Concurrent Computation of the Max-tree on GPU

Nicolas Blin

EPITA's Research and Development Laboratory
Supervisor : Edwin Carlinet

Seminar – 15 july, 2021

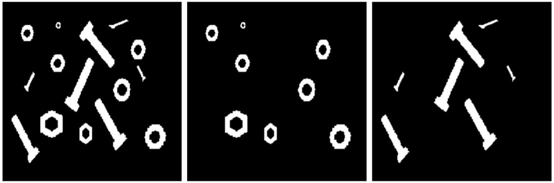
Layout

1. Contextualization : Max-tree
   1. What is it about ?
   2. Need for speed
2. Reminder about GPUs
3. State-of-the-art algorithms and Max-tree representation
4. Max-tree on GPU
5. Benchmarks and performance analysis
6. Border max-tree
7. Benchmarks and performance analysis
8. Opportunities

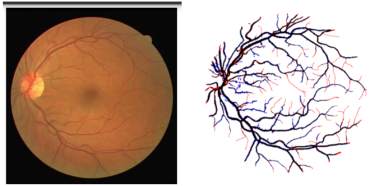
Contextualization : Max-tree

A multi-function tree

Attributes filtering[1]



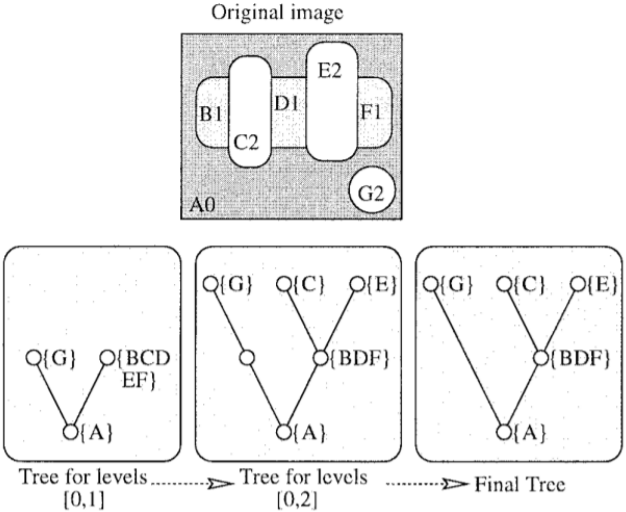
Veins segmentation[2]



- Tree-of-Shapes construction[3]

[1]Michael H. F. Wilkinson: One-Hundred-and-One Uses of a Max-Tree, in: 2004.
[2]Christophe Collet Benjamin Perret: Connected image processing with multivariate attributes: an unsupervised Markovian classification approach, in: 2014.
[3]Sébastien Crozet Edwin Carlinet Thierry Géraud: The Tree of Shapes Turned into a Max-Tree: A Simple and Efficient Linear Algorithm, in: 2018.

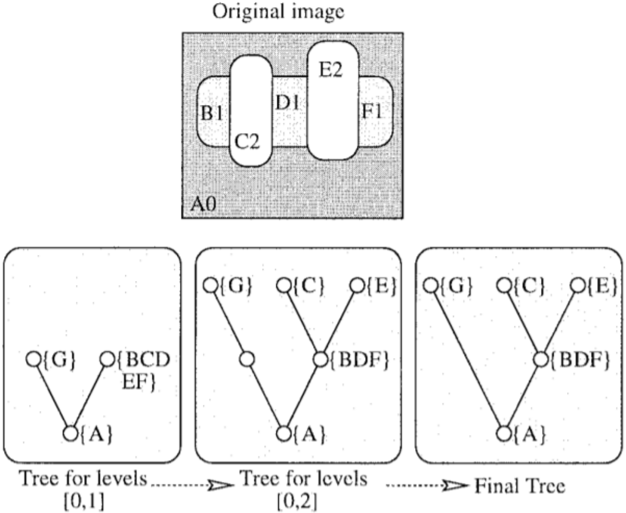
- Hierarchical representation of connected-components[4]



[4] Luis Garrido Philippe Salembier Albert Oliveras: Antiextensive connected operators for image and sequence processing, in: 1998.

Max ? Tree ?

- Hierarchical representation of connected-components[4]



[4] Philippe Salembier: Antiextensive connected operators for image and sequence processing (see n. 4).

