Image Skeletonisation

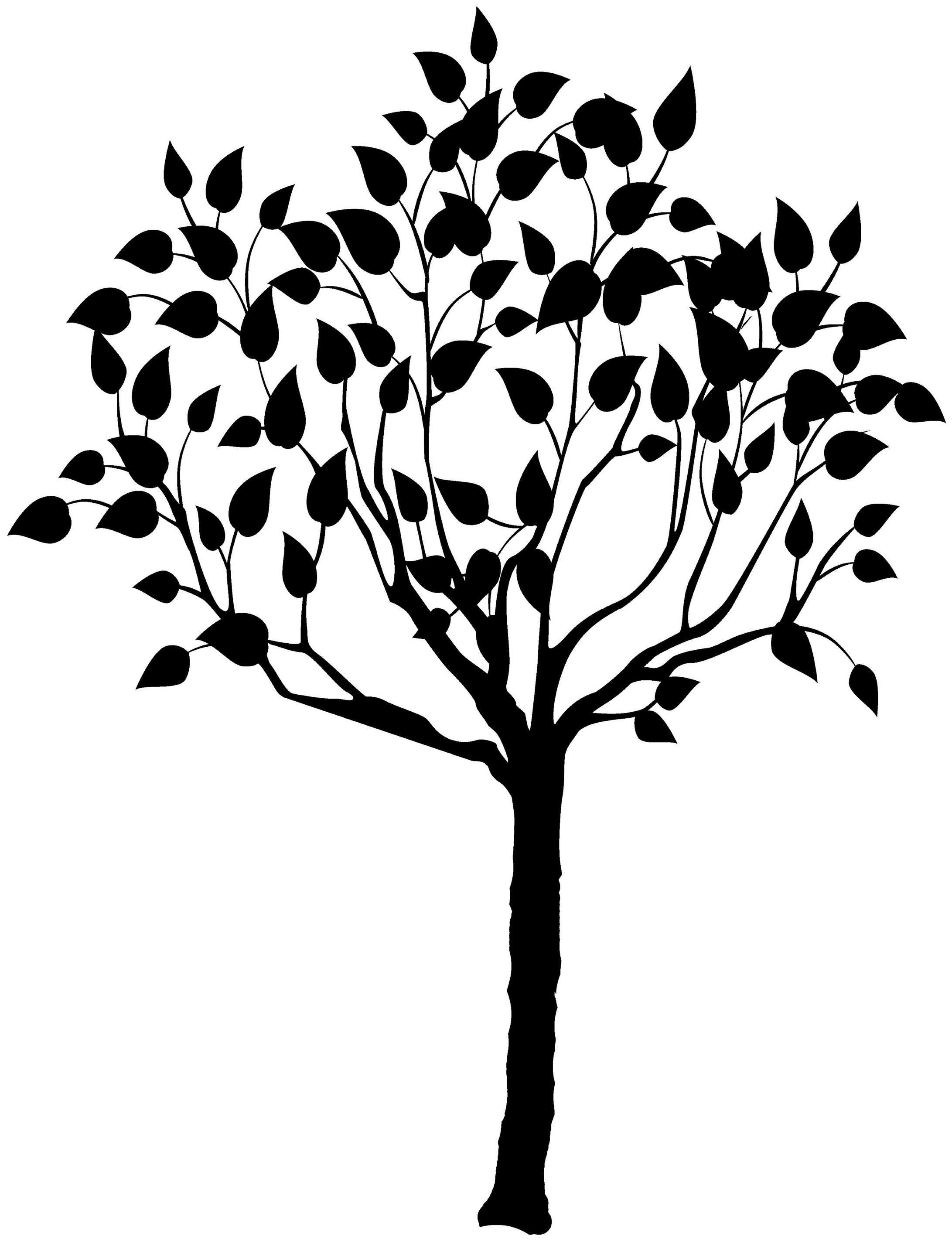
A performance comparison with Thrust

Fall 2014

Marco Antognini

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

What is it?

