OTB Users Days 2017

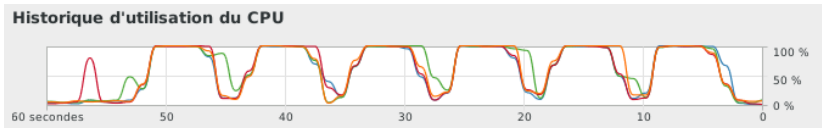
Plans for asynchronous s/writing/IO/

OTB Team

05/06/2017

What asynchronous means

- This is the CPU graph of running the OrthoRectification app :



- Question : what does OTB do when CPUs are not fully loaded ?
- Answer : IOs !
- I do not like idle CPUs . . . do you ?
- Note that for the moment everything in this talk is pure slideware

Example 1 : Ortho-rectification

	Reading time	Processing time	Writing time
Stream 1	1.4808061 s	5.2900510 s	0.8211150 s
Stream 2	2.0999321 s	5.3412859 s	0.3260920 s
Stream 3	1.8735771 s	5.3453019 s	0.8125529 s
Stream 4	1.4494860 s	5.3393729 s	0.7880170 s
Stream 5	1.4502799 s	5.3677251 s	0.8284358 s
Stream 6	1.4794530 s	5.3115728 s	0.8141160 s
Total	9.8335342 s	31.995310 s	4.3903287 s
Percentage	21.275877 %	69.225189 %	9.4989339 %

- I/O time $<$ processing time : expected gain = 30%

Example 2 : Pansharpening

	Reading time	Processing time	Writing time
Stream 1	0.0996530 s	10.387469 s	0.3946828 s
Stream 2	0.1185460 s	10.207732 s	0.2917110 s
Stream 3	0.1487219 s	10.051636 s	0.7517421 s
Stream 4	0.1215879 s	10.134621 s	0.2930951 s
Stream 5	0.1364500 s	10.115682 s	0.5939779 s
Stream 6	0.3663129 s	10.087998 s	0.2909901 s
Total	0.9912717 s	60.985138 s	2.616199 s
Percentage	1.5346519 %	94.415041 %	4.050307 %

- I/O time $\ll$ processing time : expected gain = 5%

Example 3 : Convert

	Reading time	Processing time	Writing time
Stream 1	0.0996530 s	0.7366901 s	0.3946828 s
Stream 2	0.1185460 s	0.7274349 s	0.2917110 s
Stream 3	0.1487219 s	0.7287521 s	0.7517421 s
Stream 4	0.1215879 s	0.7263241 s	0.2930951 s
Total	0.4885088 s	2.9192012 s	1.731231 s
Percentage	9.5060208 %	56.805501 %	33.688478 %

- I/O time $\simeq$ processing time : expected gain = 40 %

Conclusion

- I/O time vs. processing time heavily depends on the processing chain
- Expected gain in total time is between 5% and 40% (best case scenario)
- Reading time is at least as important as writing time. We need to do both !

A brief reminder of the pipeline

Process Object
ImageFileReader
Data Object
Image
Process Object
MedianImageFilter
Data Object
Image
Process Object
ImageFileWriter

Update()
Write()
UpdateOutputInformation()
UpdateOutputInformation()
UpdateOutputInformation()
UpdateOutputInformation()
SetSpacing()
SetOrigin()
SetSize()
SetDirection()
SetLargestPossibleRegion()
SetRequestedRegion()
PropagateRequestedRegion()
PropagateRequestedRegion()
EnlargeOutputRequestedRegion()
GenerateoutputRequestedRegion()
GenerateInputRequestedRegion()
PropagateRequestedRegion()
PropagateRequestedRegion()
EnlargeOutputRequestedRegion()
GenerateoutputRequestedRegion()
GenerateInputRequestedRegion()
UpdateOutputData()
UpdateOutputData()
UpdateOutputData()
UpdateOutputData()
GenerateData()
ImageIO->Read()
SetPixels()
GenerateData()
SetPixels()
GenerateData()

The easy part : writing

- After call to `UpdateOutputData()` for current stream, the writer passes the output buffer to the driver for writing to disk
- Making this call asynchronous with a FIFO and a separate writing thread is easy enough
- FIFO should be size-limited with a sync lock so that it does not get flowed in case reading/processing is faster than writing

The tough part : reading

Why it is more difficult

- Aim : Read next stream while processing current stream -> prefetch
- How can the reader know about requested regions for next streams ?
- Constraint : communication between reader and writer -> pipeline

Proposed solution

- Writer will make several `GenerateInputRequestedRegion()` calls for one `UpdateOutputData()` call
- On the other side of the pipeline, reader will record those regions in a FIFO (limited size)
- FIFO is processed by a separate reading thread
- Upon `UpdateOutputData()`, reader will wait for the last requested tile to be read (if not already available)

C++11 to the rescue ?

Several features from C++11 might help to implement this behaviour (in `<future>`) :

- `std::async`
- `std::packaged_task`
- `std::promise`

The tricky part is that our objects are not stateless

Conclusion and thoughts

- Gain will also be highly dependent on the hardware :
  - Single or different hard drive for input and output
  - RAID, GPFS . . .
  - Can a standard hard-drive read and write simultaneously ?
- System already does a lot of caching, should we really add another layer ?
- Trade memory for speed
- Is this worh the price (complexity of reader, writer, side effects . . .)
- Reader / Writer are quite complex already -> careful not to break anything