Parallel DEFLATE Decompression, using a GPU

ANU - COMP4560
By Matthew Whittaker
Supervised by Eric McCreath

May 26, 2015
Goals

- Look into existing algorithms
- Build base implementation
- Build a parallel GPU implementation
- Compare parallel implementation to my reference one, and others
- Assess viability
DEFLATE consists of a series of the following chunks. Very sequential – no random access.
Background

GPGPU on OpenCL – highly parallel
- Use a limited “c”
- No dynamic memory management
- Designed for doing lots of independent simple transformations in parallel
- No message passing or anything
Approaches

- Start at every bit?
  - More complicated
  - Less potential gain
  - Works better for small chunks

- Decode at evenly spaced intervals
  - Most inputs will synchronise eventually
  - If they don't synchronise, still works, just slow
  - Not as easy as it sounds
My Approach

Synchronisation points. How to detect? Once detected, where in the output buffer is next symbol? Requires complex/expensive structures.

Chunk end. Don't know where it is but we can make good guesses!

Worker start positions (most don't align with symbol boundary)
What I Achieved

- Fully functional CPU single threaded decoder – as a reference and to learn the RFC
- Limited GPU single worker decoder – as a stepping stone
- Designed a complete algorithm. Did not have time to implement it. Solves the problems but does so inefficiently. Uses approach I described, with large footprint structures to track everything. See my report for specifics.
- Other auxiliary applications DEFLATE extraction from GZIP... basic hex dump, that sort of stuff
- Investigated viability based on disk accesss, GPU access, and reference implementation speed and general practicality.
Why I did not complete parallel implementation

- Format low level and complex – doing it in parallel it is easy to introduce subtle bugs.
- No dynamic memory management – write our own? What's the cost of this? Or just use huge buffers?
- Proposed algorithm turns out to be very intricate, and more complicated than you'd think. How to detect synchronisation? How to prevent overwriting buffers we still need? If we detect synchronisation, how do we know where the rest of the output is?
- I solved these problems, using LOTs of memory. Ran out of time to finish implementation though :(
Is it viable?

- Speed
- Practicality
Speed assessment

- Golang – 6s/GB, or 22s/GB with disk cache cleared
- GZIP – 8s/GB or 20s/GB with disk cache cleared
- My CPU implementation – 80s/GB

- Already this shows small potential savings. About 30%. What about GPU access latency and throughput?
- We can only really do one chunk per kernel – it's already very hard to manage all your workers
  - Golang's deflate uses 16KB chunks
  - GZIP's uses 32KB
- Kernel latency is about 10us$^3$ (about 0.3s/GB) and throughput about 0.2s/GB this makes 0.5s/GB. High speed up would be plausible if we aren't reading data from disk.
- Even if computation is instant, speed up is small!
Not really practical

- GPGPU does not lend itself well to huffman decoding. Does not handle communication between workers or memory management well. Not *nearly* as easy to do Huffman decoding as other popular things like Bitcoin mining.
- Introducing OpenCL as a dependency to your application? Low portability (compared to CPU code, but good as far as GPGPU solutions go), relies on proprietary software: less trustworthy and large.
- Introduces huge complexity. Harder to maintain, more bugs.
- If possible, switching format to one designed for parallel decoding will be easier.
- A multi-threaded CPU approach is more suited, but there's still not that much point.
What next?

• If you're interested, decide if it's actually worthwhile for what you want to do. It will take a large amount of resources for a probably small gain. Niche use case.

• If so, implement my algorithm (described in detail in my report). Try to improve on memory usage.

• Will require a lot of time, and a lot of OpenCL (or other GPGPU platform) knowledge to do it efficiently.
Questions

• Thanks for listening!

• Resources:
  – 4: Boilerplate OpenCL code I adapted, plus a quick tutorial - http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854