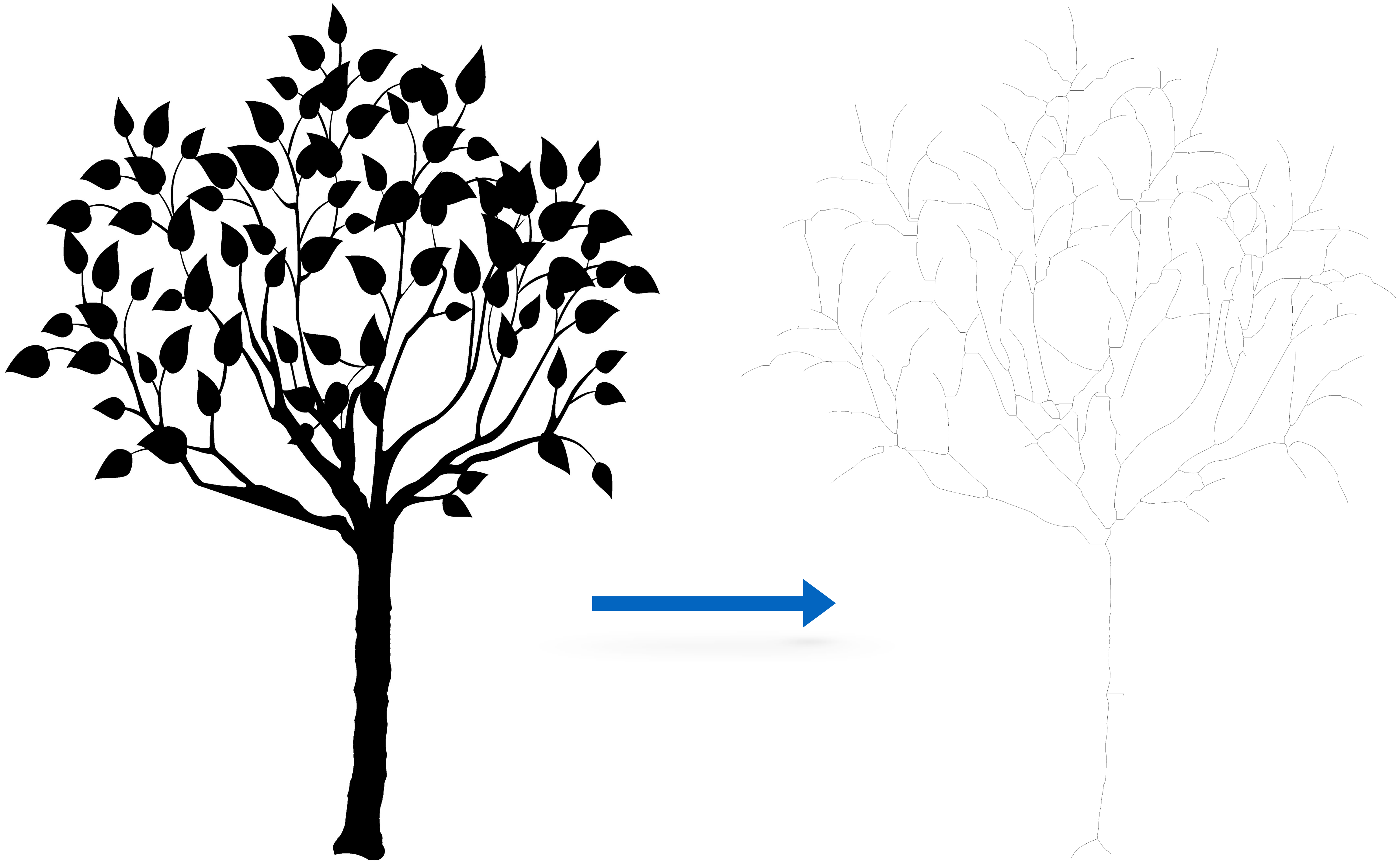


A method to reduce
a **binary image** to
its **minimal representation**

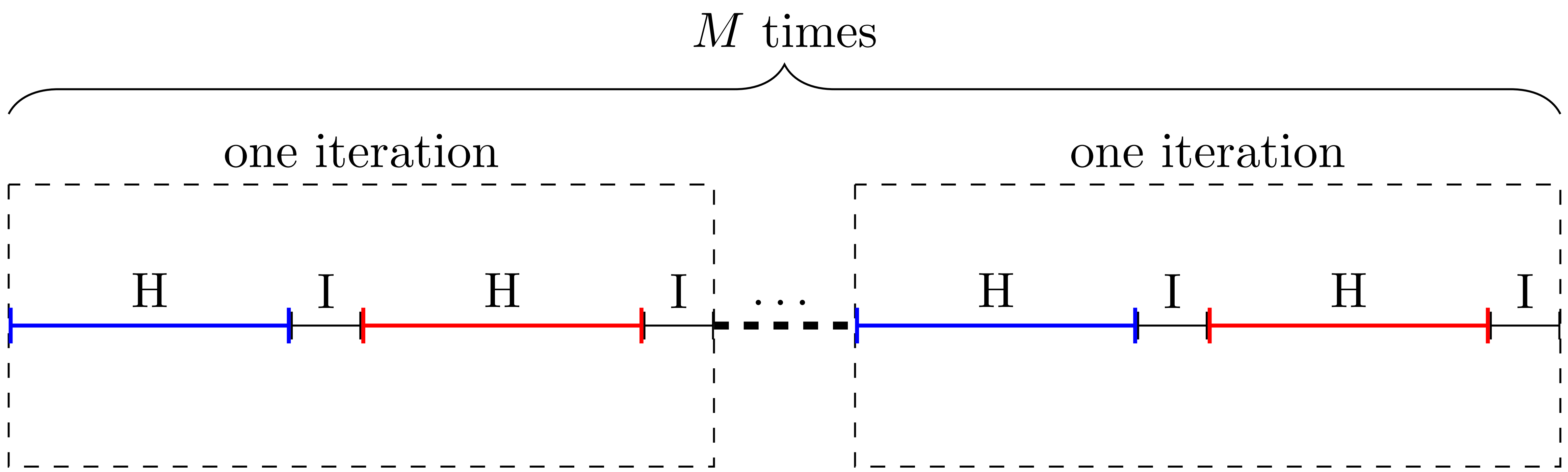
Useful for

- blood analysis
- classification of fingerprints
- text recognition
- geometrical shape analysis
- …

What is it?



Serial Algorithm



H a sub-iteration

- blue ⇒ first sub-cycle
- red ⇒ second sub-cycle

I the marked pixels are deleted

After *M* **iterations** the image has converged to its skeleton

For each sub-cycle

For each pixel p in the image

- If some predicates on the 8 direct neighbours don't hold
- then p is **marked** for deletion

