LDPC decoder on Nvidia GPU

•••

Nathan BIETTE • Jean-Raphaël CORNEL • Nicolas LE CLERC • Camille MORIN •
Romain PARRACONE • Samuel PIERRE • Emmanuelle RISSON • Louis ROMAN

1. LDPC decoder algorithm

What is a Parity Check code?

- Code words are of fixed length **n**
- In each code word there are **r** check symbols and **n-r** information symbols
- Each symbol in a code word satisfies **r** equations
- Parity Check Matrix

$n = 7 \quad r = 3$

$c=c_1c_2c_3c_4c_5c_6c_7$

$$c_1 \oplus c_2 \oplus c_5 = 0$$
$$c_1 \oplus c_2 \oplus c_4 \oplus c_6 = 0$$
$$c_1 \oplus c_3 \oplus c_4 \oplus c_7 = 0$$

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

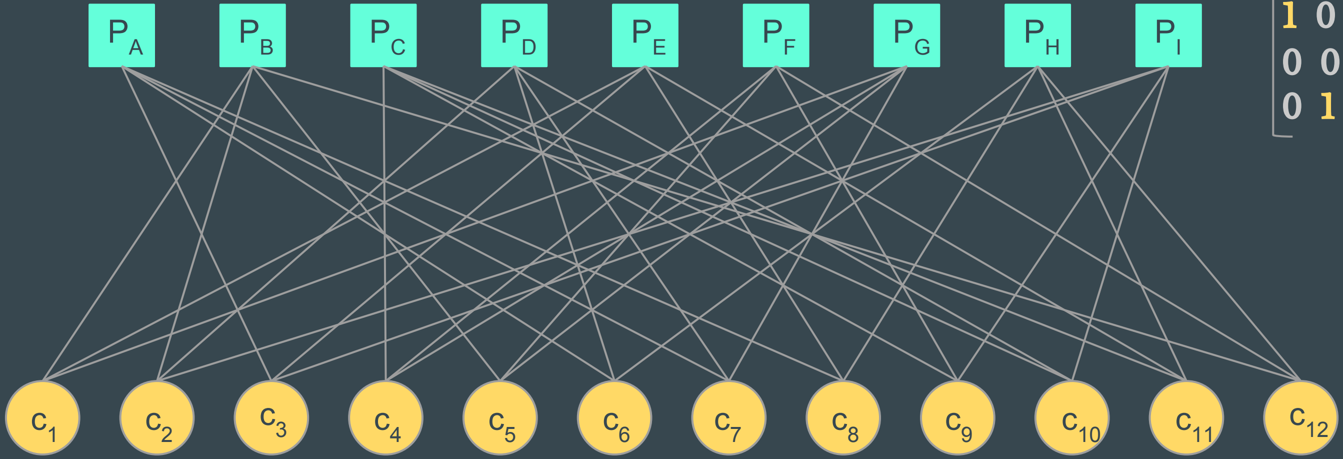
$M.c^T=0$

What is a Low-Density Parity Check code?

- Every code digit is in the same number of equations
- Every equation has the same number

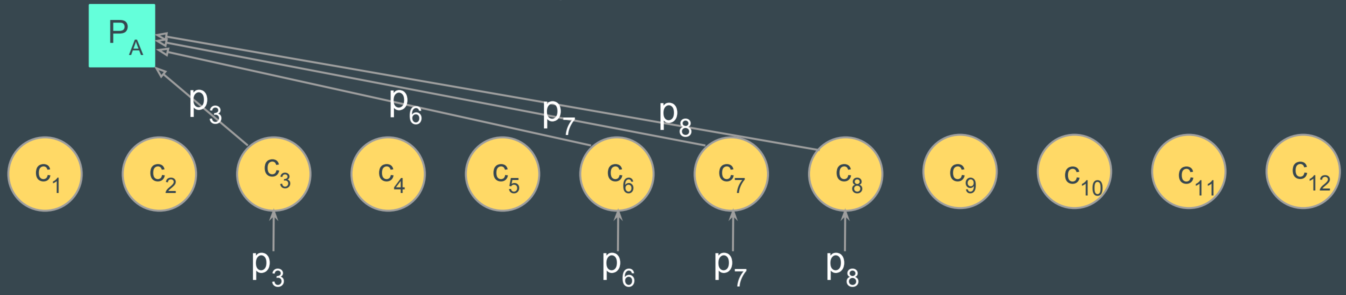
$$M = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Here is a graphical representation:



Estimation of Probabilities by Parity Nodes

For example, 1st equation: $c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$



p_i: estimation of $P[c_i=1]$

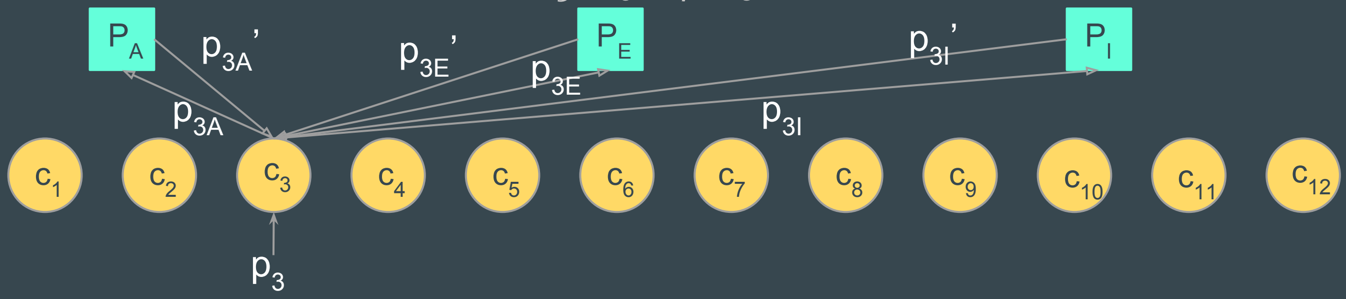
The parity node:

