# 軟體分析與最佳化 Workload Analysis Stage 2

組別: 2

成員: 612410017 林靖紳、612410066 蔡宏遠

### **Review Stage 1 Results**

• 根據 Stage 1 的分析結果,我們打算以 gcc 作為之後主要分析的編譯器

```
GCC
                                 ICC

    -O3

    -O3

         real
                 0m4.940s
                                          real
                                                  0m5.249s
         user
                 0m14.492s
                                          user
                                                  0m13.975s
                                                  0m3.782s
                 0m3.518s
                                          SVS
         SVS
     -00
                                    o -O0
         real
                 0m5.923s
                                          real
                                                  0m5.344s
                  0m19.574s
                                                  0m14.020s
         user
                                          user
                                                  0m4.003s
                  0m5.635s
         sys
                                          SVS
```

### **Execution environments**

CPU information

```
ashen@Stephanie-Lin:~$ lscpu
Architecture:
                         x86_64
  CPU op-mode(s):
                         32-bit, 64-bit
  Address sizes:
                         39 bits physical, 48 bits virtual
                         Little Endian
  Byte Order:
CPU(s):
                         12
  On-line CPU(s) list:
                         0-11
Vendor ID:
                         GenuineIntel
  Model name:
                         11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz
    CPU family:
                         6
    Model:
                         167
    Thread(s) per core:
                         2
    Core(s) per socket:
                         6
    Socket(s):
                         1
    Stepping:
    CPU max MHz:
                         4600.0000
    CPU min MHz:
                         800.0000
    BogoMIPS:
                         5424.00
```

Memory

```
ashen@Stephanie-Lin:~$ free -h
                                                            buff/cache
                                                                          available
               total
                             used
                                          free
                                                    shared
                                                     1.8Gi
                                                                   14Gi
Mem:
                31Gi
                            4.2Gi
                                          12Gi
                                                                               24Gi
               2.0Gi
                                         2.0Gi
Swap:
                               0B
```

#### OS version

```
ashen@Stephanie-Lin:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 22.04.2 LTS
Release: 22.04
Codename: jammy
```

#### GCC version

```
ashen@Stephanie-Lin:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

#### ICC version

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW2$ icc --version icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in the second half of 2023. The Intel(R) oneAPI DPC++/C++ Compiler (ICX) is the recommended compiler moving forward. Please transition to use this compiler. Use '-diag-disable=10441' to disable this message.
icc (ICC) 2021.10.0 20230609
Copyright (C) 1985-2023 Intel Corporation. All rights reserved.
```

### 編譯參數:

```
1
    export CFLAGS="-DUNIX -00 -funroll-loops -fprefetch-loop-arrays ${
    export CXXFLAGS="-DUNIX -00 -funroll-loops -fprefetch-loop-arrays
2
    export CPPFLAGS=""
3
4
    export CXXCPPFLAGS=""
5
    export LDFLAGS="-L${CC_HOME}/lib64 -L${CC_HOME}/lib"
6
    export LIBS=""
    export EXTRA_LIBS=""
7
    export PARMACS MACRO FILE="pthreads"
8
```

### 使用 vtune 初步分析

### gcc -O0

# 

$\odot$	CPU Time ①:	25.519s	
	Effective Time :	25.519s	
	Spin Time ①:	0s	
	Overhead Time ①:	0s	
	Instructions Retired:	170,988,300,000	
$\odot$	Microarchitecture Usage ①:	51.3%	of Pipeline Slots
	CPI Rate <sup>⊕</sup> :	0.632	
	Total Thread Count:	16	
	Paused Time @:	0s	

### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU ③ Time	% of CPU ③ Time
[Outside any known modul e]	[Unknown ]	6.112s	24.0%
rabinseg	dedup 🏲	4.075s	16.0%
deflate_slow	dedup	3.715s	14.6%
pqdownheap	dedup	3.149s	12.3%
sha1_block_data_order	dedup 🏲	2.617s	10.3%
[Others]	N/A*	5.850s	22.9%

<sup>\*</sup>N/A is applied to non-summable metrics.

- 1. 首先我們使用 gcc -O0 編譯發現 hotspot 為 rabinseg 函式,緊接著是第二高的 hotspot 是 deflate slow 函式
- 2. 在這樣的情況下,需要考量優化的函式有兩個,要直接用腦袋想兩個辦法有些麻煩,因此我們先嘗試使用 compiler 的編譯選項做簡單且直接的優化

# 

⊙ CPU Time ②:	16.838s
Effective Time ①:	16.838s
Spin Time ①:	0s
Overhead Time ①:	0s
Instructions Retired:	125,096,400,000

CPI Rate ①: 0.564

Total Thread Count: 16

Paused Time ①: 0s

### ▼ Top Hotspots ★

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time ③	% of CPU Time $\ensuremath{\mathfrak{D}}$
deflate_slow	dedup	3.523s	20.9%
pqdownheap	dedup	3.305s	19.6%
[Outside any known module]	[Unknown]	2.660s	15.8%
sha1_block_data_order	dedup	1.878s	11.2%
rabinseg	dedup	1.440s	8.5%
[Others]	N/A*	4.032s	23.9%

<sup>\*</sup>N/A is applied to non-summable metrics.

- 1. 可以看到 gcc -O3 編譯發現 hotspot 只剩下 deflate\_slow , 並優化了 rabinseg 函式的執行時間比例
- 2. 因此後續主要會針對在 -O3 編譯選項下的 deflate\_slow 函式進行優化

