 8x8 bloc (8x 128-bit stores)

Results


Portion of frame time latency occupied by step 1 depending on the algorithm (ms)
0
20
40
60
80
100
120
140
T1
T2
T3
T4
T5
T6
extraction of bees
filtering and preprocessing
threads wait time
naive Rpi 2 cores
naive Rpi 4 cores
optimized Rpi 2 cores
optimized Rpi 4 cores
simplified Rpi 2 cores
simplified Rpi 4 cores

Step 2: Parallel implementation

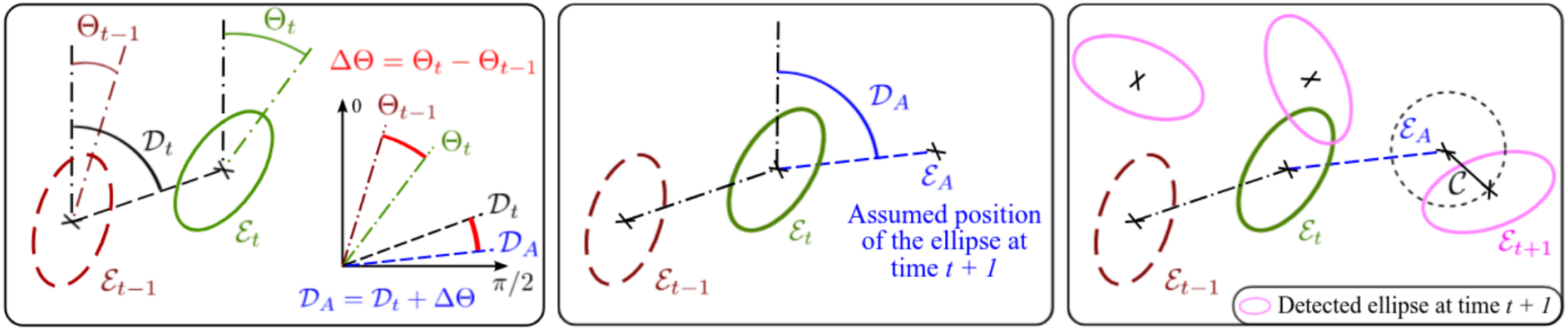


Figure 4: Concordance computation between the assumed ellipse and a detected one.