- receives the estimate p_3 , p_6 , p_7 , p_8
- calculates new estimate p_{3A} , p_{6A} , p_{7A} , p_{8A}

$$\boxed{p_{3A} = p_6 p_7 p_8 + p_6(1-p_7)(1-p_8) + p_7(1-p_6)(1-p_8) + p_8(1-p_6)(1-p_7)}$$

Estimation of Probabilities by Bit Nodes

For example, 1st equation: $c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$



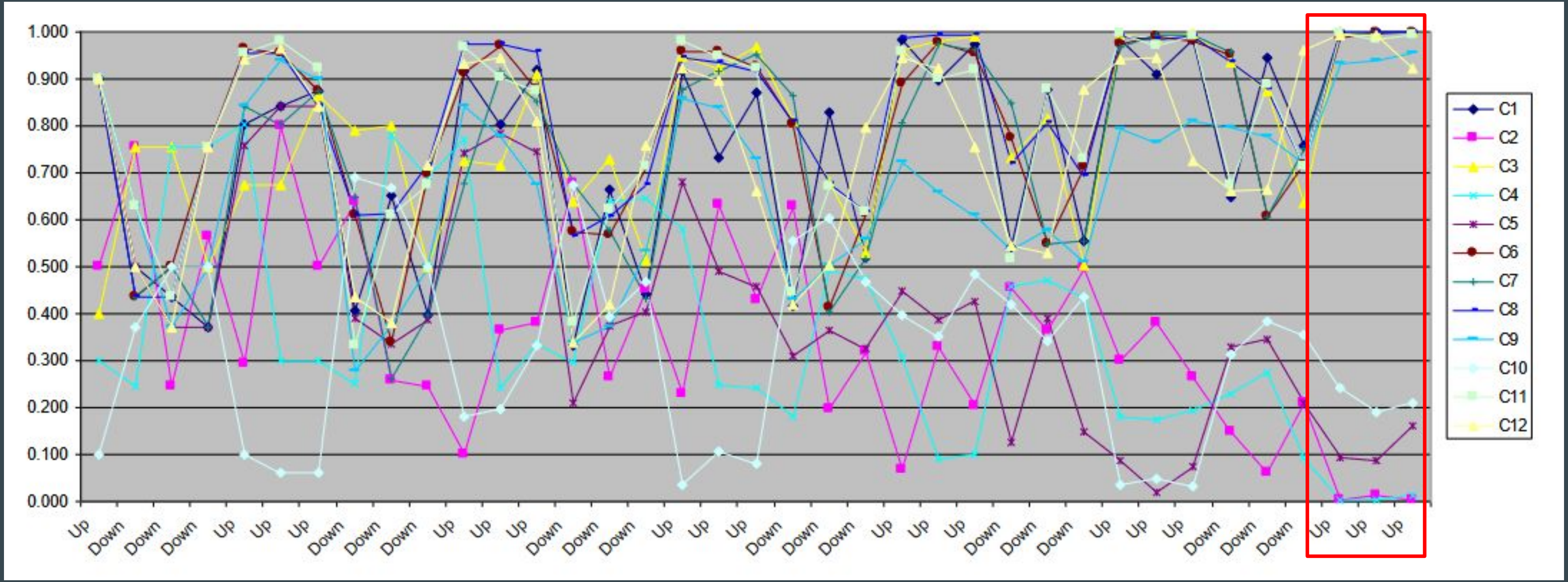
p_i: estimation of $P[c_i=1]$

The bit node:

- receives three estimate p_{3A}', p_{3E}' and p_{3I}' from P_A , P_E and P_I
- calculates three new estimate p_{3A}, p_{3E} and p_{3I} for each parity node P_A , P_E and P_I

$$p_{3A}=Kp_3p_{3E}p_{3I} \qquad p_{3E}=Kp_3p_{3A}p_{3I} \qquad p_{3I}=Kp_3p_{3A}p_{3E}$$

Iterating the process



p_i converge => good estimation of bit node value c_i (hard decision)