Need for speed

- Issue : slow construction, no real-time

- New modern distributed algorithms[5][6]
- Issue : designed to handle huge images

- Solution : create a fast GPU max-tree algorithm

[5] Michael H.F. Wilkinson Jan J. Kazemier Georgios K. Ouzounis: Connected Morphological Attribute Filters on Distributed Memory Parallel Machines, in: 2017.

[6] Geraud Gotz Cavallaro: Parallel Computation of Component Trees on Distributed Memory Machines, in: 2018.

Reminders about GPUs

Reminders about GPUs

- A GPU execute thread blocks

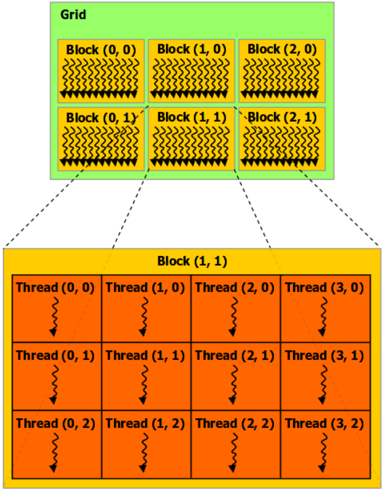


Figure: Thread blocks inside a grid

Reminders about GPUs

- The blocks are assigned and handled by SMs

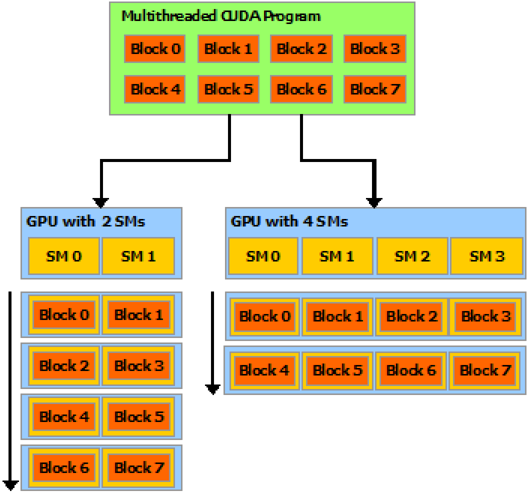


Figure: Automatic scalability with higher SM count

- Coalesced accesses (contiguous) : less memory transactions

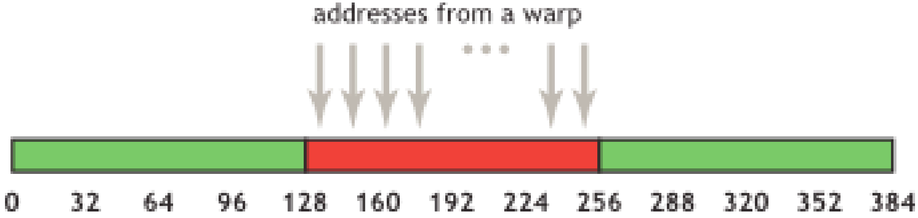


Figure: Chunk access with a single load

Reminders about GPUs

- Memory accesses do not have the same cost

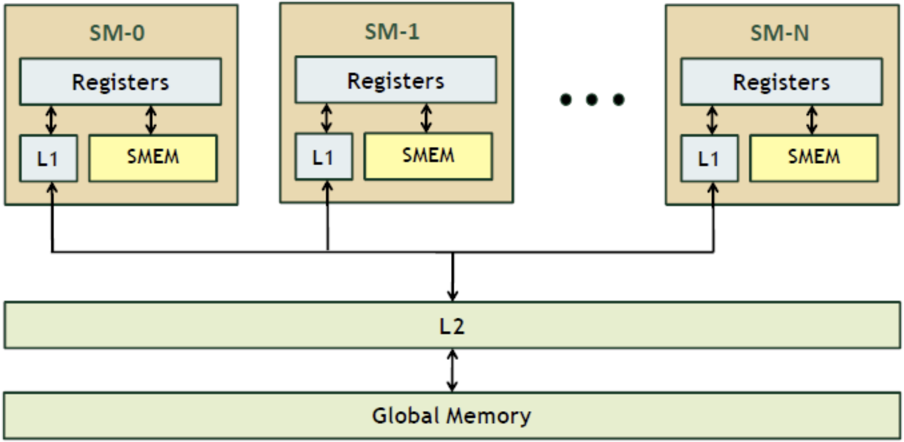


Figure: Memory hierarchy in GPUs

Reminders about GPUs

- Consequences on the desired algorithm :
  - Have a problem that is parallel enough to use all SMs
  - Maximum coalesced accesses
  - Increase data locality to maximize cache utilization

