# Ceph Code Optimization

-----

## Compile-time Optimizations

Most of ceph is built with “-O2”.  Selected parts either contain -m<extension> or contain assembler code with instruction extensions.  All of this is enabled at compile time based strictly on what options the build compiler and assembler support.  The bulk of this is limited to the erasure code logic.  There is limited support for crc32 acceleration and sha256 acceleration in several other parts.

**Special notes: zstd (14 files) is built with -O0.  Some parts of the intel isa that should support avx512 are built as empty files, because we don’t supply the right option to enable avx512 support.

### 1. General

   a. radosgw - Like most of ceph, built almost entirely w/ -O2.
   b. ceph-osd -  Rocksdb, used by bluestore, has its own build system, and looks for crc32. The erasure code is built as separate plugins, and the “ec-isa” plugin can take advantage of Intel storage acceleration.

### 2. Modules of Special Interest

   a. *openssl* - We don’t build this, so no compile flags here.
   b. *jerasure* - Most of the erasure code logic can take advantage of up to sse4.1 logic.  The erasure code plug-in “libec_isa.so” can use up to AVX2 instructions.
   c. *rocksdb* - This code uses crc32, and includes support to look for and use SSE4.2 for this.
   d. *compressor code (zstd)* - Used by s3 interface to compress objects in buckets.  Bad optimization, but contains possible use of crc32 acceleration.

## Object Code As Packaged (Ceph and platform)

### 3. Objdump Inspection of Selected Objects

   a. radosgw - The bulk of this is built with just -O2.  It links against libceph_common, which looks for and can use crc32 instructions.
   b. ceph-osd - The bluestore code can use BMI instructions, part of haswell.
   c. openssl - This is supplied by Ubuntu, and loaded at runtime by radosgw.  On Ubuntu, it is also used by libcurl.  It should be able to take advantage of most CPU instruction optimizations, including use of sha1 and avx2.
   d. jerasure -  Most parts can use up to sse4.1.  Selected parts (esp. libec_isa.so) can use additional intel features, up to avx2.

### 4. Implementation of Key Algorithms

   a. CRC (Messenger, bluestore? checksums) - Used by bluestore and messenger.  The bluestore call can definitely use extensions.  The messenger code should be able to as well.
   b. MD5 (radosgw ETAG)  - No special optimizations.
   c. SHA-256 (radosgw AWS4_HMAC_SHA256) -  There are various optimizations versions available up to AVX2.  Ceph code should be able to take advantage of this.

d. AES (civetweb/beast/libcurl SSL/TLS) - For incoming connections, civetweb & beast will both use openssl, which contains aes acceleration. For outgoing connections, libcurl on ubuntu will also use openssl, so ditto. There is some internal support for aes acceleration in ceph itself, such as for cephx.

## Test environments:

a. QE uses “magna” machines which are ivy bridge.
b. a^2 has mostly haswell machines for development.
c. Scalelab machine has skylake machines, for very temporary testing at scale.

---

## Follow-up Question:

It's understood the that various parts of Ceph *can* use SSE 4.1, 4.2 levels. What's being sought here is verification of specifically what SSE* levels the packages we receive *actually do* or *does* support. Please provide instructions on how we can verify the SSE levels in our installation and/or compile time flags that demonstrate SSE levels.

In particular, in RH's deb build environment, in this file:

~/ceph/src/erasure-code/jerasure/Makefile*

What do the INTEL_SSE*_FLAGS evaluate to?

## Response:

In luminous, upstream
commit
6faace0890 build/ops: remove autoconf leftover
removes
src/erasure-code/jerasure/Makefile*

Accordingly, cmake is used exclusively to build ceph (luminous) and newer.

There are no references to
INTEL_SSE*_FLAGS
either in the source or during the build process.

files under
src/erasure-code/jerasure/gf-complete
are built with,

```
-DINTEL_SSE -DINTEL_SSE2 -DINTEL_SSE3 -DINTEL_SSE4 -DINTEL_SSE4_PCLMUL
-DINTEL_SSSE3 -D_FILE_OFFSET_BITS=64 -D_FORTIFY_SOURCE=2 -D_GNU_SOURCE
-D_REENTRANT -D_THREAD_SAFE -D__CEPH__ -D__STDC_FORMAT_MACROS -D__linux__
-O2 -mpclmul -msse -msse2 -msse3 -msse4.1 -msse4.2 -mssse3 -std=gnu99
```

The intel isa-l code is not conditioned by any such flags.
It is conditioned with HAVE_AS_KNOWS_AVX512, which we don't currently set.
Other than that, the list of objects (and contents) is static.

ceph_zlib is affected by if HAVE_INTEL_SSE4_1 is set in cmake.
That will always be true in xenial, so CompressionPluginZlib.cc.o will
always be built.