# 討論 -O3 如何優化 rabinseg 函式

#### -O0

				■ 0V204T	JL	more near, energy up;	70.030m3 <b>-</b>
72	int rabinseg(uchar *p, int n, int winlen, u32int * rabinta	0s	5,4	0x3844	93	shll \$0x8, -0x10(%rbp)	183.186ms
73	int i;			0x3848	94	movl -0x14(%rbp), %eax	59.735ms
74	u32int h;			0x384b	94	lea 0x1(%rax), %edx	4.978ms
75	u32int x;			0x384e	94	movl %edx, -0x14(%rbp)	188.164ms
76				0x3851	94	movsxd %eax, %rdx	31.858ms
77	USED(winlen);	0s	2,7	0x3854	94	movq -0x28(%rbp), %rax	61.726ms
78	if(n < NWINDOW)			0x3858	94	add %rdx, %rax	5.973ms
79	return n;			0x385b	94	movzxb (%rax), %eax	173.230ms
80				0x385e	94	movzx %al, %eax	36.836ms
81	h = 0;			0x3861	94	orl %eax, -0x10(%rbp)	464.934ms
82	for(i=0; i <nwindow; i++){<="" td=""><td>0.006s  </td><td>29,7</td><td>0x3864</td><td>95</td><td>movl -0xc(%rbp), %eax</td><td>102.544ms</td></nwindow;>	0.006s	29,7	0x3864	95	movl -0xc(%rbp), %eax	102.544ms
83	x = h >> 24;	0.007s	13,5	0x3867	95	lea (,%rax,4), %rdx	1.991ms
84	h = (h<<8) p[i];	0.014s	102,6	0x386f	95	movq -0x38(%rbp), %rax	19.912ms
85	h ^= rabintab[x];	0.019s	91,8	0x3873	95	add %rdx, %rax	94.580ms
86	}			0x3876	95	movl (%rax), %eax	140.376ms
87	if((h & RabinMask) == 0)			0x3878	95	xorl %eax, -0x10(%rbp)	697.898ms
88	return i;			0x387b	96	movl -0x10(%rbp), %eax	87.611ms
89	while(i <n){< td=""><td>0.191s</td><td>2,157,3</td><td>0x387e</td><td>96</td><td>and \$0xfff, %eax</td><td>99.558ms</td></n){<>	0.191s	2,157,3	0x387e	96	and \$0xfff, %eax	99.558ms
90	x = p[i-NWINDOW];	0.535s	6,369,3	0x3883	96	test %eax, %eax	0.996ms
91	h ^= rabinwintab[x];	0.391s	3,461,4	0x3885	96	jnz 0x388c <block 11=""></block>	238.938ms
92	x = h >> 24;	0.216s	2,092,5	0x3887		Block 10:	
93	h <<= 8;	0.183s	1,325,7	0x3887	97	movl -0x14(%rbp), %eax	0.996ms
94	h  = p[i++];	1.027s	5,915,7	0x388a	97	jmp 0x389b <block 13=""></block>	
95	h ^= rabintab[x];	1.057s	4,368,6	0x388c		Block 11:	
96	if((h & RabinMask) == 0)	0.427s	2,162,7	0x388c	89	movl -0x14(%rbp), %eax	2.987ms
97	return i;	0.001s		0x388f	89	cmpl -0x2c(%rbp), %eax	
98	}			0x3892	89	jl 0x380b <block 9=""></block>	188.164ms
99	return n;			0x3898		Block 12:	
100	}			0x3898	99	movl -0x2c(%rbp), %eax	
101				0x389b		Block 13:	