M depends on the shape

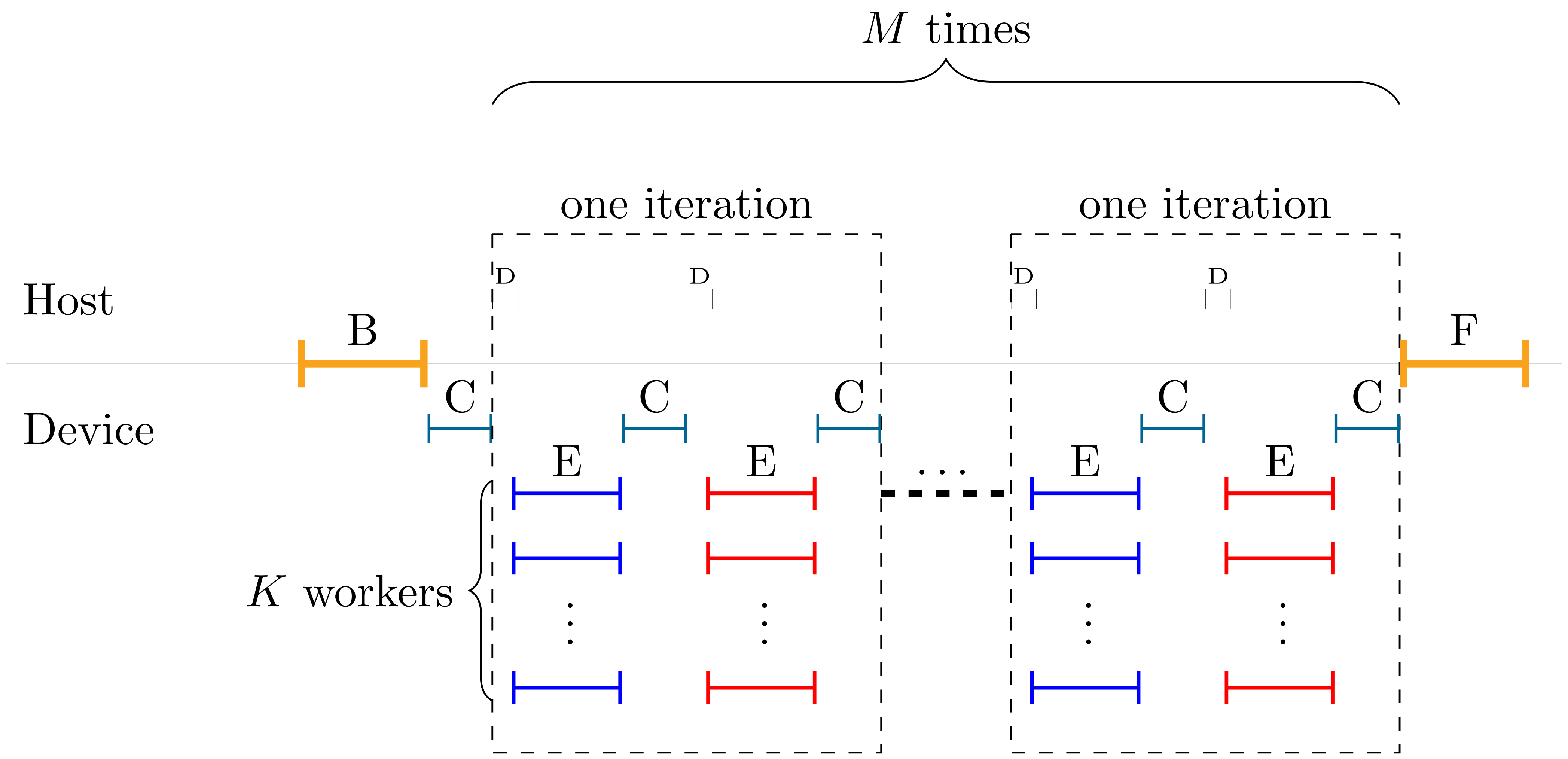
- the bigger the widest arm of the shape is, the bigger *M* will be

Implementation: Thrust

- ➡ A C++03 general purpose library by Nvidia
- ➡ GPU and CPU backends **with the same code**
  - ‣ CUDA, OpenMP, Threading Building Blocks
- ➡ API very similar to STL
- ➡ Based on high level **functional algorithms**
  - ‣ *transform, reduce, copy, count, search, partition, …*

- ➡ But can't do everything…
  - ‣ e.g. matrix multiplication

Parallel Algorithm



Each pixel can be marked regardless of the other pixels in a sub-cycle

➡ We use *K* workers to split the job

➡ With double buffering (buffer ❶ and ❷)

➡ Instead of marking we delete in ❷

B send data to *device* memory, buffer ❶

C synchronise buffers ❶ and ❷

D/E launch kernel function on *device*

- blue ⇒ first sub-cycle, red ⇒ second sub-cycle

F copy data back to *host*

Theoretical Speedup

Work split on *K* workers

- ➡ CUDA on GeForce GT 650M, Kepler GK110 architecture
    - 2 Multiprocessors (MP) @ 900 MHz with 192 cores each
    - Max 2048 threads per MP understand: resource available for max 2048 threads
    - But only 4 warps executing instructions simultaneously per MP
    - $\Longrightarrow$ 2 MP * 4 warps * 32 threads / warps = ***K* = 256**
- ➡ OMP/TBB on Core i7 3720QM
    - 4 cores @ 2600 MHz with Hyper-Threading Technology
    - $\Longrightarrow$ ***K* = 8**