To sum up: Low-Density Parity-Check decoder

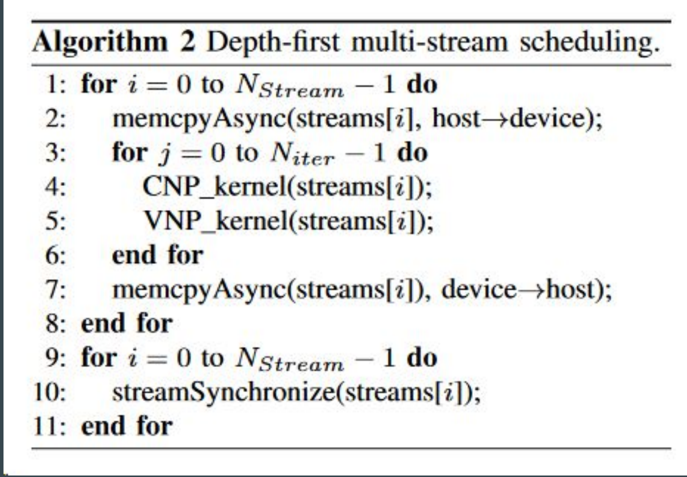
- Error-correction code
- Near-capacity performance
    - Needs:
        - High throughput
        - Multi-standard support
        - Flexibility and scalability

2. Adjustments of the algorithm

•••

Why use a GPU for the LDPC decoder?

- To achieve high throughput
  - GPU (Graphics Processing Unit)
    - high computational throughput
    - several processors working in parallel
  - LDPC
    - massively parallel algorithm
    - clear architecture to divide the operations



Algorithm 2 Depth-first multi-stream scheduling.

```
for $i = 0$ to $N_{Stream} - 1$ do
    memcpyAsync(streams[$i$], host→device);
    for $j = 0$ to $N_{iter} - 1$ do
        CNP_kernel(streams[$i$]);
        VNP_kernel(streams[$i$]);
    end for
    memcpyAsync(streams[$i$]), device→host);
end for
for $i = 0$ to $N_{Stream} - 1$ do
    streamSynchronize(streams[$i$]);
end for
```

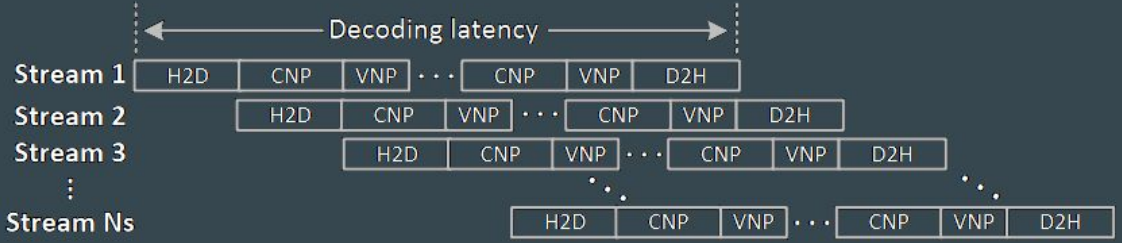
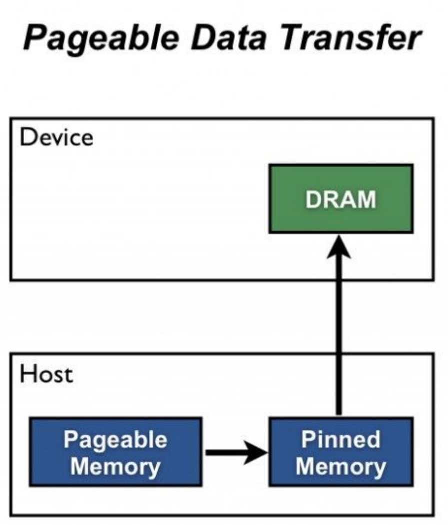


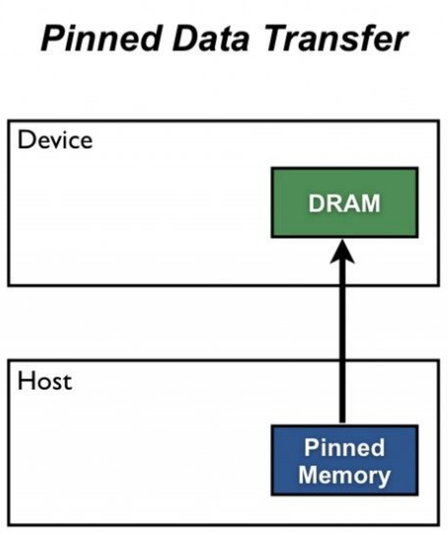
Fig. 4. Multi-stream LDPC decoding.