State-of-the-art algorithms and Max-tree representation

State-of-the-art : Max-tree algorithms

- Three main algorithms[7] : immersion, flooding and merge-based algorithms

- Flooding : depth-first search to build the tree
- Immersion : Sort pixels then use union-find
- Merge-base : Build trees separately then merge using union-find

- One less known : 1D (line image, sounds)[8]

[7] Thierry Géraud Edwin Carlinet: A Comparative Review of Component Tree Computation Algorithms, in: 2014.

[8] Arnaldo Araujo David Menotti Laurent Najman: 1D Component tree in linear time and space and its application to gray-level image multithresholding, in: 2007.

State-of-the-art : Max-tree representation

- 2 2D matrices
- One matrix represents the gray level image stored on one byte
- The other matrix represents parent relationship between pixels

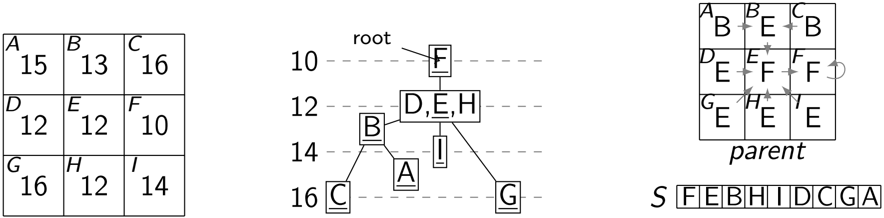


Figure: An image, corresponding max-tree and parent image

Max-tree on GPU

Max-tree on GPU : Global Pipeline

- Image splitted into tiles (1)
- Each thread builds a local max-tree for each column using the 1D algorithm (2)
- Local max-trees are merged concurrently leading to one max-tree per tile (3)
- Tiles are merge concurrently horizontally and finally vertically (4, 5, 6)