Clock ratio

- $\Longrightarrow \lambda \approx 2.88$

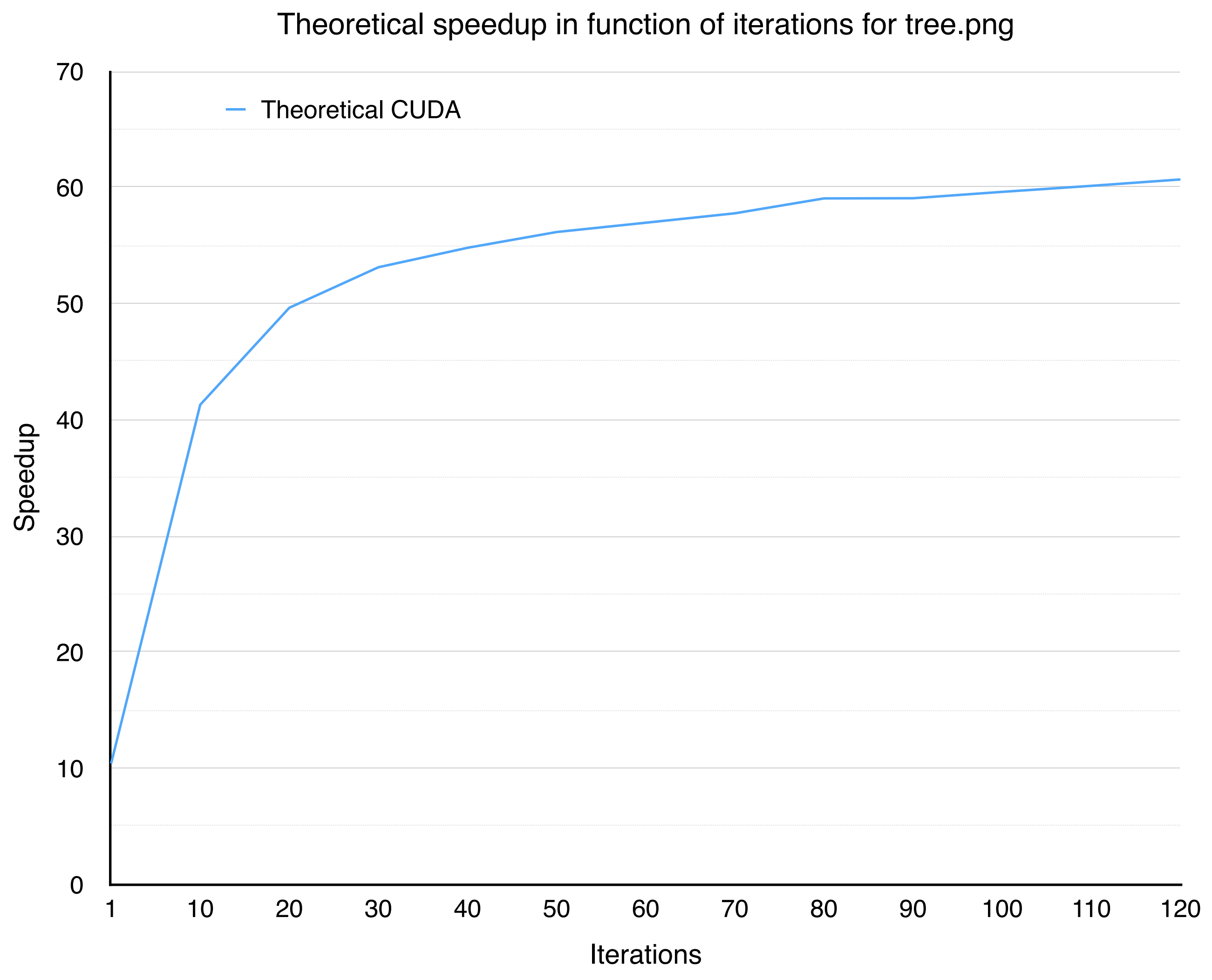
Memory speed depends on transferred size