Pinned vs. non-pinned memory

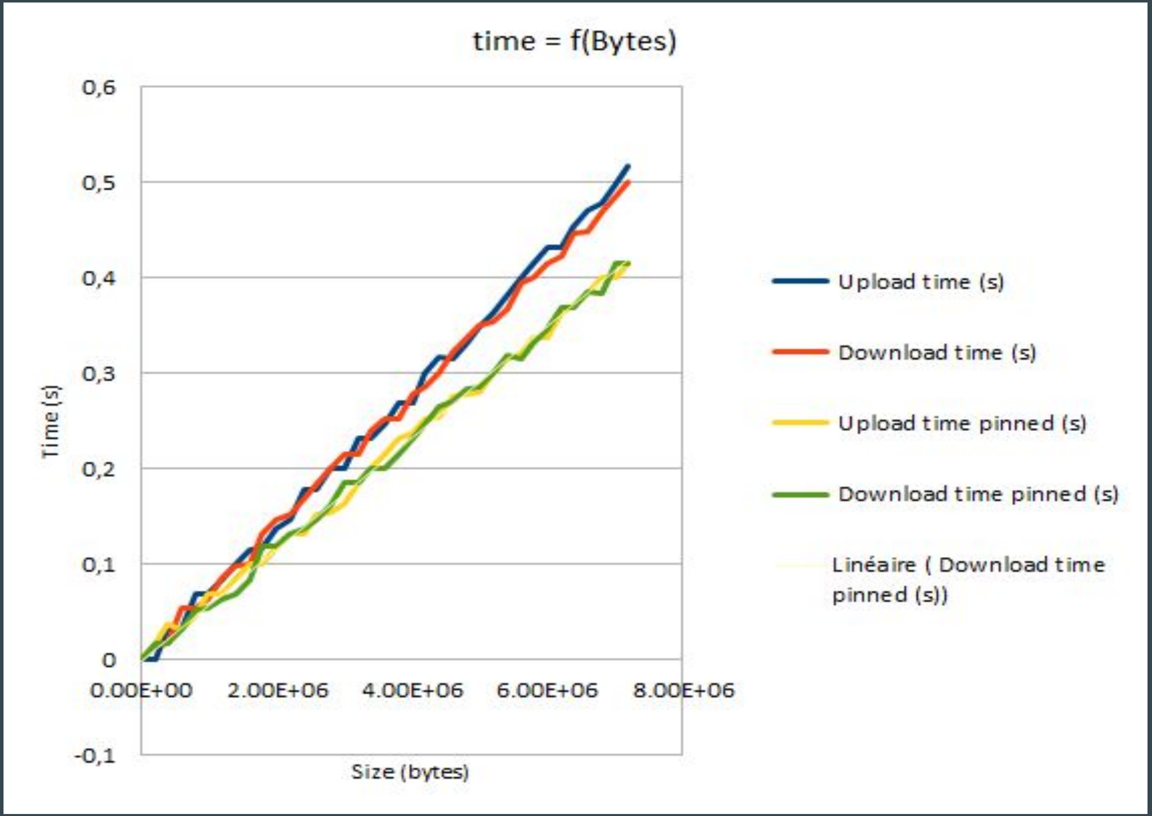
- Pinned memory is more efficient to achieve data transfer to GPU
- We put the data in a special memory space such that it is directly accessible by the DMA without intervention of the CPU
- If the data is not stored in pinned memory, the program will itself copy data in a pinned memory before sending it to the GPU
- We can directly allocate the arrays in pinned host memory to reduce time of the copy

Use: cudaMallocHost()