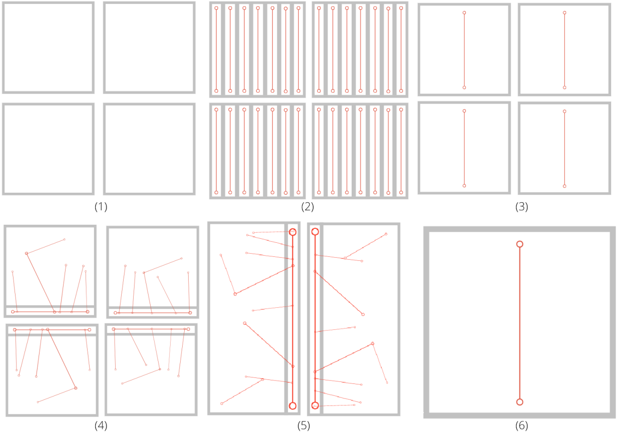


Figure: Steps of our max-tree GPU algorithm

Benchmarks and performance analysis

Benchmarks

i7 12 threads | RTX 2060 | 16 GB RAM | 6000X4000 image

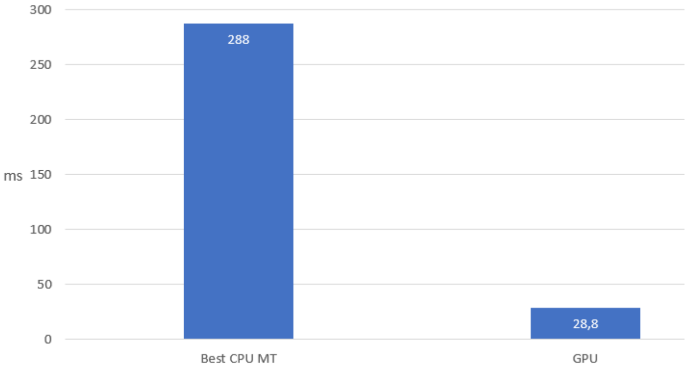


Figure: Benchmark result CPU vs GPU

- 10x speed-up
- 793MB/s : 382 fps (1920x1080) → real-time

Performance analysis : bottleneck

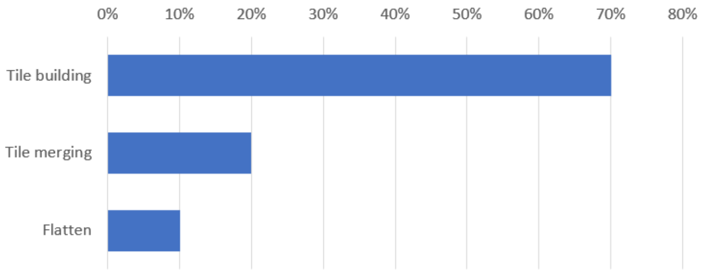


Figure: Percentage of time spent on each step

- Tile building hardly optimizable :
  - Fast 1D algorithm
  - Concurrent merges with 1 step
  - High L1 cache utilization
- Flatten : straightforward
- Tile merging : optimizable
  - Atomic non-coalesced global operations

Performance analysis : goal

- Speed-up horizontal merge
- Improve data locality and reduce data size to increase cache utilization
- Goal : Allow the SM to place the parent image in L1 cache

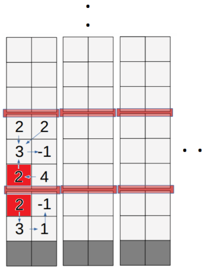


Figure: Two pixels being merged, far in memory

- Have tile nodes of each column next to each other
- Perform merges on a more compact border max-tree[9]

[9] Jan J. Kazemier: Connected Morphological Attribute Filters on Distributed Memory Parallel Machines (see n. 5).

Border max-tree

Border max-tree : useless nodes

- During union-find only the 2 merged nodes and parents are concerned

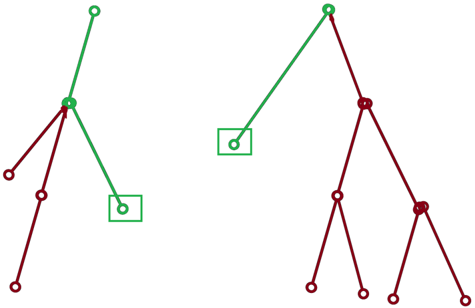


Figure: Useful nodes during merge

Border max-tree : new pipeline

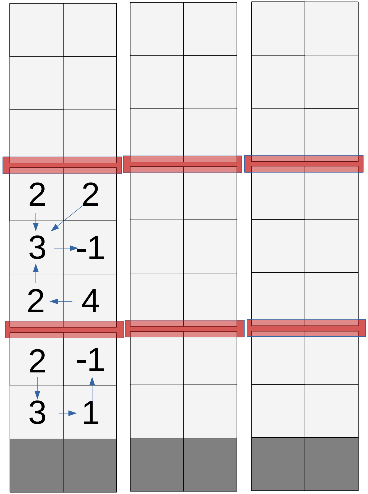
1. Build max-trees for each tile
2. Extraction border max-trees (new step)
3. Horizontal merge on border max-trees
4. Commit result in original parent image (new step)
5. Vertical merge
6. Flatten

Border max-tree : parallel algorithm

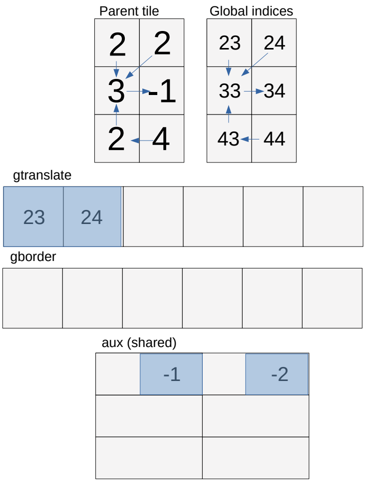
- Each thread starts on the border
- Threads concurrently climb their branch and add nodes inside the border
- To avoid concurrent accesses, nodes are marked

- Nodes are marked using negative values
- Values are shifted by 1 to handle 0 (-0 = +0)
- A translate table is used for the latter commit

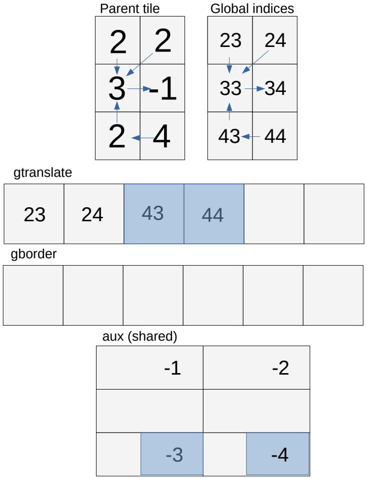
Border max-tree : Example of border extraction