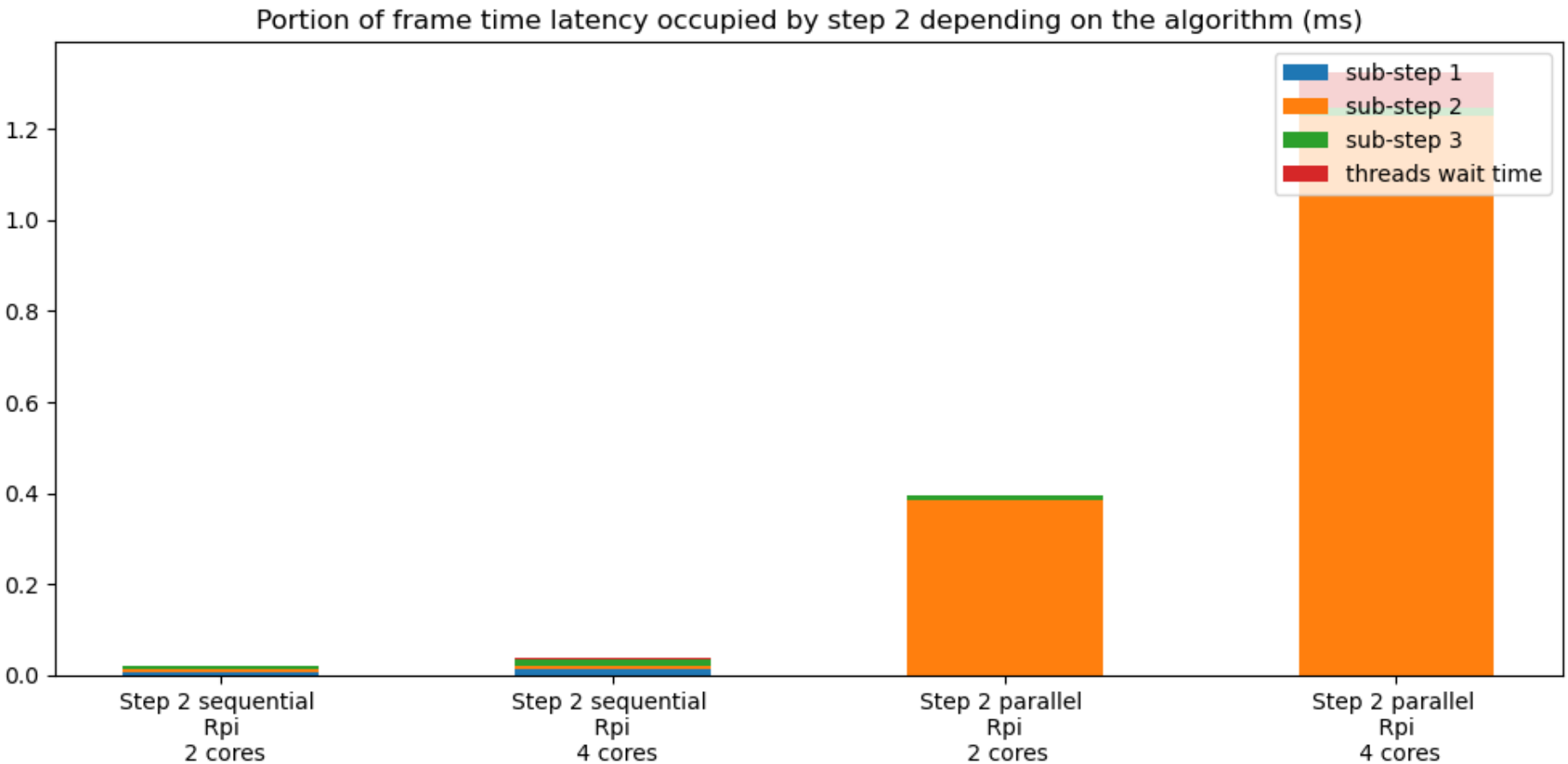
Work distribution

Each thread has a set of trajectories to process

Concurrency management

Use of a vector of Atomics to know if a bee has been picked or not

...but sequential implementation remains better



Work scheduling

Most of Step 1 is (massively) parallel

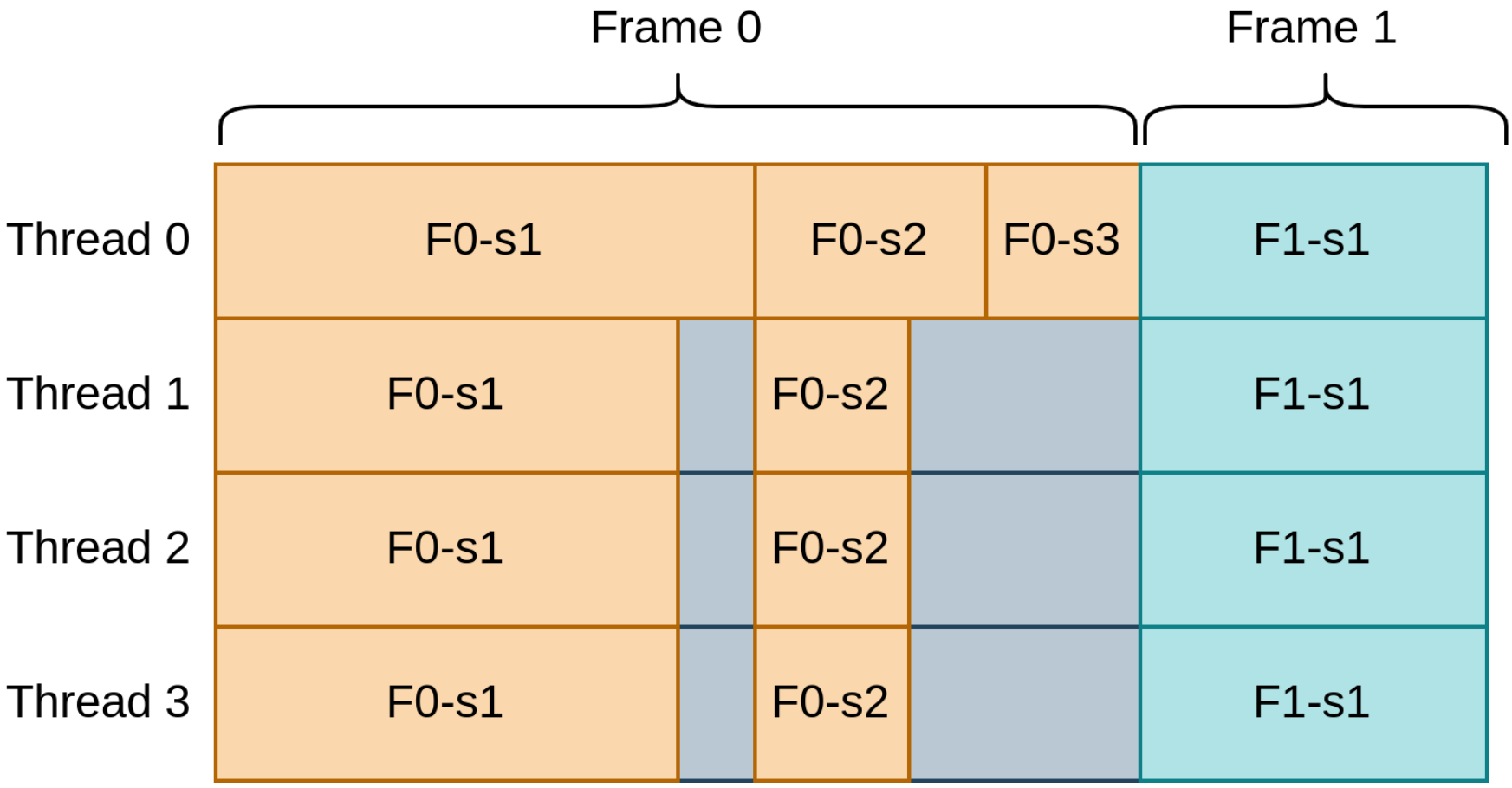
OpenMP is used to provide parallelism with some issues:

- All cores work on memory intensive sections at the same time
- Some sections of the whole algorithm remains sequential

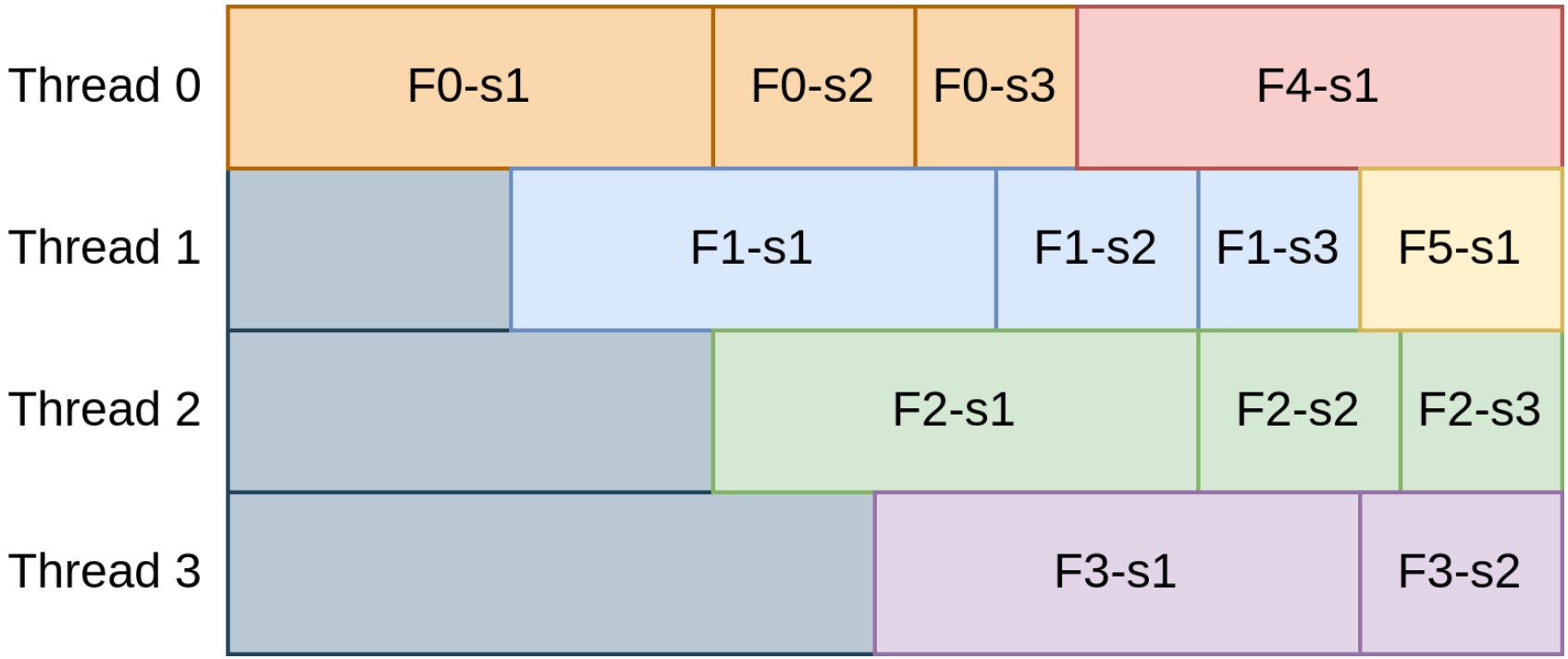
What about frames ?