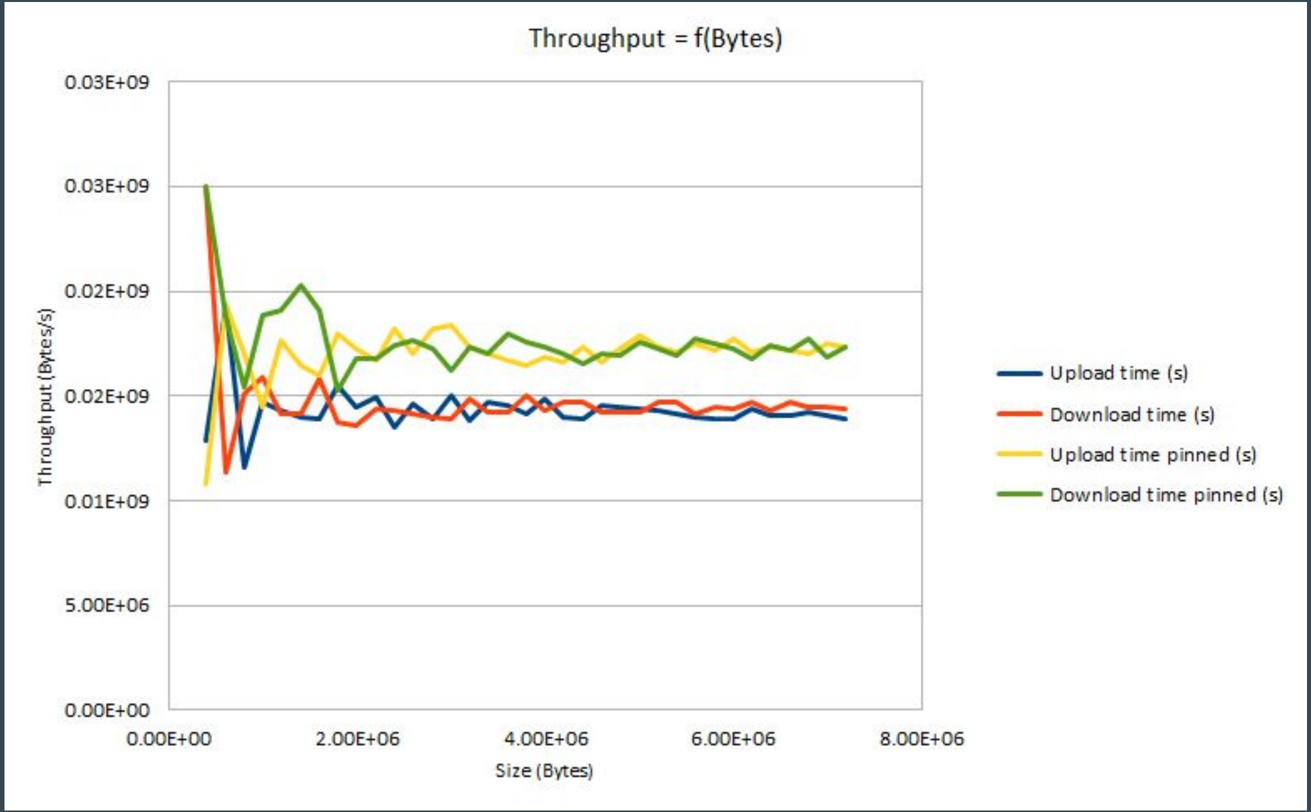




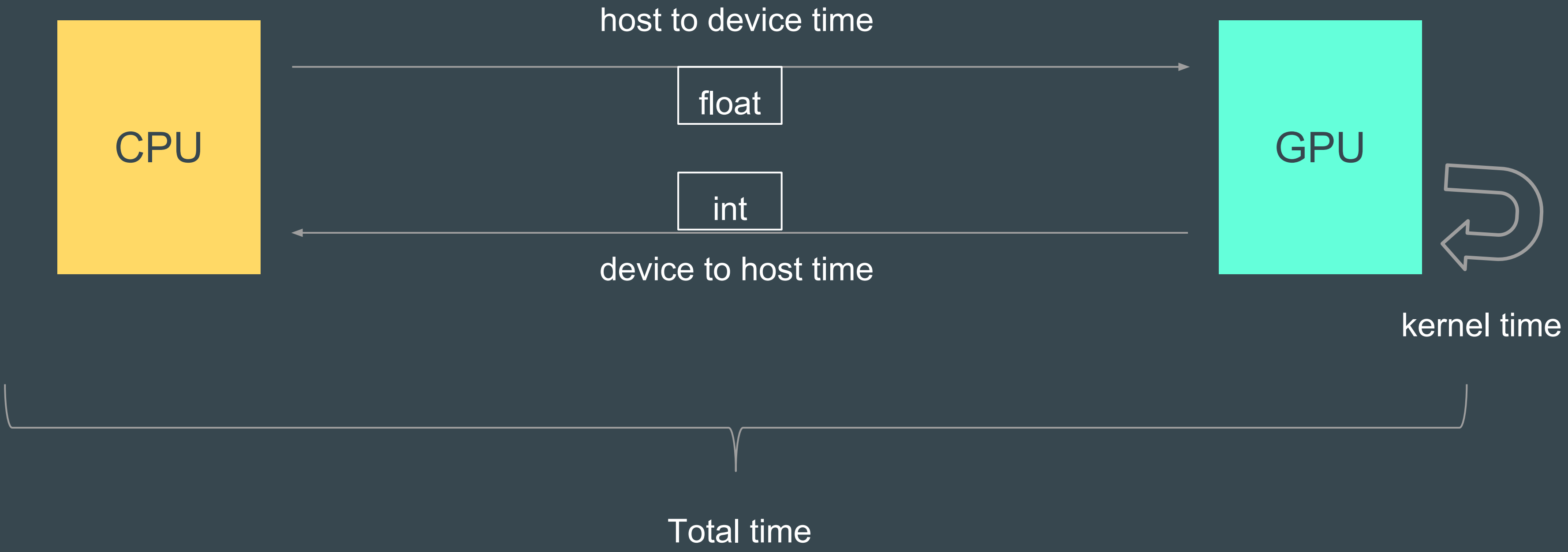
Comparison between the two memories



Comparison between the two memories