Source Line ▲	Source	CPU Time	∐ In	15	Address ▲	Source Line	Assembly	
59	for(i=0; i<256; i++)				0x39b9	95	mov %eax, %eax	
60	rabinwintab[i] = fpwinreduce(ir				0x39bb	93	shl \$0x8, %ebx	
61	return;				0x39be	95	movl (%rcx,%rax,4), %eax	
62	}				0x39be 0x39c1	94	or %ebx, %esi	
63					0x39c3	95	xor %esi, %eax	
64	void rabininit(int winlen, u32int * rabin					96	mov \$0x22, %esi	
65	//rabintab = malloc(256*sizeof rabintab				0x39c5			
66	//rabinwintab = malloc(256*sizeof rabin				0x39ca	96	test \$0xfff, %eax	
67	fpmkredtab(irrpoly, 0, rabintab);				0x39cf	96	jz 0x3b20 <block 19=""></block>	
68	fpmkwinredtab(irrpoly, winlen, rabintab				0x39d5		Block 8:	
69	return;				0x39d5	96	cmp \$0x1, %r12	
70	}				0x39d9	96	j <u>z 0x3a50 <block 13=""></block></u>	0ms
71	J				0x39db		Block 9:	
	int rebiness/usher to int n int violen	2.987ms			0x39db	96	cmp \$0x2, %r12	
72	int rabinseg(uchar *p, int n, int winlen,	2.987MS			0x39df	96	jz 0x3a1a <block 11=""></block>	
73	int i;				0x39e1		Block 10:	
74	u32int h;				0x39e1	91	movzxb -0x21(%rdi,%rsi,1), %r9d	
75	u32int x;				0x39e7	94	movzxb -0x1(%rdi,%rsi,1), %ebx	
76					0x39ec	94	mov \$0x23, %esi	
77	USED(winlen);				0x39f1	91	xorl (%r8,%r9,4), %eax	
78	if(n < NWINDOW)				0x39f5	94	mov \$0x22, %r9d	
79	return n;				0x39fb	93	mov %eax, %r12d	
80					0x39fe	92	shr \$0x18, %eax	
81	h = 0;				0x33iC	95	mov %eax, %edx	
82	for(i=0; i <nwindow; i++){<="" td=""><td>0.996ms</td><td></td><td></td><td></td><td></td><td></td><td></td></nwindow;>	0.996ms						
83	x = h >> 24;	0.996ms			0x3a03	93	shl \$0x8, %r12d movl (%rcx,%rdx,4), %eax	
84	h = (h<<8) p[i];	3.982ms			0x3a07	95		
85	h ^= rabintab[x];	3.982ms			0x3a0a	94	or %r12d, %ebx	
86	}			Ш	0x3a0d	95	xor %ebx, %eax	
87	if((h & RabinMask) == 0)	0ms		1	0x3a0f	96	test \$0xfff, %eax	
88	return i;	31113			0x3a14	96	jz 0x3b20 <block 19=""></block>	
89	while(i <n){< td=""><td>1.991ms</td><td></td><td></td><td>0x3a1a</td><td></td><td>Block 11:</td><td></td></n){<>	1.991ms			0x3a1a		Block 11:	
90	x = p[i-NWINDOW];	1.001115		-10	0x3a1a	91	movzxb -0x21(%rdi,%rsi,1), %r9d	
	h ^= rabinwintab[x];	61 706		т.	0x3a20	94	movzxb -0x1(%rdi,%rsi,1), %ebx	
91		61.726ms		н.	0x3a25	91	xorl (%r8,%r9,4), %eax	
92	x = h >> 24;	9.956ms		-11	0x3a29	94	mov %esi, %r9d	
93	h <<= 8;	124.447ms		-11-	0x3a2c	94	add \$0x1, %rsi	
94	h  = p[i++];	237.942ms			0x3a30	93	mov %eax, %r12d	
95	h ^= rabintab[x];	866.150ms			0x3a33	92	shr \$0x18, %eax	
96	if((h & RabinMask) == 0)	124.447ms			0x3a36	95	mov %eax, %eax	0ms
97	return i;				0x3a38	93	shl \$0x8, %r12d	
98	}				0x3a3c	95	movl (%rcx,%rax,4), %eax	0.996ms
99	return n;				0x3a3f	94	or %r12d, %ebx	0.0001110
100	}			ш	0x3a42	95	xor %ebx, %eax	
Course Uses A	0	4 ODLI Time (V			_			6 ODUT-
Source Line ▲	Source for(i=0; i<256; i++)	♠ CPU Time	i In	15	Address A	Source Line	Assembly % 12 d	♠ CPU Tim
59					0x3a6a	93	mov %eax, %r12d	0.000
60	rabinwintab[i] = fpwinreduce(ir				0x3a6d	92	shr \$0x18, %eax	3.982ms
60 61	rabinwintab[i] = fpwinreduce(ir return;				0x3a6d 0x3a70	92 93	shr \$0x18, %eax shl \$0x8, %r12d	41.814ms
60 61 62	rabinwintab[i] = fpwinreduce(ir				0x3a6d 0x3a70 0x3a74	92 93 94	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx	41.814ms 6.969ms
60 61 62 63	<pre>rabinwintab[i] = fpwinreduce(ir return; }</pre>				0x3a6d 0x3a70 0x3a74 0x3a77	92 93 94 95	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx	41.814ms 6.969ms   218.031ms
60 61 62 63 64	<pre>rabinwintab[i] = fpwinreduce(ir return; } void rabininit(int winlen, u32int * rabin</pre>				0x3a6d 0x3a70 0x3a74	92 93 94	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax	41.814ms 6.969ms
60 61 62 63 64	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab</pre>				0x3a6d 0x3a70 0x3a74 0x3a77	92 93 94 95	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx	41.814ms 6.969ms   218.031ms
60 61 62 63 64 65	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabin</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a	92 93 94 95 95	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax	41.814ms 6.969ms   218.031ms
60 61 62 63 64 65 66	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c	92 93 94 95 95 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx	41.814ms 6.969ms   218.031ms 1.991ms
60 61 62 63 64 65 66 67	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabin</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80	92 93 94 95 95 96 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax	41.814ms 6.969ms   218.031ms 1.991ms
60 61 62 63 64 65 66 67	rabinwintab[i] = fpwinreduce(ir return; } void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabin fpmkredtab(irrpoly, 0, rabintab);				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85	92 93 94 95 95 96 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""></block>	41.814ms 6.969ms   218.031ms 1.991ms
60 61 62 63 64 65 66 67 68	rabinwintab[i] = fpwinreduce(ir return; } void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabin fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b	92 93 94 95 95 96 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <8lock 19> Block 15:	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 6
60 61 62 63 63 64 65 66 67 68 88 69	rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabinfpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return;				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b	92 93 94 95 95 96 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %r2d xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d</block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1
600 611 622 633 664 665 666 67 688 669 770	rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91	92 93 94 95 95 96 96 96 91 94	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <8lock 19> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1.991ms   23.894ms 1.991ms   1.991ms
60 61 62 63 64 65 66 66 67 68 69	rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabin fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen,				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a78 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91 0x3a96	92 93 94 95 95 96 96 96 91 94 91	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d</block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.994ms
600 611 622 663 664 665 666 667 668 669 70	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i;</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a80 0x3a8b 0x3a8b 0x3a91 0x3a96 0x3a9d	92 93 94 95 95 96 96 96 91 91 94 94	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <8lock 19> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,*r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx	41.814ms 6.969ms 218.031ms 1.991ms 23.894ms 1.991ms 1.991ms 1.991ms 1.991ms 27.876ms 27.876ms
60 61 62 63 64 64 66 67 68 69 70 71 71 72 73	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab); return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h;</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a80 0x3a8b 0x3a8b 0x3a91 0x3a96 0x3a9d 0x3a9d	92 93 94 95 95 96 96 96 91 94 91 94 94 93	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d</block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   1.991ms   1.991ms   1.994ms 1 1.994m
60 61 62 62 63 64 64 65 66 66 67 68 69 770 771 72 73 774 775	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; } int rabinseg(uchar *p, int n, int winlen, int i;</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91 0x3a96 0x3a9a 0x3a9d 0x3a9d	92 93 94 95 95 96 96 91 94 91 94 94 93 93	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax</block>	41.814ms 6.969ms 218.031ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.992ms 1.9
60 61 62 63 63 64 65 66 66 67 68 89 97 77 77 77 77 77	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91 0x3a9d 0x3a9d 0x3a9d 0x3aa4	92 93 94 95 95 96 96 96 91 94 91 94 94 93	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d</block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   23.894ms 1.991ms   1.991ms   12.942ms 1 27.876ms 1 1.991ms   0ms 23.894ms 1 1.991ms   0ms
600 61 62 63 64 65 66 66 67 68 8 69 70 71 72 73 74 77 77 77 77 77 77 77 77 77 77	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;</pre> USED(winlen);				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a8b 0x3a8b 0x3a91 0x3a96 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aa4	92 93 94 95 95 96 96 96 91 91 94 94 94 93 92 93	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <8lock 19≥ Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx	41.814ms 6.969ms 218.031ms 1.991ms 23.894ms 1.991ms 1.991ms 1.991ms 27.876ms 1.991ms 27.876ms 23.894ms 41.814ms 41.814ms
60 61 62 62 63 64 4 65 66 66 67 70 77 72 73 74 75 76 77 77 78	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; } int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW)</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9d 0x3a9a 0x3a9a 0x3a9d 0x3aad 0x3aad	92 93 94 95 95 96 96 96 91 94 91 94 91 94 93 93 99 93	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx</block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.994ms 1.994ms 1.994ms 1.994ms 27.876ms 1.991ms   0ms 23.894ms 41.814ms 174.226ms