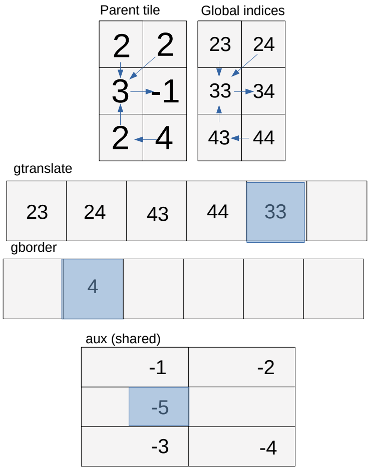
Border max-tree : Example of border extraction



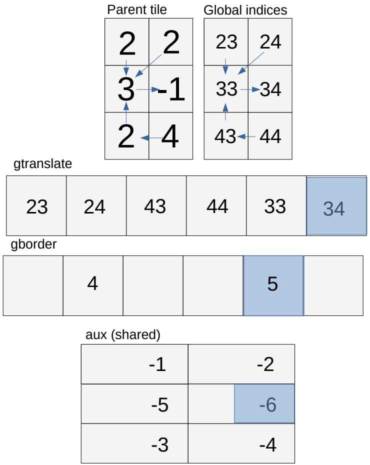
Border max-tree : Example of border extraction



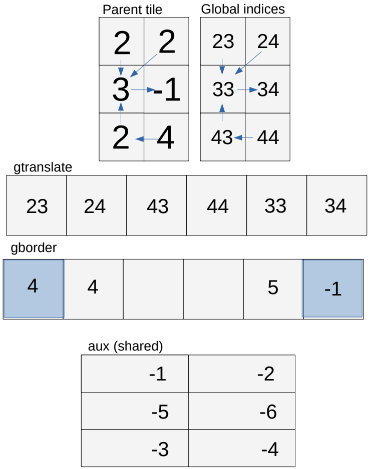
Border max-tree : Example of border extraction



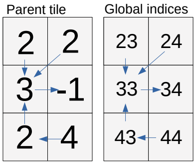
Border max-tree : Example of border extraction



Border max-tree : Example of border extraction



Border max-tree : Example of border extraction



gtranslate

23	24	43	44	33	34

gborder

4	4	4	2	5	-1

aux (shared)

-1	-2
-5	-6
-3	-4

- Each thread block writes its border in global memory
- To synchronize and know where to write, a global variable is shared among each column

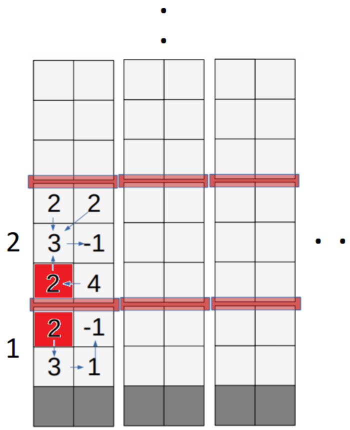
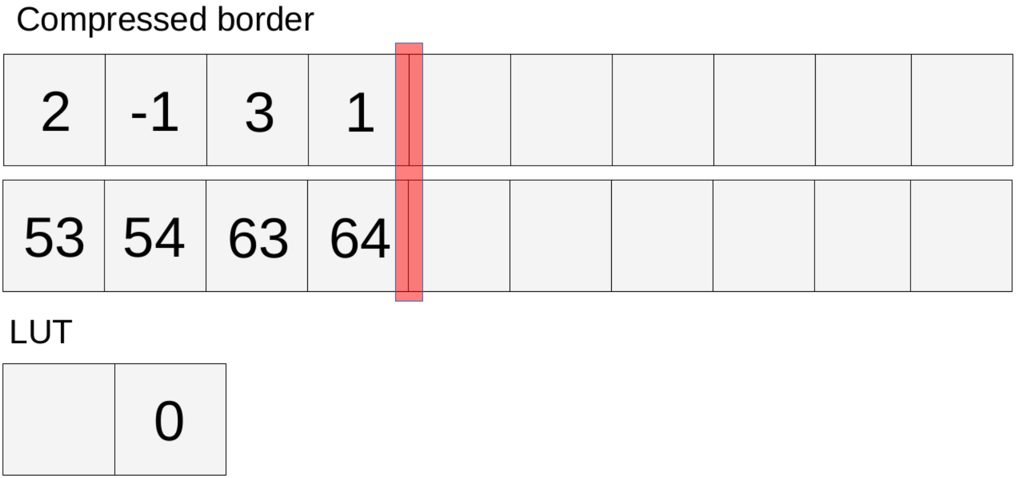