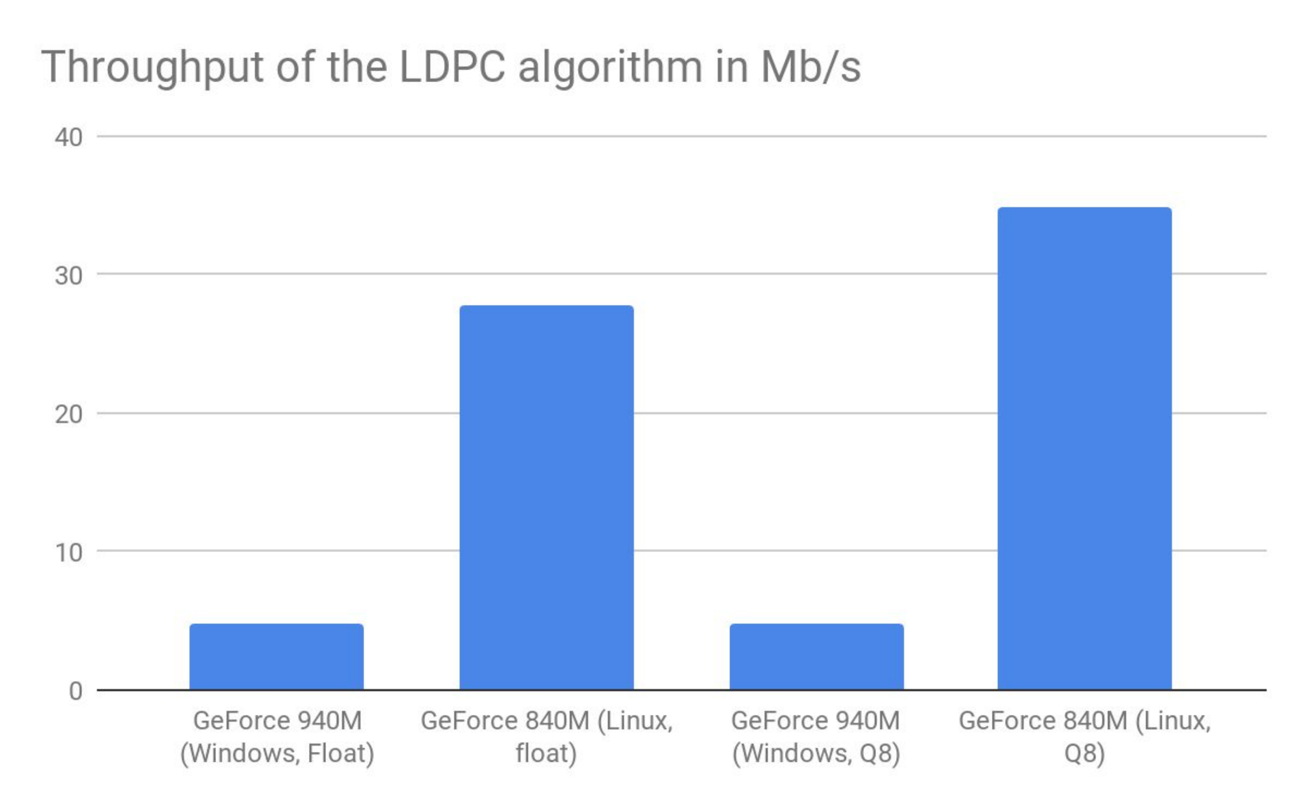
Timing measurements



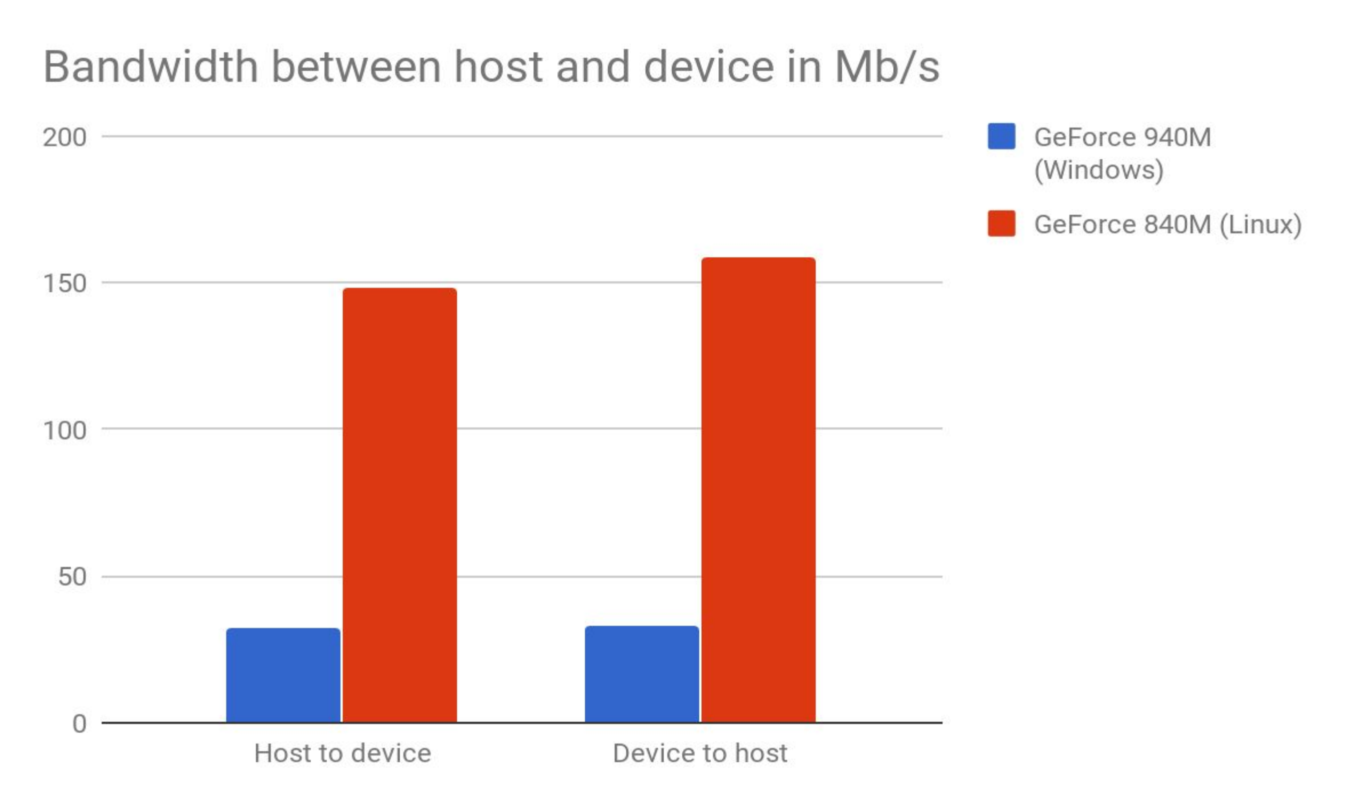
3. Benchmark results

•••

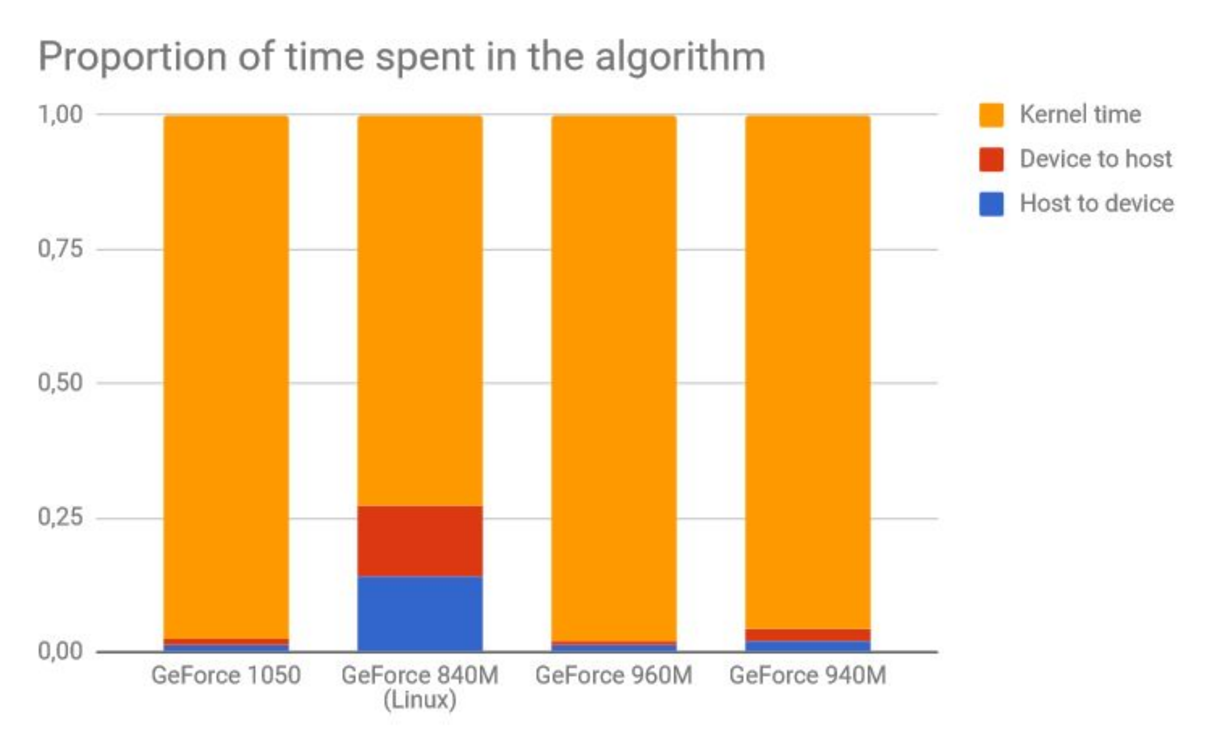
Measures


Throughput of the LDPC algorithm in Mb/s
40
30
20
10
0
GeForce 940M (Windows, Float)
GeForce 840M (Linux, float)
GeForce 940M (Windows, Q8)
GeForce 840M (Linux, Q8)

