Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Open Source Version

API Reference Manual - Version 2.13

January 30, 2015

Contents

CHAPTER 1 STORAGE LIBRARY

1.1 About This Document

This document describes the software programming interface and operation of functions in the library. Sections in this document are grouped by the functions found in individual header files that define the function prototypes. Subsections include function parameters, description and type.

This document refers to the open release version of the library. A separate, general release called the Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) is also available and contains an extended set of functions.

1.2 Overview

The Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Open Source Version is a collection of functions used in storage applications optimized for Intel architecture Intel® 64. In some cases, multiple versions of the same function are available that are optimized for a particular Intel architecture and instruction set. This software takes advantage of new instructions and users should ensure that the chosen function is compatible with hardware it will run on.

1.3 Erasure Code Functions

Functions pertaining to erasure codes implement a general Reed-Solomon type encoding for blocks of data to protect against erasure of whole blocks. Individual operations can be described in terms of arithmetic in the Galois finite field GF(2^8) with the particular field-defining primitive or reducing polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (0x1d).

For example, the function ec_encode_data() will generate a set of parity blocks P_i from the set of k source blocks D_i and arbitrary encoding coefficients $a_{i,j}$ where each byte in P is calculated from sources as:

$$P_i = \sum_{j=1}^{k} a_{i,j} \cdot D_j$$

where addition and multiplication $\cdot$ is defined in GF(2^8). Since any arbitrary set of coefficients $a_{i,j}$ can be supplied, the same fundamental function can be used for encoding blocks or decoding from blocks in erasure.

As noted in the above document, the scheduler routines do not enforce atomic access to the context structure. If a single scheduler state structure is being used by multiple threads, then the application must take care that calls are not made from different threads at the same time, i.e. thread-safety should be implemented at a level higher than these routines. This could be implemented by employing a separate context structure for each worker thread.

1.4 System Requirements

Individual functions may have various run-time requirements such as the minimum version of SSE as described in Instruction Set Requirements. General requirements are listed below.

Recommended Hardware:

- em64t: A system based on the Intel® Xeon® processor with Intel® 64 architecture.
- IA32: When available for 32-bit functions; A system based on the Intel® Xeon® processor or subsequent IA-32 architecture based processor.

Software Requirements:

Most functions in the library use the 64-bit embedded and Unix standard for calling convention `http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.95.pdf`. When available, 32-bit versions use cdecl. Individual functions are written to be statically linked with an application.

Building Library Functions:

- Yasm Assembler: version at least v1.2.0.

Building Examples and Tests:

Examples and test source follow simple command line POSIX standards and should be portable to any mostly POSIX-compliant OS.

Note

Please note that the library assumes 1MB = 1,000,000 bytes in reported performance figures.

CHAPTER 2
FUNCTION VERSION NUMBERS

2.1 Function Version Numbers

Individual functions are given version numbers with the format mm-vv-ssss.
- mm = Two hex digits indicating the processor a function was optimized for.

```
  - 00 = Nehalem/Jasper Forest/Multibinary
  - 01 = Westmere
  - 02 = Sandybridge
  - 03 = Ivy Bridge
  - 04 = Haswell
  - 05 = Silvermont
```

- vv = function version number
- ssss = function serial number

2.2 Function Version Numbers Tables

Function	Version
gf_vect_mul_sse	00-02-0034
gf_vect_mul_init	00-02-0035
gf_vect_mul_avx	01-02-0036
gf_vect_dot_prod_sse	00-04-0060
gf_vect_dot_prod_avx	02-04-0061
gf_2vect_dot_prod_sse	00-03-0062
gf_3vect_dot_prod_sse	00-05-0063
gf_4vect_dot_prod_sse	00-05-0064
gf_5vect_dot_prod_sse	00-04-0065
gf_6vect_dot_prod_sse	00-04-0066
ec_init_tables	00-01-0068
ec_encode_data_sse	00-02-0069
ec_encode_data	00-03-0133
gf_vect_mul	00-02-0134
ec_encode_data_base	00-01-0135
gf_vect_mul_base	00-01-0136
gf_vect_dot_prod_base	00-01-0137
gf_vect_dot_prod	00-02-0138
gf_vect_dot_prod_avx2	04-04-0190
gf_2vect_dot_prod_avx	02-04-0191
gf_3vect_dot_prod_avx	02-04-0192
gf_4vect_dot_prod_avx	02-04-0193
gf_5vect_dot_prod_avx	02-03-0194
gf_6vect_dot_prod_avx	02-03-0195
gf_2vect_dot_prod_avx2	04-04-0196
gf_3vect_dot_prod_avx2	04-04-0197
gf_4vect_dot_prod_avx2	04-04-0198
gf_5vect_dot_prod_avx2	04-03-0199
gf_6vect_dot_prod_avx2	04-03-019a
gf_vect_mad_sse	00-00-0200
gf_vect_mad_avx	02-00-0201
gf_vect_mad_avx2	04-00-0202
gf_2vect_mad_sse	00-00-0203
gf_2vect_mad_avx	02-00-0204
gf_2vect_mad_avx2	04-00-0205
gf_3vect_mad_sse	00-00-0206
gf_3vect_mad_avx	02-00-0207
gf_3vect_mad_avx2	04-00-0208
gf_4vect_mad_sse	00-00-0209
gf_4vect_mad_avx	02-00-020a
gf_4vect_mad_avx2	04-00-020b