60 61 62 62 63 64 64 65 66 66 67 68 8 69 770 771 72 73 73 74 75 76 777 778 779	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;</pre> USED(winlen);				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91 0x3a91 0x3a94 0x3a94 0x3a94 0x3aa4 0x3aa4	92 93 94 95 95 96 96 91 94 91 94 93 92 93 94 95 95	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %ebx, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax</block>	41.814ms 6.969ms 218.031ms 1.991ms 1.9
60 61 62 62 63 64 65 66 66 66 77 70 771 772 73 74 775 76 778 79 80	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;</pre>				0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a8b 0x3a8b 0x3a9b 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aa4 0x3aa7 0x3aab	92 93 94 95 95 96 96 96 91 94 91 94 93 93 92 93 94 95 95	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax</block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   1.991ms   1.991ms   1.994ms 1 1.994m
60 61 62 652 653 664 665 666 667 688 699 770 771 772 773 774 775 776 777 778 890 890 881	rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int x;  USED(winlen); if(n < NWINDOW) return n;  h = 0;	2.987ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a91 0x3a9d 0x3a9d 0x3a9d 0x3aa4 0x3aa4 0x3aad 0x3aab	92 93 94 95 95 96 96 91 94 91 94 93 92 93 94 95 95	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0x6ff, %eax jz 0x3b20 <block 19=""></block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.994ms 27.876ms 1.991ms   0ms 23.894ms 41.814ms 174.226ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms 1.991ms   1.991ms 1.991ms   1.
600 61 61 62 63 64 64 65 66 66 67 70 77 77 72 73 77 78 79 80 81	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){<="" pre=""></nwindow;></pre>	2.987ms   0.996ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a91 0x3a96 0x3a9a 0x3a9d 0x3a9d 0x3aa1 0x3aa4 0x3aa7 0x3aab 0x3aab	92 93 94 95 95 96 96 96 91 94 91 94 93 93 92 93 94 95 95	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16:</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   1.991ms   1.991ms   1.994ms 1 1.994m
60 61 62 62 63 64 65 66 66 67 68 89 99 770 771 772 73 73 774 775 76 777 778 879 80 0 61 81 88 2 88 3	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;</nwindow;></pre>	2.987ms   0.996ms   0.996ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aa1 0x3aa4 0x3aa1 0x3aab 0x3ab3 0x3ab3 0x3ab3 0x3ab3 0x3ab3	92 93 94 95 96 96 96 91 91 94 91 93 92 93 94 95 96 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   27.876ms 1.991ms   23.894ms 41.814ms 174.226ms 1.991ms   28.872ms 0.996ms   0ms
60 61 62 62 63 64 65 66 66 67 68 89 99 770 771 772 73 73 774 775 76 777 778 879 80 0 61 81 88 2 88 3	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k; USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;&lt;8) p[i];</nwindow;></pre>	2.987ms   0.996ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a91 0x3a96 0x3a9a 0x3a9d 0x3a9d 0x3aa1 0x3aa4 0x3aa7 0x3aab 0x3aab	92 93 94 95 95 96 96 96 91 94 91 94 93 92 93 93 94 95 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16:</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   0ms 23.894ms 41.814ms 174.226ms 1.991ms   28.872ms 0.996ms
600 61 61 62 63 63 64 65 66 66 67 68 69 9 77 77 77 77 77 77 78 80 80 81 82 83	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;</nwindow;></pre>	2.987ms   0.996ms   0.996ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aa1 0x3aa4 0x3aa1 0x3aab 0x3ab3 0x3ab3 0x3ab3 0x3ab3 0x3ab3	92 93 94 95 96 96 96 91 91 94 91 93 92 93 94 95 96 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   27.876ms 1.991ms   23.894ms 41.814ms 174.226ms 1.991ms   28.872ms 0.996ms   0ms
660 61 61 62 652 653 664 655 666 667 688 699 770 771 772 773 774 775 776 777 778 800 811 822 833 834	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k; USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;&lt;8) p[i];</nwindow;></pre>	2.987ms   0.996ms   0.996ms   3.982ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a8b 0x3a8b 0x3a9b 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aa4 0x3aa7 0x3aab 0x3aab 0x3aab 0x3aab	92 93 94 95 95 96 96 96 91 94 91 94 93 93 92 93 94 95 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %r9d</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   0ms 23.894ms 1 1.4.226ms 1.991ms   28.872ms 0.996ms   0.996ms   0ms 23.894ms 1 1.991ms   28.872ms 1.994ms
660 61 61 62 63 63 64 655 66 66 67 70 71 72 73 74 77 78 87 79 80 80 81 81 82 83 84 85 86	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24; h = (h&lt;80)[p[i]; h ^= rabintab[x];</nwindow;></pre>	2.987ms   0.996ms   0.996ms   3.982ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a9a 0x3a9a 0x3a9d 0x3a9d 0x3a9d 0x3aad 0x3aad 0x3aad 0x3aad 0x3aab 0x3aab 0x3aab 0x3aab 0x3aab 0x3aab	92 93 94 95 96 96 96 91 94 91 93 92 93 94 95 95 96 96	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb -0x1f(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   27.876ms 1.991ms   27.872ms 1.991ms   28.872ms 0.996ms   23.894ms 0.996ms   0.
60 61 62 62 63 64 65 66 66 67 68 68 69 9 770 771 772 73 73 774 75 76 677 78 8 79 80 0 81 81 82 83 84 4 85 66 68 77	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;&lt;8)[p[i];     h ^= rabintab[x]; }</nwindow;></pre>	2.987ms   0.996ms   0.996ms   3.982ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a91 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aad 0x3aad 0x3aab 0x3aab 0x3ab3	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d</block></block>	41.814ms   6.969ms   218.031ms   1.991ms   1.9
50 51 52 53 54 55 55 56 67 58 89 70 71 71 72 73 74 75 76 77 77 78 89 99 99 90 90 90 90 90 90 90 9	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;8)[p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0)</nwindow;></pre>	2.987ms    0.996ms    0.996ms    3.982ms    0ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7a 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9d 0x3a9d 0x3aa1 0x3aa4 0x3aab 0x3aab 0x3aba	92 93 94 95 96 96 96 91 94 91 93 92 93 94 95 96 96 91 94 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1 1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   27.876ms 1.991ms   28.872ms 1.991ms   28.872ms 1.996ms   23.894ms 1.996ms   23.896ms   23.896ms   23.896ms   23.896ms   23.896ms   23.896ms   23.896ms   23.896ms   23.896
500 511 512 532 533 544 555 566 57 588 599 70 771 772 773 774 775 778 799 900 911 912 932 933 944 955 966 977 978 989 980 980 981 981 983 988	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n){< pre=""></n){<></nwindow;></pre>	2.987ms   0.996ms   0.996ms   3.982ms			0x3a6d 0x3a70 0x3a77 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a91 0x3a96 0x3a9a 0x3a9a 0x3a9d 0x3aa1 0x3aad 0x3aad 0x3aac 0x3ab3	92 93 94 95 95 96 96 96 91 94 91 94 93 93 92 93 95 96 96 99 91 94 94 95 96 96	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %ex, %r12d shr \$0x18, %eax</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1.991ms   1.991ms   1.991ms   1.992ms   1.992ms   1.992ms 1.992ms 1.992ms   1.992ms 1.992ms   1.992ms 1.992ms   1.992ms 1.992ms   1.992ms
500 511 52 533 544 555 566 577 588 599 700 71 71 72 72 73 74 77 78 79 90 90 90 91 91 91 93 93 94 94 95 96 96 96	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0)     return i; while(i<n){ pre="" x="p[i-NWINDOW];" }<=""></n){></nwindow;></pre>	2.987ms   0.996ms   0.996ms   0.996ms   3.982ms   3.982ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9a 0x3a9a 0x3a9d 0x3a9a 0x3a9d 0x3aa4 0x3aa5 0x3ab3	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 91 94 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d</block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.994ms 1.991ms   27.876ms 1.991ms   23.894ms 41.814ms 174.226ms 1.991ms   28.872ms 0.996ms   0.996ms   16.925ms 0.996ms   23.894ms
500 51 51 52 533 534 555 566 567 70 71 77 77 78 79 90 90 90 91	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int x;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;8)[p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0)     return i; while(i<n){ ^="rabinwintab[x];&lt;/pre" h="" x="p[i-NWINDOW];" }=""></n){></nwindow;></pre>	2.987ms    0.996ms   0.996ms   3.982ms   3.982ms   0ms  1.991ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a77 0x3a7a 0x3a7c 0x3a8b 0x3a8b 0x3a8b 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3ab0 0x3aad 0x3aad 0x3aad 0x3aad 0x3ab0	92 93 94 95 96 96 96 91 94 91 93 92 93 94 95 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r12d shr \$0x18, %eax shl \$0x8, %r12d shr \$0x18, %eax mov %ebx, %r12d shr \$0x18, %eax mov %ebx, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx</block></block>	41.814ms 6.969ms 218.031ms 1.991ms 27.876ms 1.991ms 1.991ms 23.894ms 0.996ms 1.991ms 23.894ms 0.996ms 1.991ms 23.894ms 0.996ms 1.991ms 28.872ms 0.996ms 1.991ms 28.872ms 0.996ms 1.992ms 23.894ms 0.996ms 1.992ms 23.894ms 0.996ms 1.992ms 1.992ms 1.992ms 1.992ms 1.992ms 1.9937ms 1.9937m
500 511 512 533 534 544 555 566 67 788 69 770 771 772 78 78 78 79 90 111 122 333 344 355 366 387 388 389 390 300	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); return; }  int rabinseg(uchar *p, int n, int winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i=NWINDOW; i++){     x = h &gt; 24;     h = (n&lt;8)[p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n){ ^="rabinwintab[x];" h="" x="h">&gt; 24; }</n){></pre>	2.987ms    0.996ms   0.996ms   3.982ms   3.982ms    0ms  1.991ms    61.726ms   9.956ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7a 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9d 0x3a9d 0x3a9d 0x3a9d 0x3aab 0x3aab 0x3aab 0x3aab 0x3ab 0x3ab 0x3ab 0x3ab 0x3ab 0x3ab 0x3ab 0x3ab 0x3ab	92 93 94 95 96 96 96 91 94 91 94 93 94 95 96 99 91 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 9x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d shr \$0x18, %eax shl \$0x8, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx</block></block>	41.814ms   6.969ms   218.031ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   22.876ms   1.991ms   23.894ms   41.814ms   174.226ms   1.991ms   28.872ms   0.996ms   23.894ms   0.996ms   23.894ms   23.894ms   0.996ms   23.894ms    23.894ms   23.89