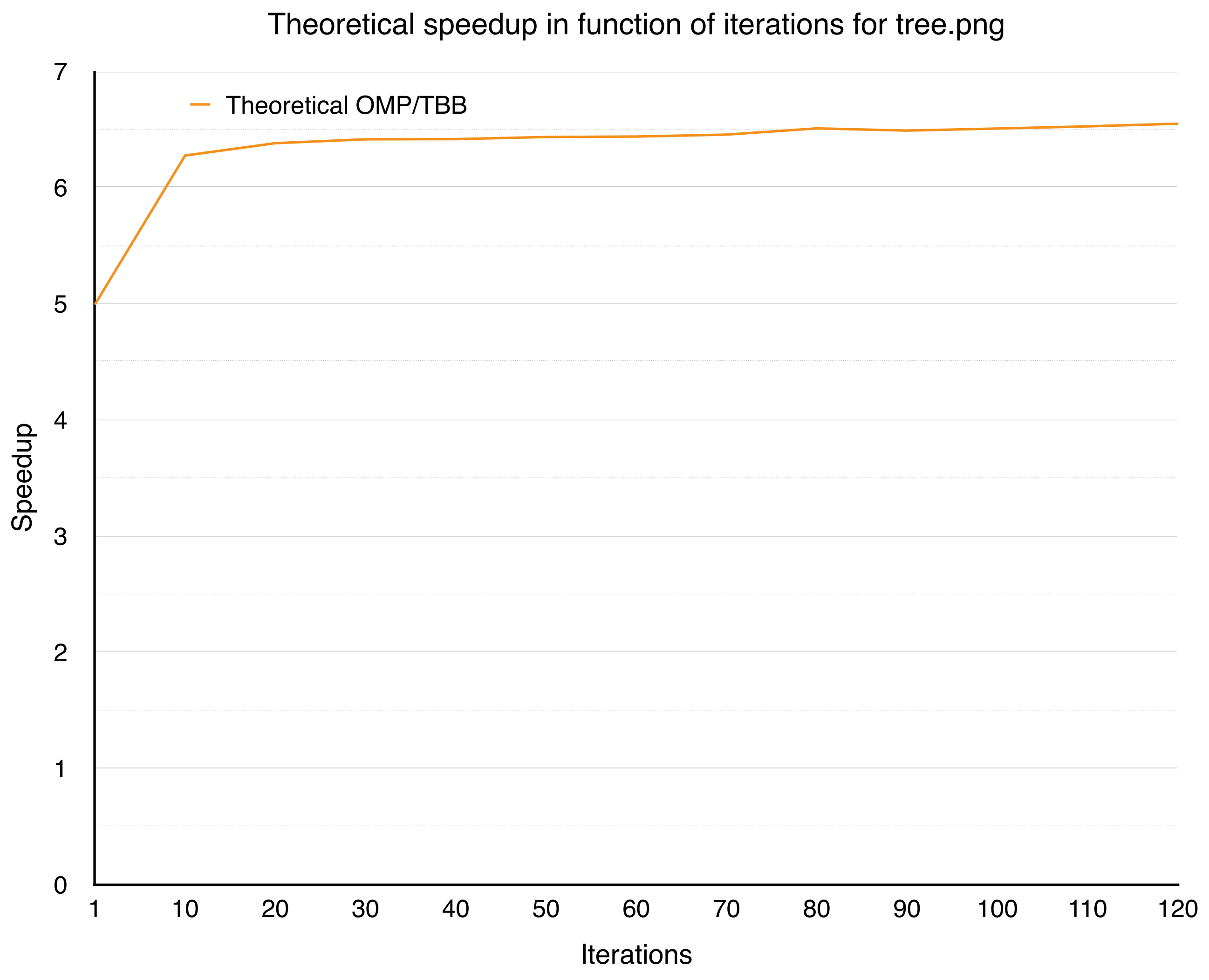
- See final report

Theoretical Speedup

$$Speedup(N,M,K) = \frac{t_{ser}}{t_{par}} = \frac{t_{ser}}{\frac{N}{H{\rightarrow}D_s} + \frac{N}{D{\leftrightarrow}D_s} + 2M\left(\frac{\lambda}{K}t_{sub} + \frac{N}{D{\leftrightarrow}D_s}\right) + \frac{N}{D{\rightarrow}H_s}}$$