Most of the operations can be done on multiple frames in parallel

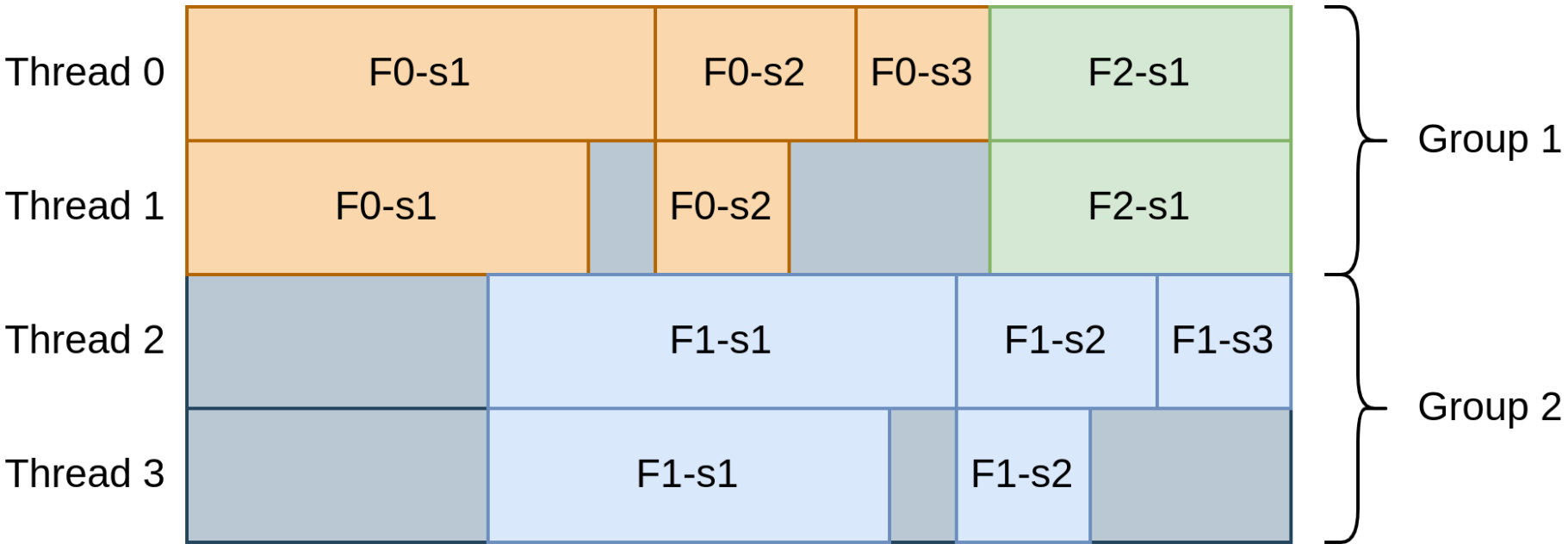
Possible policies: All thread on each frame