Figure: Thread block scheduling in our example


Compressed border
2
-1
3
1
53
54
63
64
LUT
0

Border max-tree : Example of border write in global

Compressed border

2	-1	3	1	8	8	8	6	9	-1
53	54	63	64	23	24	43	44	33	34

LUT

4	0

Border max-tree : Example of border write in global

Compressed border

2	-1	3	1	8	8	8	6	9	-1
53	54	63	64	23	24	43	44	33	34

LUT

4	0

Benchmarks and performance analysis

Benchmarks : extraction time

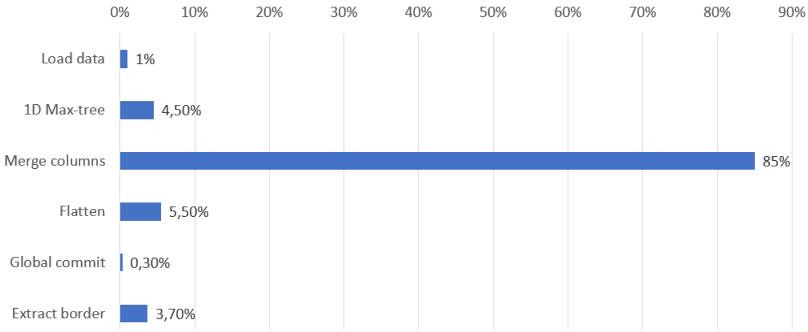


Figure: Percentage of time spent each step during tile building

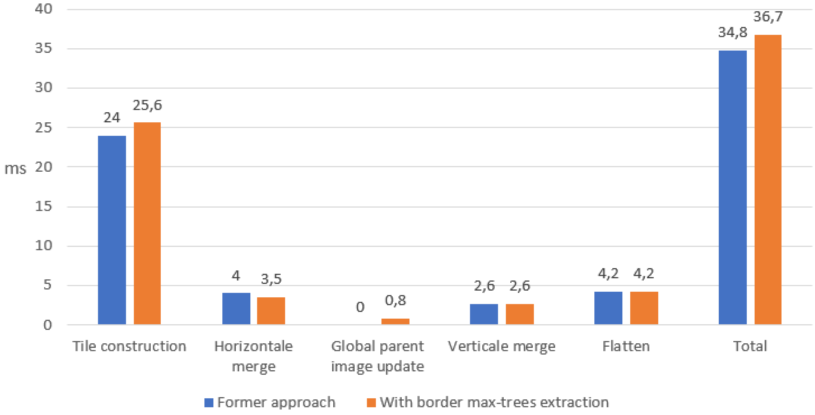
- Extraction time is low
- Mean space saving : 20% of total size

Performance analysis : border handling

- Shared memory is being reused
- Maximum possible parallelism
- Marking with position prevents multiple climbing but enables quick access
- Writing tile information inside column without sync

Benchmarks : full comparison

i7 12 threads | RTX 2060 | 16 GB RAM | 6000X4000 image



- Low speed-up during horizontal merge (0.5 ms), not enough to compensate time of the 2 extra steps (2.4 ms)
- Looks like data is still too big to fit in cache

Opportunities

Opportunities

- Border extraction is fast
- Size reduction is significant : /5

- For small enough images / big border compression
  $\rightarrow$ Data can be stored in L1 cache manually
- Could be used for a distributed GPU version

- **Fast GPU Tree-of-Shapes algorithm**

Questions ?

Bibliography

Benjamin Perret, Christophe Collet: Connected image processing with multivariate attributes: an unsupervised Markovian classification approach, in: 2014.

David Menotti Laurent Najman, Arnaldo Araujo: 1D Component tree in linear time and space and its application to gray-level image multithresholding, in: 2007.

Edwin Carlinet Thierry Géraud, Sébastien Crozet: The Tree of Shapes Turned into a Max-Tree: A Simple and Efficient Linear Algorithm, in: 2018.

