

各種誤り検出符号の説明

mledc

今回発明した符号。計算内容についてはこの文書の「計算内容」の章を参照。

crc32

PNG などで用いられている CRC32 の計算をそのまま利用。

fletcher32

16bit ずつデータを取得したデータ列(符号なし)を $x_i (i = 0..(N-1))$ とし、

$$a_i = a_{i-1} + x_{i-1}$$

$$b_i = b_{i-1} + a_{i-1}$$

とする。

a_0, b_0 には適当な初期値を入れる。

誤り検出符号の値としては、 a_N, b_N を 65535 で割った余りを並べた 32bit 整数を用いる。

fletcher32

16bit ずつデータを取得したデータ列(符号なし)を $x_i (i = 0..(N-1))$ とし、

$$a_i = a_{i-1} + x_{i-1}$$

$$b_i = b_{i-1} + a_{i-1}$$

$$c_i = c_{i-1} + b_{i-1}$$

$$d_i = d_{i-1} + c_{i-1}$$

とする。

a_0, b_0, c_0, d_0 には適当な初期値を入れる。

誤り検出符号の値としては、 a_N, b_N, c_N, d_N を 255 で割った余りを並べた 32bit 整数を用いる。

計算の手続きとしては、ZFS が利用している fletcher4 とほぼ同じだが、ビット数が異なる。また、最終的な値として剰余を使っているところも違う。

計算内容

データの取得

データは 2byte ずつ取得する。端数がある場合は末尾にもう 1byte ゼロがあることにする。
→ 0x12, 0x34, 0x56 と 0x12, 0x34, 0x56, 0x00 の区別はつかない。

というわけで、入力バイト数の半分(端数切り上げ)個の、符号なし16ビット整数が手に入る

計算

計算に必要な定数

以下の定数を必要とする。

変数名	説明	補足
init	初期値	2進数で0と1がいい感じに混ざっている値がいいんじゃないかと思う。
mul	乗数	2進数で0と1がいい感じに混ざっている 素数 がいいんじゃないかと思う。

計算に必要な変数

実質的に 32bit 符号なし整数1個。この変数の名前を `c` とする。

初期化

`c` を `init` で初期化する

更新

符号なし16bit整数の入力データ `x` を受け取り、以下の計算をする:

$$c \leftarrow \text{rol}(c) \times \text{mul} + x$$

関数 `rol` は、1bit 左ローテート。数学っぽく書くと以下の通り:

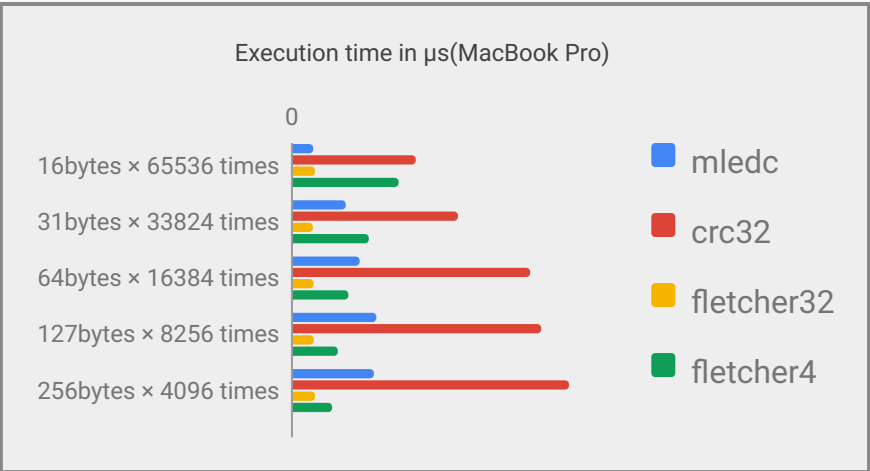
$$\text{rol}(x) = \text{mod}(\lfloor x \times 2 + x \div 2^{31} \rfloor, 2^{32})$$

関数 `mod` は剰余関数。数学っぽく書くと以下の通り:

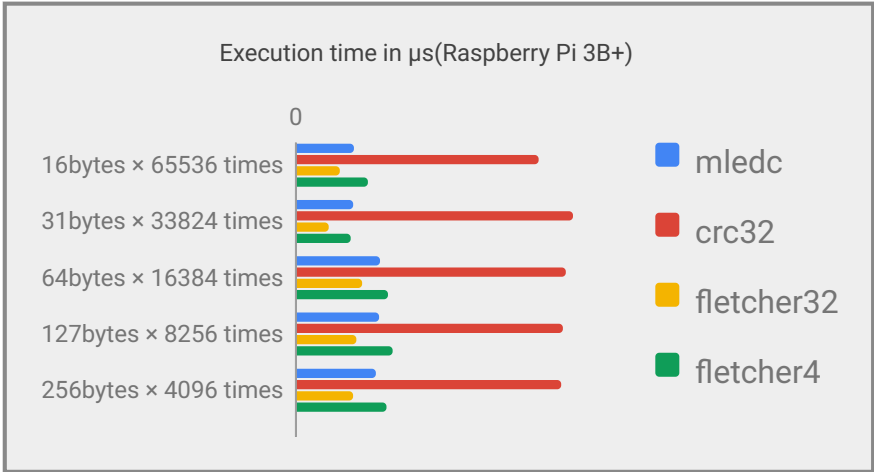
$$\text{mod}(a, n) = a - n \lfloor a \div n \rfloor$$