Possible policies: One thread per frame



Possible policies: Hybrid approach



Global concurrency challenge

Shared structures

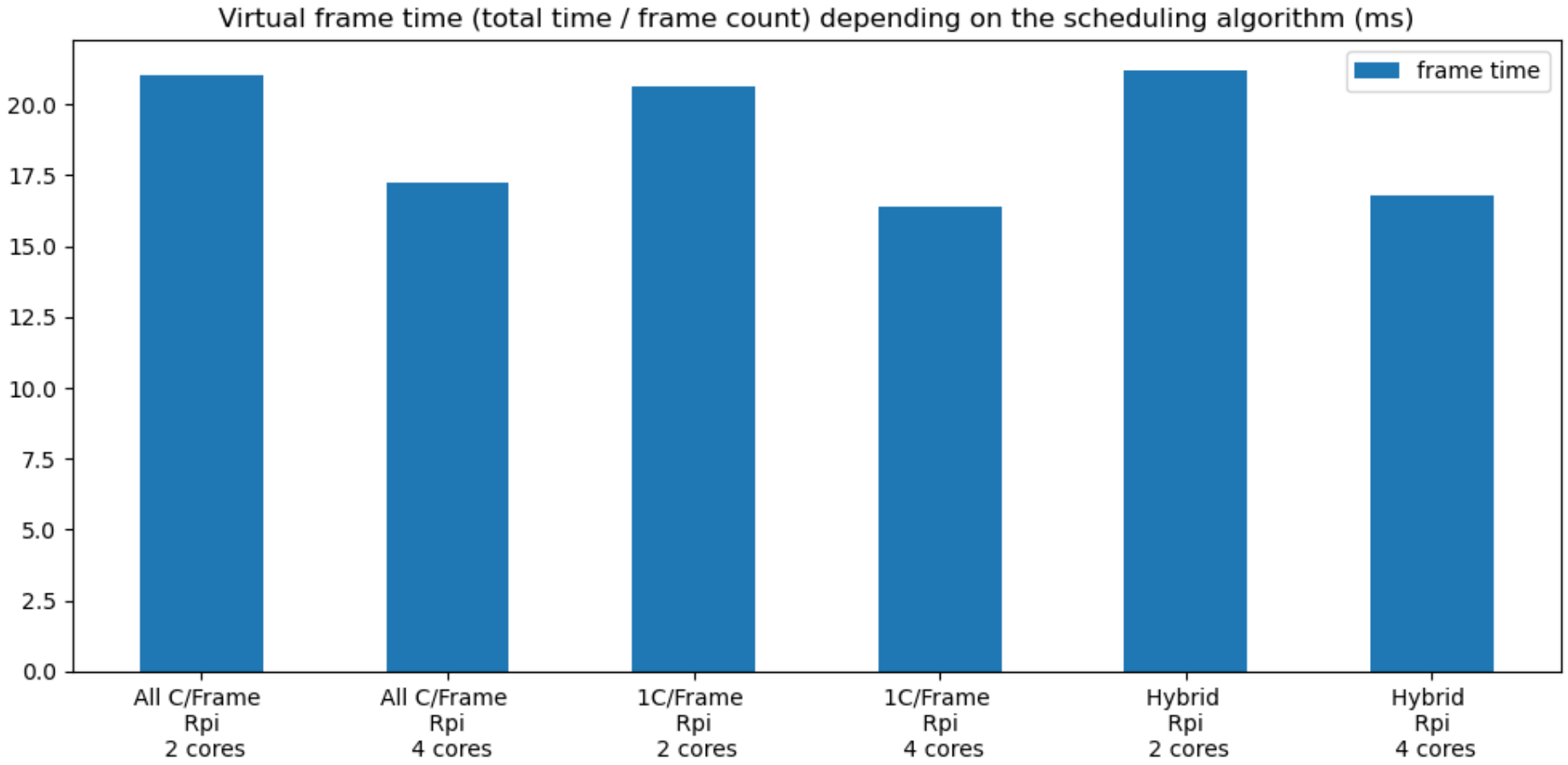
- Frames accumulator: Cut it into N blocs (N is determined by a benchmark) which can be locked by a thread if it's free and already updated with previous frame.
- Tracked bees: Access are done in a critical section

Critical sections

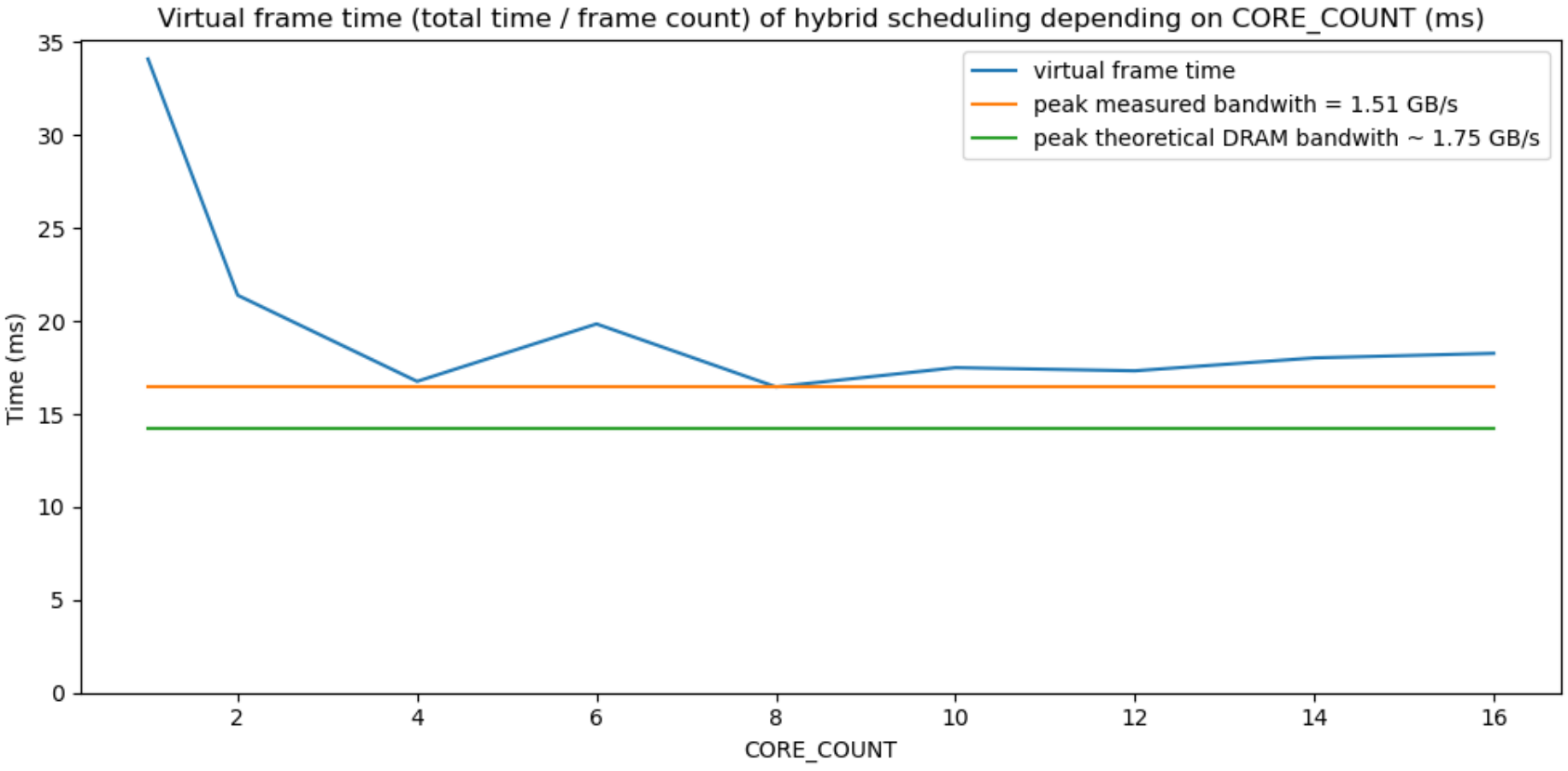
Not simple Mutex, a thread can lock it only if the previous frame has already been processed.