Edwin Carlinet, Thierry Géraud: A Comparative Review of Component Tree Computation Algorithms, in: 2014.

Gotz Cavallaro, Geraud: Parallel Computation of Component Trees on Distributed Memory Machines, in: 2018.

Jan J. Kazemier Georgios K. Ouzounis, Michael H.F. Wilkinson: Connected Morphological Attribute Filters on Distributed Memory Parallel Machines, in: 2017.

Philippe Salembier Albert Oliveras, Luis Garrido: Antiextensive connected operators for image and sequence processing, in: 1998.

Wilkinson, Michael H. F.: One-Hundred-and-One Uses of a Max-Tree, in: 2004.

Appendix : Benchmarks, 4 techniques

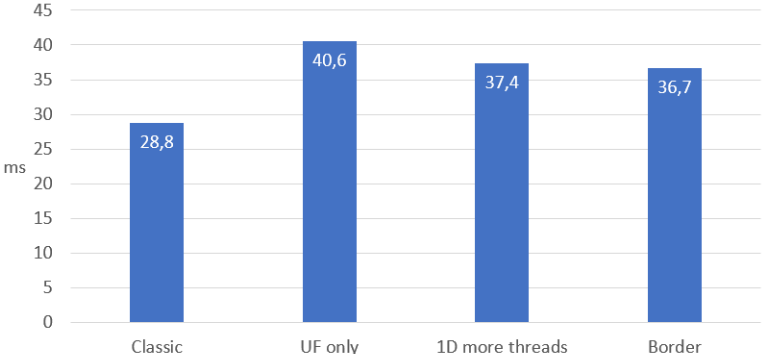


Figure: Time comparison of the 4 techniques

Appendix : Benchmarks, memcpy

- With stream, time spent on memcpy can be reduced (overlap)

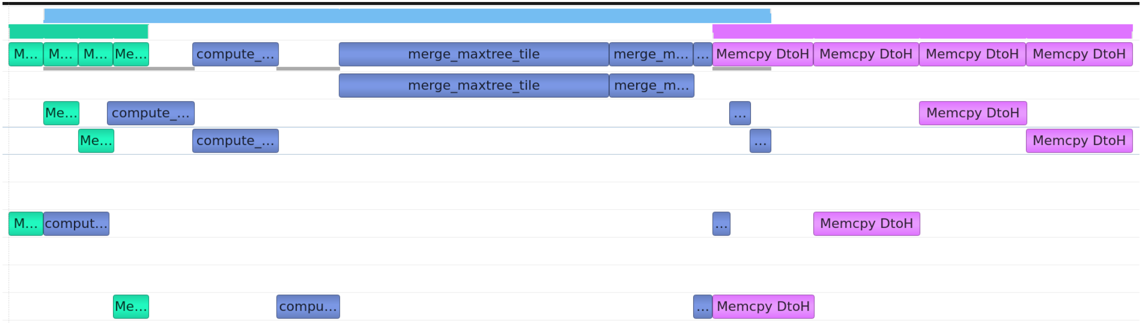


Figure: Time spent on memcpy

- 50% of time spent on memcpy
- Copying parent image is longer (bigger)
- Taking into account memcpy : 57,6 ms, 396.5 MB/S, 191 fps

Appendix : data layout

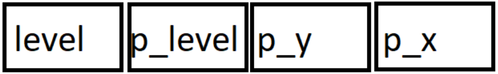


Figure: Data layout before extraction

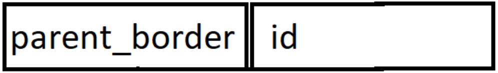
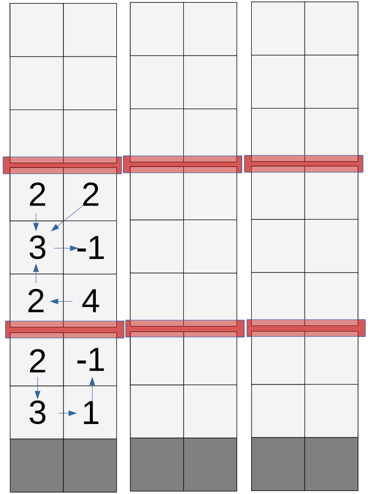
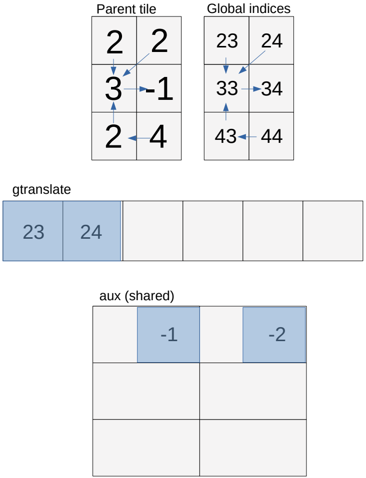