Function	Version
gf_5vect_mad_sse	00-00-020c
gf_5vect_mad_avx	02-00-020d
gf_5vect_mad_avx2	04-00-020e
gf_6vect_mad_sse	00-00-020f
gf_6vect_mad_avx	02-00-0210
gf_6vect_mad_avx2	04-00-0211
ec_encode_data_update	00-02-0212
gf_vect_mad	00-01-0213

CHAPTER 3
INSTRUCTION SET REQUIREMENTS

ec_encode_data_avx (int len, int k, int rows, unsigned char ∗gftbls, unsigned char ∗∗data, unsigned char ∗∗coding)
AVX

ec_encode_data_avx2 (int len, int k, int rows, unsigned char ∗gftbls, unsigned char ∗∗data, unsigned char ∗∗coding)
AVX2

ec_encode_data_sse (int len, int k, int rows, unsigned char ∗gftbls, unsigned char ∗∗data, unsigned char ∗∗coding)
SSE4.1

ec_encode_data_update_avx (int len, int k, int rows, int vec_i, unsigned char ∗g_tbls, unsigned char ∗data, unsigned char ∗∗coding)
AVX

ec_encode_data_update_avx2 (int len, int k, int rows, int vec_i, unsigned char ∗g_tbls, unsigned char ∗data, unsigned char ∗∗coding)
AVX2

ec_encode_data_update_sse (int len, int k, int rows, int vec_i, unsigned char ∗g_tbls, unsigned char ∗data, unsigned char ∗∗coding)
SSE4.1

gf_2vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
AVX

gf_2vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
AVX2

gf_2vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
SSE4.1

gf_2vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
AVX

gf_2vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
AVX2

gf_2vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
SSE4.1

gf_3vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
AVX

gf_3vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
AVX2

gf_3vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
SSE4.1

gf_3vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX

gf_3vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX2

gf_3vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    SSE4.1

gf_4vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX

gf_4vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX2

gf_4vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    SSE4.1

gf_4vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX

gf_4vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX2

gf_4vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    SSE4.1

gf_5vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX

gf_5vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX2

gf_5vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    SSE4.1

gf_5vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX

gf_5vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX2

gf_5vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    SSE4.1

gf_6vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX

gf_6vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    AVX2

gf_6vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)
    SSE4.1

gf_6vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX

gf_6vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    AVX2

gf_6vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)
    SSE4.1

gf_vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)
    AVX

gf_vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)
    AVX2

gf_vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)
    SSE4.1

gf_vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)
    AVX

gf_vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)
    AVX2

gf_vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)
    SSE4.1

gf_vect_mul_avx (int len, unsigned char ∗gftbl, void ∗src, void ∗dest)
    AVX

gf_vect_mul_sse (int len, unsigned char ∗gftbl, void ∗src, void ∗dest)
    SSE4.1

CHAPTER 4
FILE INDEX

4.1 File List

Here is a list of all documented files with brief descriptions:

CHAPTER 5
FILE DOCUMENTATION

5.1 erasure_code.h File Reference

Interface to functions supporting erasure code encode and decode.

```
#include "gf_vect_mul.h"
```

Functions

- void ec_init_tables (int k, int rows, unsigned char *a, unsigned char *gftbls)

    Initialize tables for fast Erasure Code encode and decode.
- void ec_encode_data (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