500 511 512 52 533 544 555 566 577 588 599 770 771 772 773 774 775 778 779 930 931 932 933 944 955 966 977 979 970 971 971 972 973 974 975 979 975 976 977 977 978 979 979 970 970 971 971 972 973 974 975 976 977 978 979 970 970 970 970 970 970 970 970 970	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h">&gt; 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n){ ^="rabinwintab[x];" h="" x="h">&gt; 24;     h &lt;&lt;= 8;</n){></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms    0ms  1.991ms    61.726ms   9.956ms   124.447ms			0x3a6d 0x3a70 0x3a77 0x3a74 0x3a77 0x3a77 0x3a80 0x3a85 0x3a8b 0x3a91 0x3a96 0x3a94 0x3a94 0x3aa4 0x3aa7 0x3aab 0x3aba 0x3ab3	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 99 91 94 94 93 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %ebx, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %reax,4), %edx mov %edx, %eax</block></block>	41.814ms 6.969ms 218.031ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 22.894ms 1.991ms 1.9990ms 1.9990
500 511 52 533 544 555 566 577 578 588 599 770 771 772 773 774 775 778 779 800 811 822 833 844 835 856 866 877 888 899 900 911 922 933 944	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0)     return i; while(i<n){ *="rabinwintab[x];" h="" x="h"> 24;     h &lt;= rabinwintab[x];     x = h &gt; 24;     h &lt;= s;     h  = p[i++];</n){></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms    0ms  1.991ms    61.726ms   9.256ms   124.447ms   237.942ms			0x3a6d 0x3a70 0x3a77 0x3a7a 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a91 0x3a96 0x3a9a 0x3a9d 0x3aa4 0x3aa4 0x3aa5 0x3ab3 0x3aa6 0x3ab3	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 99 99 99 99 99 99 99 99	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz_0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%r6,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r3d lea 0x3(%rsi), %rbx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax test \$0xfff, %eax</block></block>	41.814ms   6.969ms   218.031ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   22.876ms   1.991ms   23.894ms   41.814ms   174.226ms   1.991ms   28.872ms   0.996ms   23.894ms   0.996ms   23.894ms   23.894ms   0.996ms   23.894ms    23.894ms   23.89
500 511 522 533 534 555 566 577 588 599 770 771 772 788 799 500 531 532 533 544 535 536 537 538 539 500 501 502 503 503 503 503 504 505 505 505 505 505 505 505 505 505	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int h; u32int k; USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i=NWINDOW); i++){     x = h &gt;&gt; 24;     h = rabintab[x];     y if((h &amp; RabinMask) == 0)     return i; while(i<n){ h="rabinwintab[x];" x="h">&gt; 24;     h &lt;&lt;= 8;     h  = p[i++];     h &lt;= rabintab[x]; }</n){></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms   3.982ms    0ms  1.991ms   61.726ms   9.956ms   124.447ms   237.942ms   866.150ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9a 0x3a9a 0x3a9d 0x3aa1 0x3aa4 0x3aab 0x3aab 0x3aba	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 99 91 94 94 93 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x1(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb -0x1f(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d shr \$0x18, %r9,4), %eax xorl (%r8,%r9,4), %eax xorl (%r8,%r9,4), %eax xorl (%rsi,%rsi,1), %r9d movzxb =0x1f(xrdi,%rsi,1), %r9d movzxb =0x1f(xrdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax test \$0x6fr, %eax test \$0x6ff, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""></block></block></block>	41.814ms 6.969ms 218.031ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 22.894ms 1.991ms 1.9990ms 1.9990
500 511 512 533 534 545 555 566 57 588 599 70 71 71 72 73 744 75 76 77 78 78 79 9 9 9 9 9 9 9 9 9 9 9 9 9 9	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;8)[p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n){ ^="rabintab[x];" h="" x="h"> 24;     h &lt;= 8;     h  = p[i++];     h *= rabintab[x]; if((h &amp; RabinMask) == 0)</n){></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms    0ms  1.991ms    61.726ms   9.256ms   124.447ms   237.942ms			0x3a6d 0x3a70 0x3a77 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a9a 0x3a9a 0x3a9d 0x3a9d 0x3a9d 0x3aad 0x3aad 0x3aad 0x3aba	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d sr \$0x18, %eax shl \$0x8, %r12d sr \$0x18, %eax shl \$0x8, %r12d shr \$0x18, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 17:</block></block></block>	41.814ms 6.969ms 218.031ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 22.894ms 1.991ms 1.9990ms 1.9990
500 511 512 532 533 544 555 566 57 588 599 70 771 772 773 774 775 778 779 900 511 512 52 533 534 535 536 537 538 539 500 500 500 500 500 500 500 500 500 50	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n)( ^="rabintab[x];" h="" x="h"> 24; h &lt;&lt;= 8; h  = p[i++); h ^= rabintab[x]; if((h &amp; RabinMask) == 0) return i;</n)(></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms   3.982ms    0ms  1.991ms   61.726ms   9.956ms   124.447ms   237.942ms   866.150ms			0x3a6d 0x3a70 0x3a74 0x3a77 0x3a7a 0x3a7c 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9a 0x3a9a 0x3a9d 0x3aa1 0x3aa4 0x3aab 0x3aab 0x3aba	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 99 99 99 99 99 99 99 99	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 17: movzxb -0x1e(%rdi,%rsi,1), %r9d</block></block></block>	41.814ms 6.969ms 218.031ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 1.991ms 22.894ms 1.991ms 1.9990ms 1.9990
660 61 61 62 63 63 664 655 666 677 688 699 770 771 772 773 774 775 776 777 778 89 80 81 82 83 83 84 85 86 87 88 88 89 99 90 91 91 92 93 94	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab fpmkredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;8)[p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n){ ^="rabintab[x];" h="" x="h"> 24;     h &lt;= 8;     h  = p[i++];     h *= rabintab[x]; if((h &amp; RabinMask) == 0)</n){></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms   3.982ms    0ms  1.991ms   61.726ms   9.956ms   124.447ms   237.942ms   866.150ms			0x3a6d 0x3a70 0x3a77 0x3a74 0x3a77 0x3a7a 0x3a80 0x3a85 0x3a8b 0x3a9a 0x3a9a 0x3a9d 0x3a9d 0x3a9d 0x3aad 0x3aad 0x3aad 0x3aba	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax shl \$0x8, %r12d sr \$0x18, %eax shl \$0x8, %r12d sr \$0x18, %eax shl \$0x8, %r12d shr \$0x18, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 17:</block></block></block>	41.814ms 6.969ms   218.031ms 1.991ms   18.916ms 1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   0ms 23.894ms 1.991ms   28.872ms 0.996ms   23.894ms 1.9996ms   23.996ms   23.994ms 1.9996ms   23.994
60 61	<pre>rabinwintab[i] = fpwinreduce(ir return; }  void rabininit(int winlen, u32int * rabin //rabintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab //rabinwintab = malloc(256*sizeof rabintab); fpmkwinredtab(irrpoly, 0, rabintab); fpmkwinredtab(irrpoly, winlen, rabintab return; }  int rabinseg(uchar *p, int n, int winlen, int i; u32int h; u32int k;  USED(winlen); if(n &lt; NWINDOW) return n;  h = 0; for(i=0; i<nwindow; i++){="" x="h"> 24;     h = (h&lt;&lt;8) p[i];     h ^= rabintab[x]; } if((h &amp; RabinMask) == 0) return i; while(i<n)( ^="rabintab[x];" h="" x="h"> 24; h &lt;&lt;= 8; h  = p[i++); h ^= rabintab[x]; if((h &amp; RabinMask) == 0) return i;</n)(></nwindow;></pre>	2.987ms    0.996ms   0.996ms   0.996ms   3.982ms   3.982ms    0ms  1.991ms   61.726ms   9.956ms   124.447ms   237.942ms   866.150ms			0x3a6d 0x3a70 0x3a77 0x3a77 0x3a77 0x3a77 0x3a80 0x3a85 0x3a8b 0x3a8b 0x3a9a 0x3a9a 0x3a9a 0x3a9d 0x3aa1 0x3aa4 0x3aa2 0x3ab3	92 93 94 95 96 96 96 91 94 91 94 93 92 93 94 95 96 96 97 98 99 99 99 99 99 99 99 99 99	shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %ebx xorl (%rcx,%rax,4), %ebx mov %ebx, %eax lea 0x1(%rsi), %rbx test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 15: movzxb -0x21(%rdi,%rbx,1), %r9d movzxb -0x1(%rdi,%rbx,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x2(%rsi), %rbx mov %eax, %r12d shr \$0x18, %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 16: movzxb -0x1f(%rdi,%rsi,1), %r9d movzxb 0x1(%rdi,%rsi,1), %edx xorl (%r8,%r9,4), %eax mov %ebx, %r9d lea 0x3(%rsi), %rbx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %eax sht \$0x8, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %eax, %r12d or %r12d, %edx xorl (%rcx,%rax,4), %edx mov %edx, %eax test \$0xfff, %eax jz 0x3b20 <block 19=""> Block 17: movzxb -0x1e(%rdi,%rsi,1), %r9d</block></block></block>	41.814ms 6.969ms   218.031ms 1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   1.991ms   27.876ms 1.991ms   23.894ms 1.991ms   28.872ms 0.996ms   1.992ms   23.894ms 1.992ms   23.894ms 1.992ms   23.894ms 1.992ms   23.894ms 1.992ms   23.894ms 1.993ms 1.9