Figure: Data layout during extraction

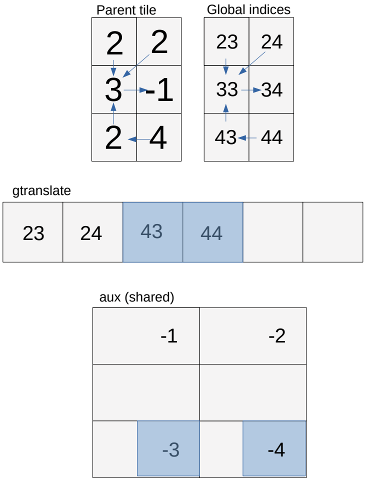
Appendix : True border extraction



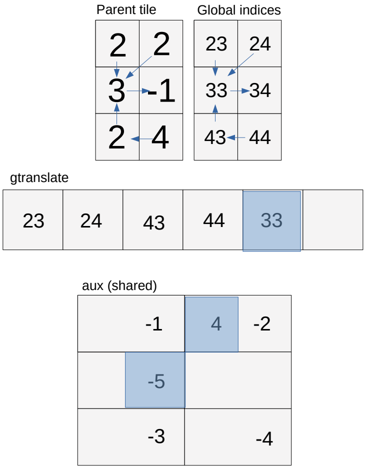
Appendix : True border extraction



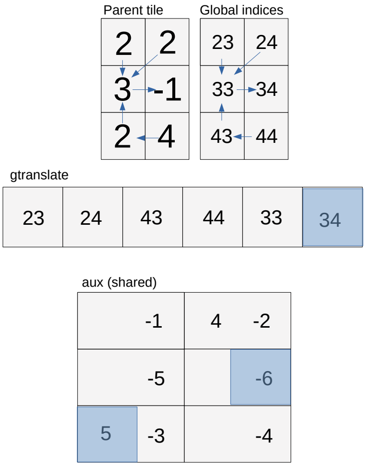
Appendix : True border extraction

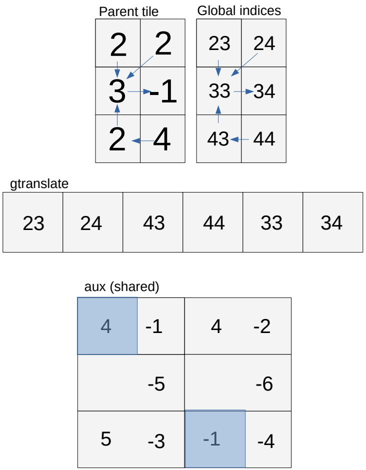


Appendix : True border extraction

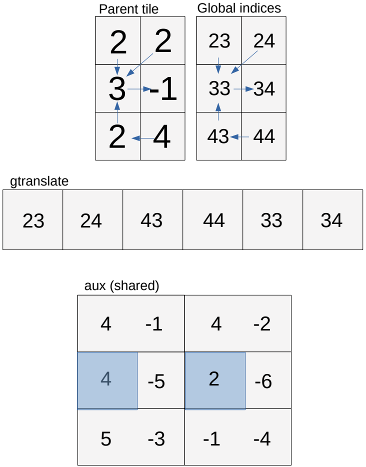


Appendix : True border extraction

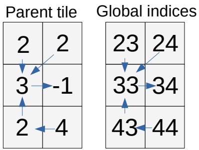



Parent tile
2 2
3 -1
2 4
Global indices
23 24
33 34
43 44
gtranslate
23 24 43 44 33 34
aux (shared)
4 -1 4 -2
-5 -6
5 -3 -1 -4

Appendix : True border extraction



Appendix : True border extraction



aux (shared)

4 -1	4 -2
4 -5	2 -6
5 -3	-1 -4

gtranslate

23	24	43	44	33	34

border

4	4	4	2	5	-1