    Generate or decode erasure codes on blocks of data, runs appropriate version.
- void ec_encode_data_sse (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

    Generate or decode erasure codes on blocks of data.
- void ec_encode_data_avx (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

    Generate or decode erasure codes on blocks of data.
- void ec_encode_data_avx2 (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

    Generate or decode erasure codes on blocks of data.
- void ec_encode_data_base (int len, int srcs, int dests, unsigned char *v, unsigned char **src, unsigned char **dest)

    Generate or decode erasure codes on blocks of data, runs baseline version.
- void ec_encode_data_update (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

    Generate update for encode or decode of erasure codes from single source, runs appropriate version.
- void ec_encode_data_update_sse (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

    Generate update for encode or decode of erasure codes from single source.
- void ec_encode_data_update_avx (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

    Generate update for encode or decode of erasure codes from single source.
- void ec_encode_data_update_avx2 (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

Generate update for encode or decode of erasure codes from single source.

- void ec_encode_data_update_base (int len, int k, int rows, int vec_i, unsigned char ∗v, unsigned char ∗data, unsigned char ∗∗dest)

  Generate update for encode or decode of erasure codes from single source.
- void gf_vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)

  GF(2^8) vector dot product.
- void gf_vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)

  GF(2^8) vector dot product.
- void gf_vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)

  GF(2^8) vector dot product.
- void gf_2vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with two outputs.
- void gf_2vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with two outputs.
- void gf_2vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with two outputs.
- void gf_3vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with three outputs.
- void gf_3vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with three outputs.
- void gf_3vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with three outputs.
- void gf_4vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with four outputs.
- void gf_4vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with four outputs.
- void gf_4vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with four outputs.
- void gf_5vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with five outputs.

- void gf_5vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with five outputs.
- void gf_5vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with five outputs.
- void gf_6vect_dot_prod_sse (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with six outputs.
- void gf_6vect_dot_prod_avx (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with six outputs.
- void gf_6vect_dot_prod_avx2 (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗∗dest)

  GF(2^8) vector dot product with six outputs.
- void gf_vect_dot_prod_base (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)

  GF(2^8) vector dot product, runs baseline version.
- void gf_vect_dot_prod (int len, int vlen, unsigned char ∗gftbls, unsigned char ∗∗src, unsigned char ∗dest)

  GF(2^8) vector dot product, runs appropriate version.
- void gf_vect_mad (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)

  GF(2^8) vector multiply accumulate, runs appropriate version.
- void gf_vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_base (int len, int vec, int vec_i, unsigned char ∗v, unsigned char ∗src, unsigned char ∗dest)

  GF(2^8) vector multiply accumulate, baseline version.
- void gf_2vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 2 accumulate. SSE version.
- void gf_2vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 2 accumulate. AVX version of gf_2vect_mad_sse().
- void gf_2vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 2 accumulate. AVX2 version of gf_2vect_mad_sse().

- void gf_3vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 3 accumulate. SSE version.
- void gf_3vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 3 accumulate. AVX version of gf_3vect_mad_sse().
- void gf_3vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 3 accumulate. AVX2 version of gf_3vect_mad_sse().
- void gf_4vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 4 accumulate. SSE version.
- void gf_4vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 4 accumulate. AVX version of gf_4vect_mad_sse().
- void gf_4vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 4 accumulate. AVX2 version of gf_4vect_mad_sse().
- void gf_5vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 5 accumulate. SSE version.
- void gf_5vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 5 accumulate. AVX version.
- void gf_5vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 5 accumulate. AVX2 version.
- void gf_6vect_mad_sse (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 6 accumulate. SSE version.
- void gf_6vect_mad_avx (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 6 accumulate. AVX version.
- void gf_6vect_mad_avx2 (int len, int vec, int vec_i, unsigned char ∗gftbls, unsigned char ∗src, unsigned char ∗∗dest)

  GF(2^8) vector multiply with 6 accumulate. AVX2 version.
- unsigned char gf_mul (unsigned char a, unsigned char b)

  Single element GF(2^8) multiply.
- unsigned char gf_inv (unsigned char a)

  Single element GF(2^8) inverse.
- void gf_gen_rs_matrix (unsigned char ∗a, int m, int k)

  Generate a matrix of coefficients to be used for encoding.