$\Rightarrow$ All global concurrency is managed by the `GlobalConcurrency` class.

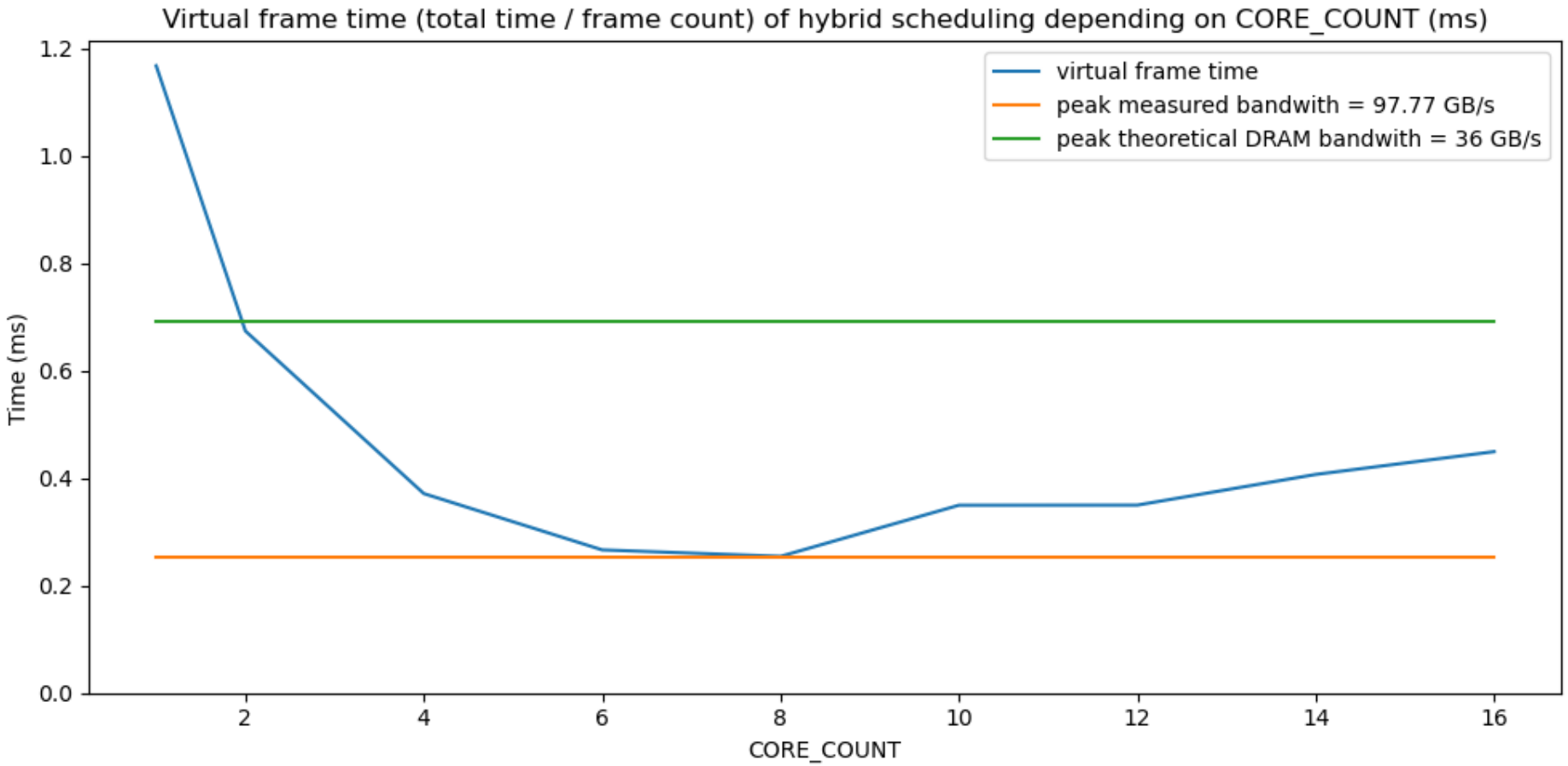
Results


Virtual frame time (total time / frame count) depending on the scheduling algorithm (ms)
frame time
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
All C/Frame Rpi 2 cores
All C/Frame Rpi 4 cores
1C/Frame Rpi 2 cores
1C/Frame Rpi 4 cores
Hybrid Rpi 2 cores
Hybrid Rpi 4 cores