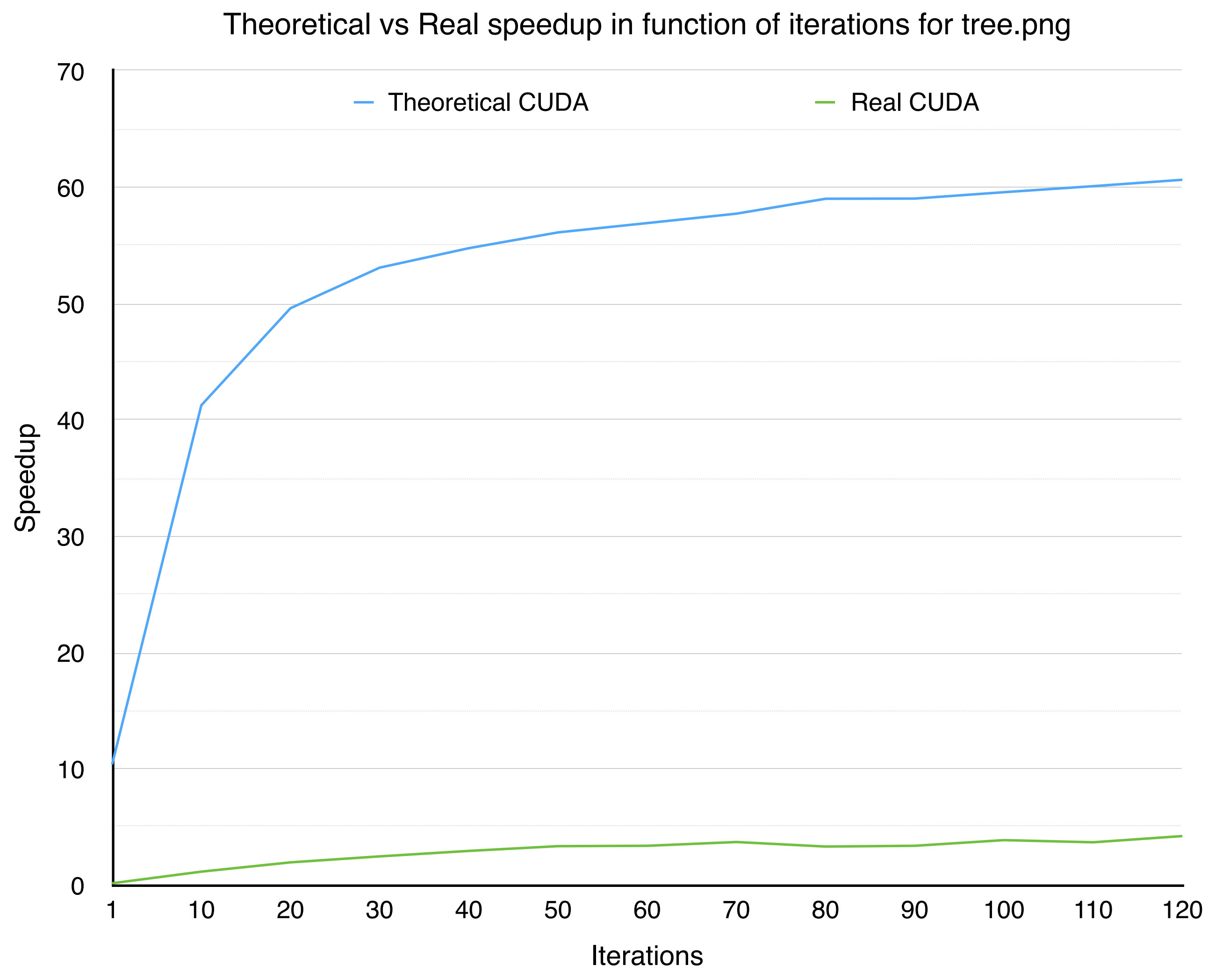
- The number of pixels, *N*
- The number of required iterations, *M*
- The number of worker threads, *K*