- void gf_gen_cauchy1_matrix (unsigned char ∗a, int m, int k)

  Generate a Cauchy matrix of coefficients to be used for encoding.
- int gf_invert_matrix (unsigned char ∗in, unsigned char ∗out, const int n)

  Invert a matrix in GF(2^8)

5.1.1 Detailed Description

Interface to functions supporting erasure code encode and decode. This file defines the interface to optimized functions used in erasure codes. Encode and decode of erasures in GF(2^8) are made by calculating the dot product of the symbols (bytes in GF(2^8)) across a set of buffers and a set of coefficients. Values for the coefficients are determined by the type of erasure code. Using a general dot product means that any sequence of coefficients may be used including erasure codes based on random coefficients. Multiple versions of dot product are supplied to calculate 1-6 output vectors in one pass. Base GF multiply and divide functions can be sped up by defining GF_LARGE_TABLES at the expense of memory size.

5.1.2 Function Documentation

5.1.2.1 void ec_encode_data (int *len,* int *k,* int *rows,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *data,* unsigned char ∗∗ *coding*)

Generate or decode erasure codes on blocks of data, runs appropriate version.

Given a list of source data blocks, generate one or multiple blocks of encoded data as specified by a matrix of GF(2^8) coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each block of data (vector) of source or dest data.
k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗k∗rows
data	Array of pointers to source input buffers.
coding	Array of pointers to coded output buffers.

Returns

none

5.1.2.2 void ec_encode_data_avx (int *len*, int *k*, int *rows*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *data*, unsigned char ∗∗ *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires AVX

5.1.2.3 void ec_encode_data_avx2 (int *len*, int *k*, int *rows*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *data*, unsigned char ∗∗ *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires AVX2

5.1.2.4 void ec_encode_data_base (int *len*, int *srcs*, int *dests*, unsigned char ∗ *v*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

Generate or decode erasure codes on blocks of data, runs baseline version.

Baseline version of ec_encode_data() with same parameters.

5.1.2.5 void ec_encode_data_sse (int *len*, int *k*, int *rows*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *data*, unsigned char ∗∗ *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires SSE4.1

5.1.2.6 void ec_encode_data_update (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char ∗ *g_tbls*, unsigned char ∗ *data*, unsigned char ∗∗ *coding*)

Generate update for encode or decode of erasure codes from single source, runs appropriate version.

Given one source data block, update one or multiple blocks of encoded data as specified by a matrix of GF(2^8) coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes from one input source at a time.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each block of data (vector) of source or dest data.
k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
vec_i	The vector index corresponding to the single input source.
g_tbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗k∗rows
data	Pointer to single input source used to update output parity.
coding	Array of pointers to coded output buffers.

Returns

none

5.1.2.7 void ec_encode_data_update_avx (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char ∗ *g_tbls*, unsigned char ∗ *data*, unsigned char ∗∗ *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires AVX

5.1.2.8 void ec_encode_data_update_avx2 (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char ∗ *g_tbls*, unsigned char ∗ *data*, unsigned char ∗∗ *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires AVX2

5.1.2.9 void ec_encode_data_update_base (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char ∗ *v*, unsigned char ∗ *data*, unsigned char ∗∗ *dest*)

Generate update for encode or decode of erasure codes from single source.

Baseline version of ec_encode_data_update().

5.1.2.10 void ec_encode_data_update_sse (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char ∗ *g_tbls*, unsigned char ∗ *data*, unsigned char ∗∗ *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires SSE4.1

5.1.2.11 void ec_init_tables (int *k*, int *rows*, unsigned char ∗ *a*, unsigned char ∗ *gftbls*)

Initialize tables for fast Erasure Code encode and decode.

Generates the expanded tables needed for fast encode or decode for erasure codes on blocks of data. 32bytes is generated for each input coefficient.

Parameters

k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
a	Pointer to sets of arrays of input coefficients used to encode or decode data.
gftbls	Pointer to start of space for concatenated output tables generated from input coefficients. Must be of size 32∗k∗rows.

Returns

none

5.1.2.12 void gf_2vect_dot_prod_avx (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.

gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.13 void gf_2vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.14 void gf_2vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.15 void gf_2vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 2 accumulate. AVX version of gf_2vect_mad_sse().

Requires AVX

5.1.2.16 void gf_2vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 2 accumulate. AVX2 version of gf_2vect_mad_sse().