計算量

mledc は、計算内容が簡単な上にメモリアクセスが少ないので、計算量は CRC32 と比べるとだいぶ少ないが、ほぼ足し算しかしていない fletcher32 と比べると遅い。
MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports) だと



mledc は fletcher32 にだいぶ負けているが、Raspberry Pi 3B+ではそれほど大きな差はない。



誤り検出性能

乱数で 1～255 バイトのデータ列を作り、そこにノイズを付加し、約1.5億回の試行を行った。
誤り検出に失敗した回数は下表の通り：

誤り検出符号	誤り検出失敗回数	誤り検出失敗率(百万分率)	誤り検出失敗率の逆数
mledc	5	0.0322	3107万
crc32	0	0.0000	n/a
fletcher32	118	0.7595	132万
fletcher4	13125	84.4734	1万2000

crc32 は流石。
mledc は 5ミス。fletcher32 と比べると20倍ぐらい良かった。
今回の更新で増やした fletcher4 は、非常に悪かった。速度と誤り検出能力の両面で fletcher32 に負けているので、採用はない。

データ長・誤りビット数・検出失敗数

データ長を、16byte～64byte。誤りビット数を 1bit～16bit まで変化させ、誤り検出失敗数を調査した。
対象とした誤り検出符号は、mledc と fletcher32。
各条件で 100万回 試行した。
結果は下表の通り：

fletcher32

ノイズのビット数。左端が1。右端が16。

データ長。上端が16 bytes。下端が63 bytes。

0	0	161	33	3	1	1	1	0	0	0	0	0	0	0
0	0	156	44	4	0	2	1	0	0	0	0	0	0	0
0	0	135	29	3	3	0	0	1	0	0	0	0	0	0
0	0	112	29	9	3	1	0	1	0	0	0	0	0	0
0	0	116	34	7	2	0	0	0	0	0	0	0	0	0
0	0	119	36	6	1	1	0	0	0	0	0	0	0	0
0	0	88	27	4	3	0	0	0	0	0	0	0	0	0
0	0	136	19	4	0	0	0	0	0	0	0	0	0	0
0	0	94	33	4	2	2	0	0	0	0	0	0	0	0
0	0	111	31	4	3	1	0	0	0	0	0	0	0	0
0	0	104	19	5	1	1	1	0	0	0	0	0	0	0
0	0	92	27	3	1	0	0	0	0	0	0	0	0	0
0	0	97	29	4	0	1	0	0	0	0	0	0	0	0
0	0	98	18	1	1	0	0	0	0	0	0	0	0	0
0	0	84	18	5	0	0	0	0	0	0	0	0	0	0
0	0	92	21	2	1	2	1	0	0	0	0	0	0	0
0	0	87	25	2	3	0	0	0	0	0	0	0	0	0
0	0	77	17	4	2	0	0	1	0	0	0	0	0	0
0	0	85	22	1	1	1	0	0	0	0	0	0	0	0
0	0	89	28	2	0	0	0	0	0	0	0	0	0	0
0	0	77	14	4	0	0	1	0	0	0	0	0	0	0
0	0	89	24	6	2	0	0	0	0	0	0	0	0	0
0	0	62	27	7	0	0	0	0	0	0	0	0	0	0
0	0	85	19	2	2	0	0	0	0	0	0	0	0	0
0	0	74	11	1	0	0	0	0	0	0	0	0	0	0
0	0	70	15	3	1	0	0	0	0	0	0	0	0	0
0	0	73	17	3	1	0	0	0	0	0	0	0	0	0
0	0	58	13	2	6	0	1	0	0	0	0	0	0	0
0	0	64	24	2	0	0	0	0	0	0	0	0	0	0
0	0	57	14	5	1	0	0	0	0	0	0	0	0	0
0	0	56	23	4	0	0	0	0	0	0	0	0	0	0
0	0	53	20	2	1	0	0	0	0	0	0	0	0	0
0	0	65	19	4	0	0	0	0	0	0	0	0	0	0
0	0	56	17	6	1	0	0	0	0	0	0	0	0	0
0	0	43	13	5	0	0	0	0	0	0	0	0	0	0
0	0	49	14	3	1	0	0	0	0	0	0	0	0	0
0	0	49	10	2	1	0	0	0	0	0	0	0	0	0
0	0	50	7	2	0	0	1	0	0	0	0	0	0	0
0	0	48	15	1	3	0	0	0	0	0	0	0	0	0
0	0	46	17	1	1	0	0	0	0	0	0	0	0	0
0	0	51	17	2	1	0	0	0	0	0	0	0	0	0
0	0	43	13	1	0	0	0	0	0	0	0	0	0	0
0	0	52	12	3	0	0	0	0	0	0	0	0	0	0
0	0	46	11	3	0	0	0	0	0	0	0	0	0	0
0	0	41	13	1	1	0	0	0	0	0	0	0	0	0
0	0	48	8	2	0	0	0	0	0	0	0	0	0	0
0	0	53	17	1	0	0	0	0	0	0	0	0	0	0
0	0	39	14	1	2	0	0	0	0	0	0	0	0	0