The memory bandwidth issue

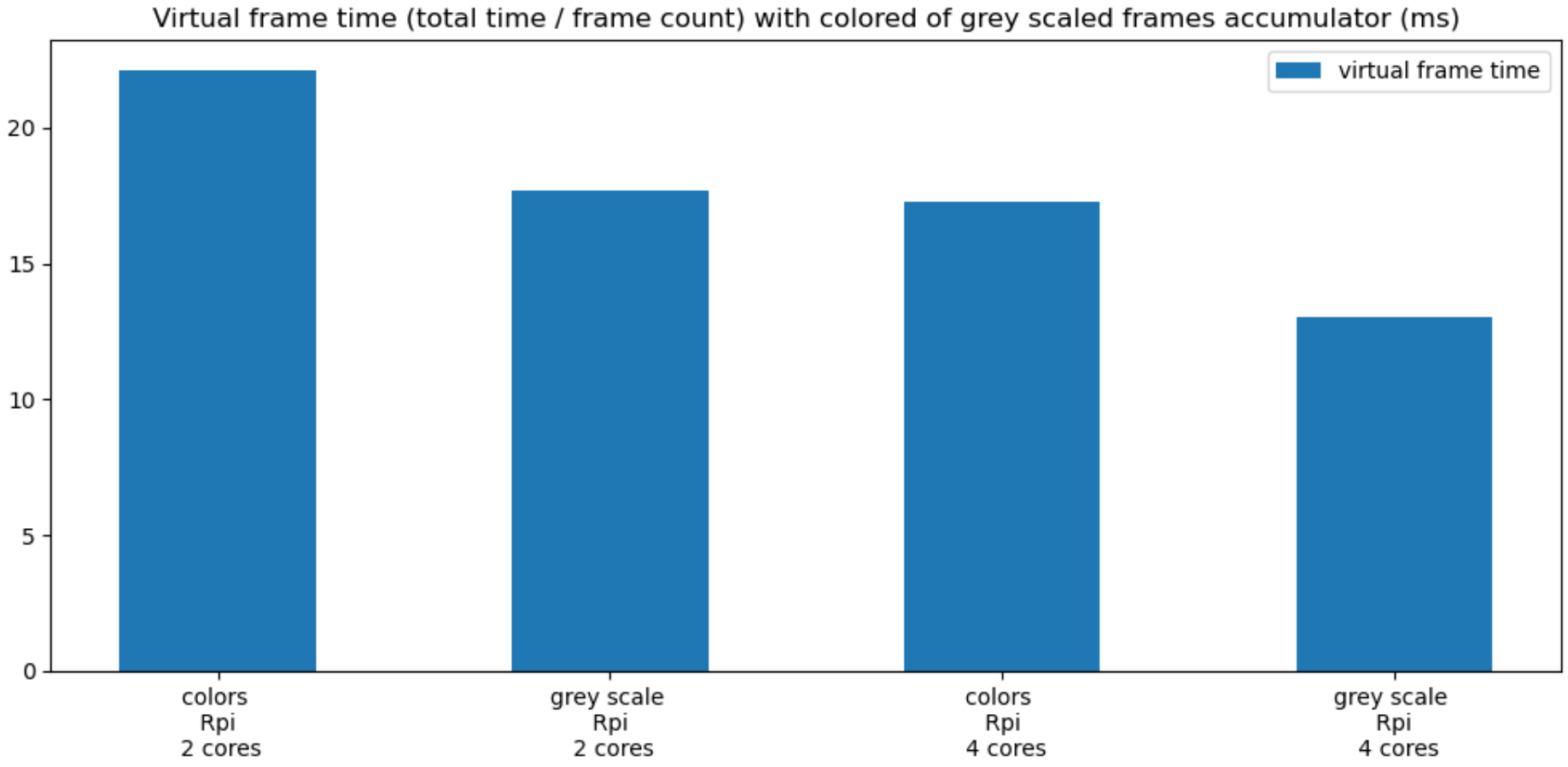


The memory bandwidth issue

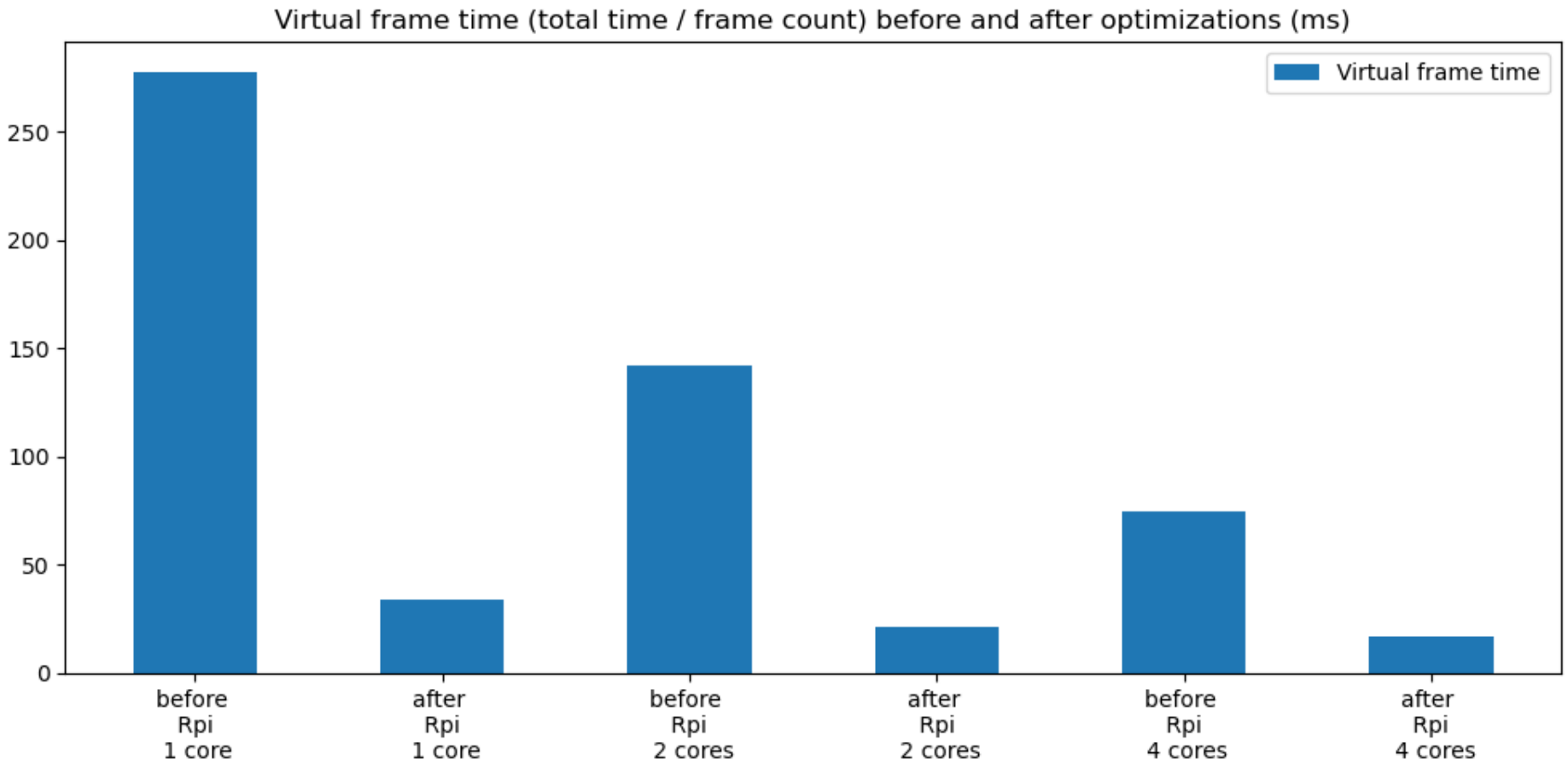


$\Rightarrow$ Cache is big enough to store the frames accumulator

Grey scale processing



Final benchmarks: 60 fps



60 fps on real time with camera seems possible !

Around 75 fps in HD grey scale benchmarks, impact of camera on bandwidth, margin

Future work

- Tweak parameters to have the best accuracy
- Get frame from the camera
- Make it real time compatible
- Evaluate if a CNN model is a good option to provide better accuracy
- Make a working prototype :)

Vielen Dank !