Requires AVX2

5.1.2.17 void gf_2vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 2 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

5.1.2.18 void gf_3vect_dot_prod_avx (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3∗32∗vlen byte constant array based on the three sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 3∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.19 void gf_3vect_dot_prod_avx2 (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3∗32∗vlen byte constant array based on the three sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 3∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.20 void gf_3vect_dot_prod_sse (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3∗32∗vlen byte constant array based on the three sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 3∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.21 void gf_3vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 3 accumulate. AVX version of gf_3vect_mad_sse().

Requires AVX

5.1.2.22 void gf_3vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 3 accumulate. AVX2 version of gf_3vect_mad_sse().

Requires AVX2

5.1.2.23 void gf_3vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 3 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

5.1.2.24 void gf_4vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4*32*vlen byte constant array based on the four sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 4*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.25 void gf_4vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4*32*vlen byte constant array based on the four sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 4∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.26 void gf_4vect_dot_prod_sse (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4∗32∗vlen byte constant array based on the four sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 4∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.27 void gf_4vect_mad_avx (int *len,* int *vec,* int *vec_i,* unsigned char ∗ *gftbls,* unsigned char ∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. AVX version of gf_4vect_mad_sse().

Requires AVX

5.1.2.28 void gf_4vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. AVX2 version of gf_4vect_mad_sse().

Requires AVX2

5.1.2.29 void gf_4vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

5.1.2.30 void gf_5vect_dot_prod_avx (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.31 void gf_5vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.32 void gf_5vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.33 void gf_5vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 5 accumulate. AVX version.

Requires AVX

5.1.2.34 void gf_5vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 5 accumulate. AVX2 version.

Requires AVX2

5.1.2.35 void gf_5vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 5 accumulate. SSE version.

Requires SSE4.1

5.1.2.36 void gf_6vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6*32*vlen byte constant array based on the six sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.37 void gf_6vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6*32*vlen byte constant array based on the six sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.38 void gf_6vect_dot_prod_sse (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6∗32∗vlen byte constant array based on the six sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 6∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

5.1.2.39 void gf_6vect_mad_avx (int *len,* int *vec,* int *vec_i,* unsigned char ∗ *gftbls,* unsigned char ∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 6 accumulate. AVX version.

Requires AVX

5.1.2.40 void gf_6vect_mad_avx2 (int *len,* int *vec,* int *vec_i,* unsigned char ∗ *gftbls,* unsigned char ∗ *src,* unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 6 accumulate. AVX2 version.

Requires AVX2

5.1.2.41 void gf_6vect_mad_sse (int *len,* int *vec,* int *vec_i,* unsigned char $*$ *gftbls,* unsigned char $*$ *src,* unsigned char $**$ *dest*)

GF(2^8) vector multiply with 6 accumulate. SSE version.

Requires SSE4.1

5.1.2.42 void gf_gen_cauchy1_matrix (unsigned char $*$ *a,* int *m,* int *k*)

Generate a Cauchy matrix of coefficients to be used for encoding.

Cauchy matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as 1/(i + j) | i != j, i:{0,k-1} j:{k,m-1}. Any sub-matrix of a Cauchy matrix should be invertable.

Parameters

a	[mxk] array to hold coefficients
m	number of rows in matrix corresponding to srcs + parity.
k	number of columns in matrix corresponding to srcs.

Returns

none

5.1.2.43 void gf_gen_rs_matrix (unsigned char $*$ *a,* int *m,* int *k*)

Generate a matrix of coefficients to be used for encoding.

Vandermonde matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as 2^{i*(j-k+1)} i:{0,k-1} j:{k,m-1}. Commonly used method for choosing coefficients in erasure encoding but does not guarantee invertable for every sub matrix. For large k it is possible to find cases where the decode matrix chosen from sources and parity not in erasure are not invertable. Users may want to adjust for k > 5.

Parameters

a	[mxk] array to hold coefficients
m	number of rows in matrix corresponding to srcs + parity.
k	number of columns in matrix corresponding to srcs.

Returns

none

5.1.2.44 unsigned char gf_inv (unsigned char *a*)

Single element GF(2^8) inverse.