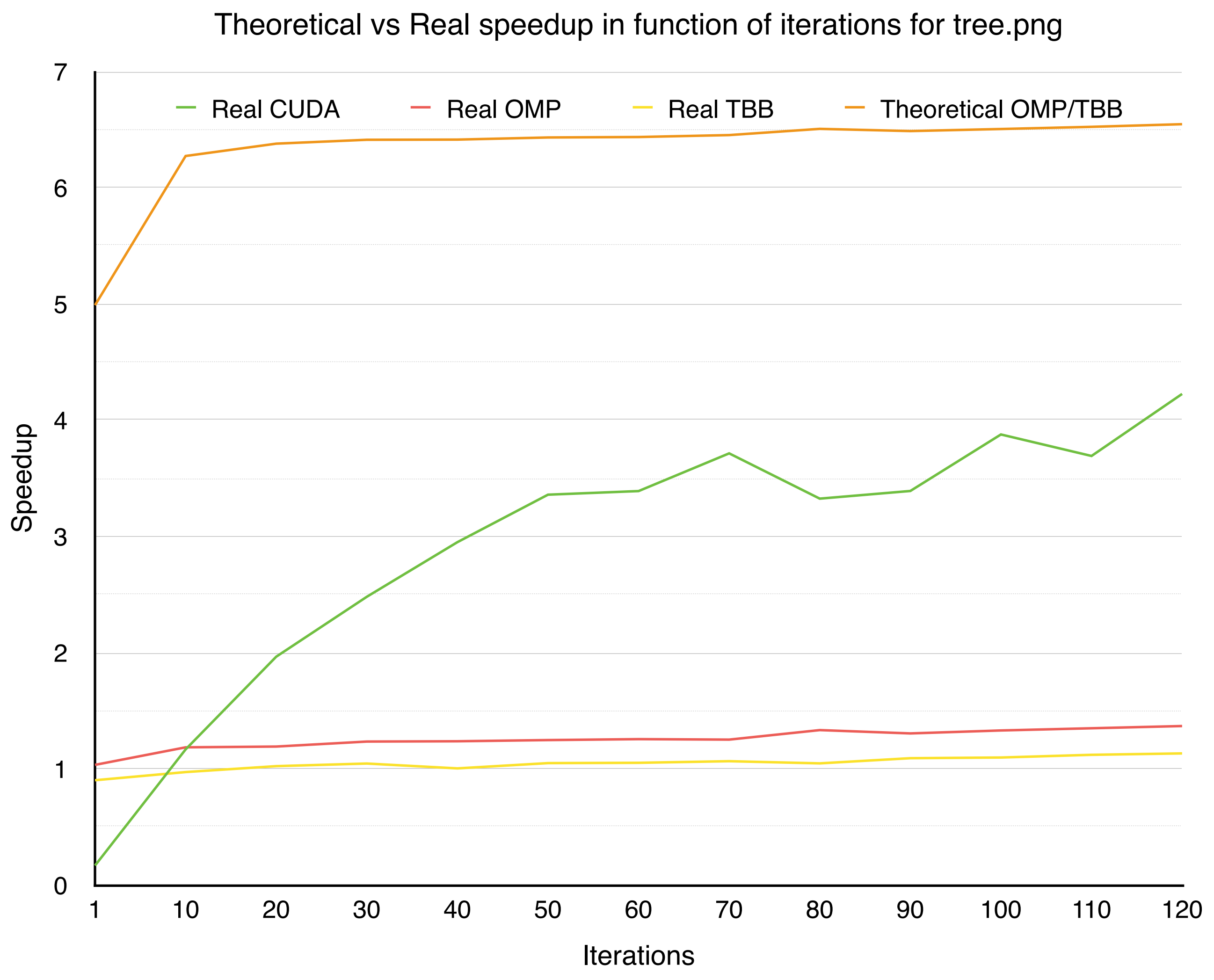




The Reality

- ➡ The speedups are **much smaller**
- ➡ CUDA still offer significant speedup, though
- ➡ CUDA can be slower when *M* is too small
  - ‣ Can't always compensate for transfer times and slower GPU clock





The Reality: Why?

The model doesn't take into account several important points

- Resource availability
  - **Memory Banks**: queuing when memory bus is busy
  - **Cache Misses**: potentially huge delays
  - **Limited ALUs**: operations delayed when no available ALU
  - 32-bits VS 64-bits operations: up to 2/3 performance lost
- Thread divergence
- Data Structure ⟺ **Stridden Memory**
  - If not carefully designed, accessing data kill bandwidth
  - E.g. stride-two access imply a 50% bandwidth penalty

Such model would be too difficult to use…

What’s not there?

- Speedup VS Iterations
  - Other images show similar behaviour than *tree.png*
- Speedup VS Image Size
  - What happens when *N* varies but not *M* ?
- Memory Bandwidth
  - Speed varies with transferred size

Questions?