Engineering adaptive data compression using arithmetic coding and prediction by partial matching (PPM)

Janne Kataja

Project in Data Compression Techniques

2012-04-26

Topics

Topics covered in this presentation:

- Implementation details
- Improvements
- Experiments

Implementation: Underlying data structures 1/2

Prediction Partial Matching model needs to maintain a data structure which stores frequencies of the symbols which have occured in contexts.

- Implemented using cuckoo hashing.
- Cuckoo hashing uses two hash functions which give the two possible locations for a key.
- Collisions in the table are resolved by displacing the occupying key to it's alternate location.
  - Resolving is repeated until a vacant position is found.
  - After max. attempts, the hash table needs to be rebuilt.
- Lookup time is constant.
- Insertions succeed in expected constant time.

Implementation: Underlying data structures 2/2

- Key is length+context+symbol packed in 64 bit long.
- Value is the frequency of the text and a bit vector of symbols occuring after.
- Bit vector of followers improved performance (fragmentation).
- Allocates fixed amount of memory for the hash table, set from the command line.
- When memory is exhausted, instead of rebuilding the contents are thrown away.
- Uses `crc32` hardware instruction as hashing function.

Lesson learned: array based trie. use cuckoo hash with more than 2 hashes for resolving collisions instead of maintaining expensive data structures.

Implementation: Frequency of escape

assigns count of symbols in context.

Implementation: Rescale

- Arithmetic coding using 64 bit variables allows for more freedom in choosing rescale threshold.
- Frequencies of all symbols in all contexts are halved.
- Rescale reduces weight of past data:
  - Improved local adaptation; distribution is skewed towards local data.
  - Worse result when underlying distributions in input do not change.
- Allowing frequency to be right-shifted to zero gives slightly better compression with `enwik8`: 1.851 bpc vs 1.863 bpc.

Improvement: Bootstrap

Asettamalla BOOTSTRAP -define ohjelma käyttää viimeistä kahta kilotavua rengaspuskurina ja lisää sen sisällön tyhjennyksen yhteydessä.

- Data structure is cleared when pre-allocated memory is exhaused.
- Use recent buffer (dafault 32 KiB) to bootstrap model.

Improvement: Update exclusion

Update exclusion

`pompom -o6 -m2048 < enwik8`

- without 1.972 bpc
  - Symbol with had frequency in a higher order context can appear later.
- lazy 1.899 bpc
  - exclusion runs only 1 order deep
- Full exlucion: 1.869 bpc
  - Exclusion keeps a bit vector from highest to the lowest order. Vector contains the symbols which have frequency assigned.

Improvement: Rescaling

implemented an option to rescale whenever threshold value of escaped frequency is met:

- `pompom -o5 -m256 < calgary.tar`
  - with -a option: 2.052 bpc
  - without -a option: 2.066 bpc
- Improves local adaptability.
- Distribution is skewed faster towards local context.
- Minor improvement when input is e.g. an archive containing unrelated files.
- Similar worsened bpc when input data is uniform.

Introduction

spread out in memory crc32 instruction
The instruction is added in SSE4.2 (available in i5/i7 or later). Otherwise, software hashing will be used. Using CRC32 instruction has a major impact on performance.

Usage

```
$ bin/pompom -h

Usage: pompom [OPTION]...
Compress or decompress input using fixed-order PPM compression.
Reads from standard input and writes to standard output.

Options:
  -c [ --stdout ]                compress to stdout (default)
  -d [ --decompress ]            decompress to stdout
  -h [ --help ]                  show this help
  -a [ --adapt ]                 compress: fast local adaptation
  -A [ --adaptsize ] arg (=22)   compress: adaptation threshold in bits [
  -r [ --reset ]                 compress: full reset model on memory lim
  -b [ --bootsize ] arg (=32)    compress: bootstrap buffer size in KiB [
  -n [ --count ] arg (=0)        compress: stop after count bytes
  -o [ --order ] arg (=3)        compress: model order [1,6]
  -m [ --mem ] arg (=32)         compress: memory use in MiB [8,2048]
```

Recap:

calgary comparative

Introduction

enwik8 comparative

Introduction

order/mem use graph

Introduction

order/length graph 100,10k,100k,1M,10M,100M

Model options

order
-o sets order and -m sets memory use in MiB
enwik8 calgary -o3 -m8 -o4 -m64 -o5 -m512 -o6 -m2048
options are stored at the beginning of the the compressed data

Questions

Full source code is hosted on github:
`https://github.com/jkataja/pompom`