Parameters

a	Input element

Returns

Field element b such that a x b = {1}

5.1.2.45 int gf_invert_matrix (unsigned char * *in*, unsigned char * *out*, const int *n*)

Invert a matrix in GF(2^8)

Parameters

in	input matrix
out	output matrix such that [in] x [out] = [I] - identity matrix
n	size of matrix [nxn]

Returns

0 successful, other fail on singular input matrix

5.1.2.46 unsigned char gf_mul (unsigned char *a*, unsigned char *b*)

Single element GF(2^8) multiply.

Parameters

a	Multiplicand a
b	Multiplicand b

Returns

Product of a and b in GF(2^8)

5.1.2.47 void gf_vect_dot_prod (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char * *dest*)

GF(2^8) vector dot product, runs appropriate version.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

5.1.2.48 void gf_vect_dot_prod_avx (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

5.1.2.49 void gf_vect_dot_prod_avx2 (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗ *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

5.1.2.50 void gf_vect_dot_prod_base (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗ *dest*)

GF(2^8) vector dot product, runs baseline version.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients. Only elements 32∗CONST∗j + 1 of this array are used, where j = (0, 1, 2...) and CONST is the number of elements in the array of input coefficients. The elements used correspond to the original input coefficients.
src	Array of pointers to source inputs.
dest	~~Pointer to destination data array.~~

Returns

none

5.1.2.51 void gf_vect_dot_prod_sse (int *len,* int *vlen,* unsigned char ∗ *gftbls,* unsigned char ∗∗ *src,* unsigned char ∗ *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

5.1.2.52 void gf_vect_mad (int *len,* int *vec,* int *vec_i,* unsigned char ∗ *gftbls,* unsigned char ∗ *src,* unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, runs appropriate version.

Does a GF(2^8) multiply across each byte of input source with expanded constant and add to destination array. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

5.1.2.53 void gf_vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires AVX

5.1.2.54 void gf_vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires AVX2

5.1.2.55 void gf_vect_mad_base (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *v*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, baseline version.

Baseline version of gf_vect_mad() with same parameters.

5.1.2.56 void gf_vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires SSE4.1

5.2 gf_vect_mul.h File Reference

Interface to functions for vector (block) multiplication in GF(2^8).

Functions

- int gf_vect_mul_sse (int len, unsigned char ∗gftbl, void ∗src, void ∗dest)

    GF(2^8) vector multiply by constant.
- int gf_vect_mul_avx (int len, unsigned char ∗gftbl, void ∗src, void ∗dest)

    GF(2^8) vector multiply by constant.
- int gf_vect_mul (int len, unsigned char ∗gftbl, void ∗src, void ∗dest)

    GF(2^8) vector multiply by constant, runs appropriate version.
- void gf_vect_mul_init (unsigned char c, unsigned char ∗gftbl)

    Initialize 32-byte constant array for GF(2^8) vector multiply.
- void gf_vect_mul_base (int len, unsigned char ∗a, unsigned char ∗src, unsigned char ∗dest)

    GF(2^8) vector multiply by constant, runs baseline version.

5.2.1 Detailed Description

Interface to functions for vector (block) multiplication in GF(2^8). This file defines the interface to routines used in fast RAID rebuild and erasure codes.

5.2.2 Function Documentation

5.2.2.1 int gf_vect_mul (int *len,* unsigned char ∗ *gftbl,* void ∗ *src,* void ∗ *dest*)

GF(2^8) vector multiply by constant, runs appropriate version.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

5.2.2.2 int gf_vect_mul_avx (int *len*, unsigned char ∗ *gftbl*, void ∗ *src*, void ∗ *dest*)

GF(2^8) vector multiply by constant.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Requires AVX

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

5.2.2.3 void gf_vect_mul_base (int *len*, unsigned char ∗ *a*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply by constant, runs baseline version.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
a	Pointer to 32-byte array of pre-calculated constants based on C. only use 2nd element is used.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

5.2.2.4 void gf_vect_mul_init (unsigned char *c*, unsigned char ∗ *gftbl*)

Initialize 32-byte constant array for GF(2^8) vector multiply.

Calculates array {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} } as required by other fast vector multiply functions.

Parameters

c	Constant input.
gftbl	Table output.

5.2.2.5 int gf_vect_mul_sse (int *len*, unsigned char ∗ *gftbl*, void ∗ *src*, void ∗ *dest*)

GF(2^8) vector multiply by constant.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Requires SSE4.1

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

5.3 types.h File Reference

Defines standard width types.

5.3.1 Detailed Description

Defines standard width types.

INDEX