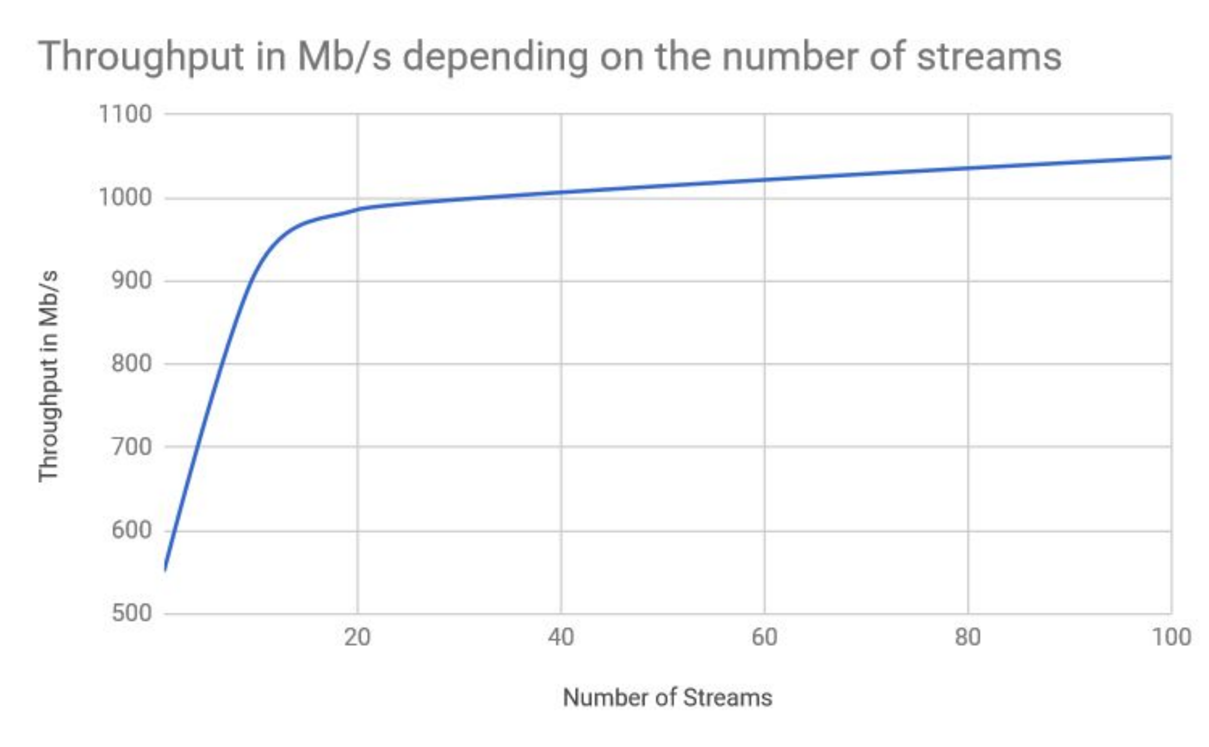
Measures


Bandwidth between host and device in Mb/s
200
150
100
50
0
Host to device
Device to host
🟦 GeForce 940M (Windows)
🟥 GeForce 840M (Linux)

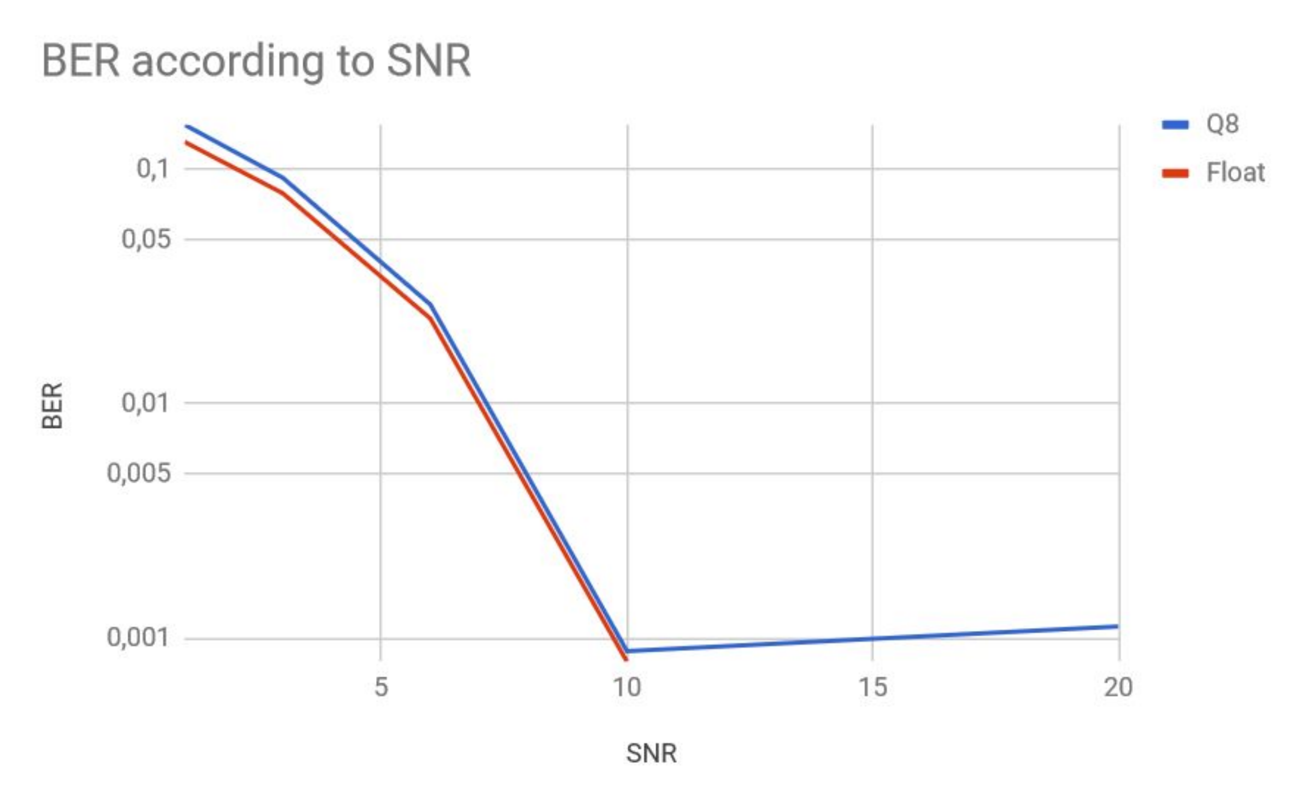
Measures


Proportion of time spent in the algorithm
1,00
0,75
0,50
0,25
0,00
Kernel time
Device to host
Host to device
GeForce 1050
GeForce 840M (Linux)
GeForce 960M
GeForce 940M

Measures


Throughput in Mb/s depending on the number of streams
Throughput in Mb/s
1100
1000
900
800
700
600
500
20
40
60
80
100
Number of Streams

Measures


BER according to SNR
Q8
Float
BER
0,1
0,05
0,01
0,005
0,001
5
10
15
20
SNR