Source Line ▲	Source	♦ CPU Time    ■	Ins	Address ▲	Source Line	Assembly	
59	for(i=0; i<256; i++)			0x3ac0	94	movzxb 0x1(%rdi,%rsi,1), %edx	23.894ms
60	rabinwintab[i] = fpwinreduce(ir			0x3ac5	91	xorl (%r8,%r9,4), %eax	0.996ms
61	return;			0x3ac9	94	mov %ebx, %r9d	16.925ms
62	}			0x3acc	94	lea 0x3(%rsi), %rbx	0.996ms
63				0x3ad0	93	mov %eax, %r12d	23.894ms
64	void rabininit(int winlen, u32int * rabin			0x3ad3	92	shr \$0x18, %eax	2.987ms
65	//rabintab = malloc(256*sizeof rabintab			0x3ad6	93	shl \$0x8, %r12d	21.903ms
66	//rabinwintab = malloc(256*sizeof rabin			0x3ada	94	or %r12d, %edx	0ms
67	fpmkredtab(irrpoly, 0, rabintab);			0x3add	95	xorl (%rcx,%rax,4), %edx	238.938ms
68	fpmkwinredtab(irrpoly, winlen, rabintab			0x3ae0	95	mov %edx, %eax	0.996ms
69	return;			0x3ae2	96	test \$0xfff, %eax	30.863ms
70	}			0x3ae7	96	jz 0x3b20 <block 19=""></block>	
71				0x3ae9		Block 17:	
72	int rabinseg(uchar *p, int n, int winlen,	2.987ms		0x3ae9	91	movzxb -0x1e(%rdi,%rsi,1), %r9d	
73	int i;			0x3aef	91	xorl (%r8,%r9,4), %eax	16.925ms
74	u32int h;			0x3af3	94	mov %ebx, %r9d	
75	u32int x;			0x3af6	91	mov %eax, %edx	17.920ms
76				0x3af8	93	mov %eax, %r12d	0ms
77	USED(winlen);			0x3afb	94	movzxb 0x2(%rdi,%rsi,1), %eax	28.872ms
78	if(n < NWINDOW)			0x3b00	94	add \$0x4, %rsi	0.996ms
79	return n;			0x3b04	93	shl \$0x8, %r12d	10.951ms
80				0x3b08	92	shr \$0x18, %edx	2.987ms
81	h = 0;			0x3b0b	94	or %r12d, %eax	24.889ms
82	for(i=0; i <nwindow; i++){<="" td=""><td>0.996ms</td><td></td><td>0x3b0e</td><td>95</td><td>xorl (%rcx,%rdx,4), %eax</td><td>228.982ms</td></nwindow;>	0.996ms		0x3b0e	95	xorl (%rcx,%rdx,4), %eax	228.982ms
83	x = h >> 24;	0.996ms		0x3b11	96	test \$0xfff, %eax	
84	h = (h<<8) p[i];	3.982ms	-1	0x3b16	96	jnz 0x3a50 <block 13=""></block>	44.801ms
85	h ^= rabintab[x];	3.982ms	-1	0x3b1c		Block 18:	
86	}		-1	0x3b1c	96	nopl %eax, (%rax)	
87	if((h & RabinMask) == 0)	0ms	-1	0x3b20		Block 19:	
88	return i;			0x3b20	94	mov %r9d, %r10d	
89	while(i <n){< td=""><td>1.991ms</td><td>-1</td><td>0x3b23</td><td></td><td>Block 20:</td><td></td></n){<>	1.991ms	-1	0x3b23		Block 20:	
90	x = p[i-NWINDOW];		-1	0x3b23	100	popq %rbx	
91	h ^= rabinwintab[x];	61.726ms	-1	0x3b24	100	mov %r10d, %eax	
92	x = h >> 24;	9.956ms	-1	0x3b27	100	popq %r12	
93	h <<= 8;	124.447ms	-1	0x3b29	100	popq %rbp	
94	h  = p[i++];	237.942ms		0x3b2a	100	retq	
95	h ^= rabintab[x];	866.150ms		0x3b2b		Block 21:	
96	if((h & RabinMask) == 0)	124.447ms		0x3b2b	100	mov \$0x20, %r10d	
97	return i;			0x3b31	100	popq %rbx	
98	}			0x3b32	100	popq %r12	
99	return n;			0x3b34	100	mov %r10d, %eax	
100	}			0x3b37	100	popq %rbp	

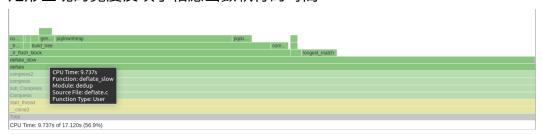
#### • 分析

- 1. 可以看到組語的部份 -O3 將 -O0 一共五行的程式碼分為三部份去做亂序執行
- 2. 分析原因應該是因為亂序執行允許 CPU 在執行指令時根據可用的執行單元和 資源來選擇最優的執行順序,提高程式並行性,從而在同一時鐘周期內執行更 多的指令 (Instruction per cycle)
- 3. 這個結果可以從上面第三部份的 gcc -O3 vtune 分析圖中看到其 CPI 確實小於 gcc -O0 (Cycle per instruction)

# Flame Graph



- 從 flame graph 中可以看到以下幾點:
  - 整個程式的執行流程
  - 函數呼叫的頻率與深度
    - 深度較深的函數呼叫可能是潛在的性能瓶頸,像是 deflate\_slow 函式
  - 耗時的操作
    - 矩形區塊的寬度反映了相應函數執行的時間



# **Sampling**

#### **CPI** Rate

可以從 CPI 看到 -O3 程式的平行化程度有明顯的提昇

• CPI (-O0) : 0.632

• CPI (-O3): 0.564

### **Memory**

可以看到下圖使用 -O3 與 -O0 比較在 load/store 還有 LLC miss 的次數都大幅度的降低

• gcc -O0

# 

	•		
	CPU Time :	20.030s	
$\odot$	Memory Bound ①:	7.1%	of Pipeline Slots
	L1 Bound :	14.1%	of Clockticks
	L2 Bound :	2.0%	of Clockticks
	L3 Bound :	1.9%	of Clockticks
	○ DRAM Bound ②:	1.0%	of Clockticks
	DRAM Bandwidth Bound ①:	0.0%	of Elapsed Time
	Store Bound ®:	1.4%	of Clockticks
	Loads:	53,766,070,468	
	Stores:	24,477,828,827	
	LLC Miss Count ®:	394,603	
	Total Thread Count:	16	
	Paused Time :	0s	

# 

CPU Time 1: 11.712s 8.3% of Pipeline Slots L1 Bound 3: of Clockticks 16.0% L2 Bound 3: 0.0% of Clockticks L3 Bound 3: 0.9% of Clockticks of Clockticks 1.4% DRAM Bandwidth Bound 1: 0.0% of Elapsed Time of Clockticks Store Bound 9: 2.3% Loads: 20,190,359,746 Stores: 8,824,850,780 LLC Miss Count 1: 329,261 Total Thread Count: 16 Paused Time 1: 0s

# 是否有使用 compiler directive 的機會

在 deflate\_slow 函式中使用率最高的 hotspot 就是 INSERT\_STRING,這個函式主要 是將 str 插入到 hash dictionary 中,並記錄具有相同 hash key 的最近字串的資訊

Source	CPU Time: Total	CPU	Address ▲	Source Line	Assembly	♦ CPU Time: Total
no beccer macen at the next window position.			UXUDOU	1023	tea oxiiiuc(xiiip), xiiio	
1						
			0xd54d	1557	pushq %rbp	
int flush;			0xd54e	1557	mov %esi, %ebp	
{			0xd550	1557	pushq %rbx	
IPos hash_head = NIL; /* head of hash chain */			0xd551	1557	mov %rdi, %rbx	
int bflush; /* set if current block m			0xd554	1557	sub \$0x18, %rsp	
			0xd558	1568	movl 0xa4(%rdi), %esi	
/* Process the input block. */			0xd55e	1568	data16 nop	
for (;;) {			0xd560		Block 2:	
/* Make sure that we always have enough lookar			0xd560	1568	cmp \$0x105, %esi	
* at the end of the input file. We need MAX_N			0xd566	1568	jbe 0xd70e <block 22=""></block>	
* for the next match, plus MIN_MATCH bytes to			0xd56c		Block 3:	
* string following the next match.			0xd56c	1580	movl 0x9c(%rbx), %eax	0.2%
*/			0xd572		Block 4:	
if (s->lookahead < MIN_LOOKAHEAD) {			0xd572	1580	movq 0x50(%rbx), %rcx	0.5%
fill_window(s);	0.1%		0xd576	1580	lea 0x2(%rax), %esi	0.1%
if (s->lookahead < MIN_LOOKAHEAD && flush			0xd579	1580	movl 0x70(%rbx), %edi	0.1%
return need_more;			0xd57c	1580	mov %eax, %r11d	
}			0xd57f	1580	movg 0x68(%rbx), %r9	
if (s->lookahead == 0) break; /* flush the			0xd583	1580	movq 0x60(%rbx), %r15	
}			0xd587	1580	movzxb (%rcx,%rsi,1), %r8d	
·			0xd58c	1580	movl 0x80(%rbx), %ecx	0.5%
/* Insert the string window[strstart strsta						0.4%
* dictionary, and set hash head to the head of						0.1%
*/						0.2%
if (s->lookahead >= MIN MATCH) {						
	11 6%					0.1%
}	221070					0.4%
,						0.5%
/* Find the longest match, discarding those <=						6.4%
*/						2.0%
	0.496			1000		2.0%
	0.470			1505		0.4%
	IPos hash_head = NIL; /* head of hash chain */ int bflush; /* set if current block i  /* Process the input block. */ for (;;) {  /* Make sure that we always have enough lookal	local block_state deflate_slow(s, flush)  deflate_state "s;  int flush;  {    IPos hash_head = NIL;	local block_state deflate_slow(s, flush)  deflate_state 's; int flush;  {    IPOs hash_head = NIL;	local block_state deflate_slow(s, flush)  deflate_state's;  int flush;  {    Ipos hash_head = NIL;	local block_state deflate_slow(s, flush)	local block_state deflate_slow(s, flush)

#### 且在 INSERT\_STRING 函式的部份已使用 compiler directive

```
/* ------
* Insert string str in the dictionary and set match head to the previous head
* of the hash chain (the most recent string with same hash key). Return
* the previous length of the hash chain.
* If this file is compiled with -DFASTEST, the compression level is forced
* to 1, and no hash chains are maintained.
* IN assertion: all calls to to INSERT_STRING are made with consecutive
     input characters and the first MIN MATCH bytes of str are valid
     (except for the last MIN_MATCH-1 bytes of the input file).
*/
#ifdef FASTEST
#define INSERT_STRING(s, str, match_head) \
  (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
   match_head = s->head[s->ins_h], \
   s->head[s->ins_h] = (Pos)(str))
#else
#define INSERT_STRING(s, str, match_head) \
  (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
   match head = s->prev[(str) & s->w mask] = s->head[s->ins h], \
   s \rightarrow head[s \rightarrow ins h] = (Pos)(str)
#endif
```

## 是否有使用 SIMD 的機會

同樣是 INSERT\_STRING 作為 hotspot,反組譯後在 vtune 上並沒有看到使用 SIMD 的痕跡

Source Line ▲	Source	& CPU Time: Total	CPU	Address ▲	Source Line	Assembly	
LUUZ	no beccer macen at the next window position.			UXUSSU	1023	tea oxiiiuc(xxiip), xxiis	
1553	*/			0xd544	1557	pushq %r12	
1554	local block_state deflate_slow(s, flush)			0xd546	1623	lea 0x120d3(%rip), %r12	
1555	deflate_state *s;			0xd54d	1557	pushq %rbp	
1556	int flush;			0xd54e	1557	mov %esi, %ebp	
1557	{			0xd550	1557	pushq %rbx	
1558	IPos hash_head = NIL; /* head of hash chain */			0xd551	1557	mov %rdi, %rbx	
1559	int bflush; /* set if current block m			0xd554	1557	sub \$0x18, %rsp	
1560				0xd558	1568	movl 0xa4(%rdi), %esi	
1561	/* Process the input block. */			0xd55e	1568	data16 nop	
1562	for (;;) {			0xd560		Block 2:	
1563	/* Make sure that we always have enough lookal			0xd560	1568	cmp \$0x105, %esi	
1564	* at the end of the input file. We need MAX_N			0xd566	1568	jbe 0xd70e <block 22=""></block>	
1565	* for the next match, plus MIN_MATCH bytes to			0xd56c		Block 3:	
1566	* string following the next match.			0xd56c	1580	movl 0x9c(%rbx), %eax	0.2%
1567	*/			0xd572		Block 4:	
1568	if (s->lookahead < MIN_LOOKAHEAD) {			0xd572	1580	movq 0x50(%rbx), %rcx	0.5%
1569	fill_window(s);	0.1%		0xd576	1580	lea 0x2(%rax), %esi	0.1%
1570	if (s->lookahead < MIN_LOOKAHEAD && flush			0xd579	1580	movl 0x70(%rbx), %edi	0.1%
1571	return need_more;			0xd57c	1580	mov %eax, %r11d	
1572	}			0xd57f	1580	movq 0x68(%rbx), %r9	
1573	if (s->lookahead == 0) break; /* flush the			0xd583	1580	movg 0x60(%rbx), %r15	
1574	}			0xd587	1580	movzxb (%rcx,%rsi,1), %r8d	
1575				0xd58c	1580	movl 0x80(%rbx), %ecx	0.5%
1576	/* Insert the string window[strstart strsta			0xd592	1580	andl 0x4c(%rbx), %r11d	0.4%
1577	* dictionary, and set hash_head to the head (			0xd596	1580	shl %cl, %edi	0.1%
1578	*/			0xd598	1580	xor %edi, %r8d	0.2%
1579	if (s->lookahead >= MIN MATCH) {			0xd59b	1580	andl 0x7c(%rbx), %r8d	
1580	<pre>INSERT_STRING(s, s-&gt;strstart, hash_head);</pre>	11.6%		0xd59f	1580	lea (%r9,%r8,2), %r10	0.1%
1581	}			0xd5a3	1580	movl %r8d, 0x70(%rbx)	0.4%
1582	•			0xd5a7	1580	movzxw (%r10), %r14d	0.5%
1583	/* Find the longest match, discarding those <=			0xd5ab	1580	movw %r14w, (%r15,%r11,2)	6.4%
1584	*/			0xd5b0	1580	movw %ax, (%r10)	2.0%
1585	s->prev length = s->match length, s->prev match	0.4%		0xd5b4	2000	Block 5:	2.070
1586	s->match_length = MIN_MATCH-1;	0.470		0xd5b4	1585	movl 0x90(%rbx), %edx	0.4%

但我認為使用 SIMD 的機會是有的,可以將需要插入 hash dictionary 的 string 和 string 準備插入的相關資訊 buffer 起來,然後在批次記錄這些具有相同 hash key 的字串,因此就可以使用 SIMD 指令做插入,大幅度提高 INSERT\_STRING 的並行處理效率

- 結論: 以上分析出 gcc -O3 中 deflate slow 函式是整個執行過程中的熱點
- 觀察 deflate slow 函式的 source code

```
1554 local block state deflate slow(s, flush)
1555
        deflate state *s;
1556
        int flush;
1557 {
1558
        IPos hash_head = NIL;  /* head of hash chain */
1559
        int bflush;
                                  /* set if current block must be flushed */
1560
1561
        /* Process the input block. */
1562
        for (;;) {
1563
            /* Make sure that we always have enough lookahead, except
1564
             * at the end of the input file. We need MAX MATCH bytes
             * for the next match, plus MIN MATCH bytes to insert the
1565
1566
             * string following the next match.
1567
1568
            if (s->lookahead < MIN LOOKAHEAD) {</pre>
1569
                fill window(s);
1570
                if (s->lookahead < MIN LOOKAHEAD && flush == Z NO FLUSH) {</pre>
1571
                     return need more;
1572
1573
                if (s->lookahead == 0) break; /* flush the current block */
1574
            }
1575
1576
            /* Insert the string window[strstart .. strstart+2] in the
1577
            * dictionary, and set hash_head to the head of the hash chain:
1578
1579
            if (s->lookahead >= MIN MATCH) {
1580
                INSERT STRING(s, s->strstart, hash head);
1581
            }
1500
```

- 目的:用於實現「slow」壓縮策略
- 大致可以分成以下幾個步驟
  - 透過迴圈處理輸入的 blocks
  - 確保始終有足夠的 lookahead 進行處理,需要時填充window
  - 插入當前字串到字典中
  - 尋找最長匹配,捨棄比前一個匹配短的匹配
  - 根據壓縮策略處理匹配和文字,更新 hash table
  - 在需要時刷新 block

### 認為可以優化的地方

- 如果是對於包含較多重複資料(字串)的數據,我們認為或許不需要用到 hash table 相對較為複雜的資料結構
- 只需要針對 Run-Length Encoding, 簡單地比較相鄰的字節,而不需要複雜的匹配條件。
- 利用相同字節的重複次數即為匹配的長度,不需要進行額外的計算。
- 不用像 deflate\_slow 一樣嘗試找到最長的匹配,並僅在確定在下一個窗口位置沒有更好匹配時,才使用當前的匹配
- 影響
  - 壓縮率:
    - 以上的改動可能會影響資料的壓縮效率
    - 以上方法應該比較適用於像是對於像素圖像中的單色區域,或者包含大量 重複字符的文本
  - 複雜度:
    - 上述方法的實現應該較為簡單易懂,deflate\_slow 有太多的條件判斷分支
- Pseudo code

```
1
     deflate_rle:
 2
       while not end_of_input:
 3
         fill_window(s)
         while s.lookahead >= MAX_MATCH:
 4
           run = find_consecutive_repeats(s)
 5
           if run >= MIN_MATCH:
 6
 7
             process_consecutive_repeats(s, run)
           else:
 8
 9
             process_single_literal(s)
           if flushing_needed:
10
             flush_block(s)
11
12
13
       flush_block(s, Z_FINISH)
       return Z_FINISH ? finish_done : block_done
14
15
```