组会报告

徐益

2018 年 9 月 20 日

1 工作内容

1. 数据采集测试；
2. 低时延 AVX2-LDPC 译码实现；

2 数据采集测试

2.1 ramdisk

2.1.1 传统 ramdisk

```
# mkdir /mnt/test
# mke2fs /dev/ram0
# mount /dev/ram /mnt/test
```

写入速率：700MB/s 800MB/s

2.1.2 ramfs

```
# mkdir /testRam
# mount -t ramfs none /testRAM
# mount -t ramfs none /testRAM -o maxsize=2000
```

写入速率：900MB/s 1100MB/s

2.1.3 tmpfs

```
# mkdir -p /mnt/tmpfs
# mount tmpfs /mnt/tmpfs -t tmpfs
# mount tmpfs /mnt/tmpfs -t tmpfs -o size=32g
```

写入速率：1.2GB/s 1.3GB/s

2.2 测试结果

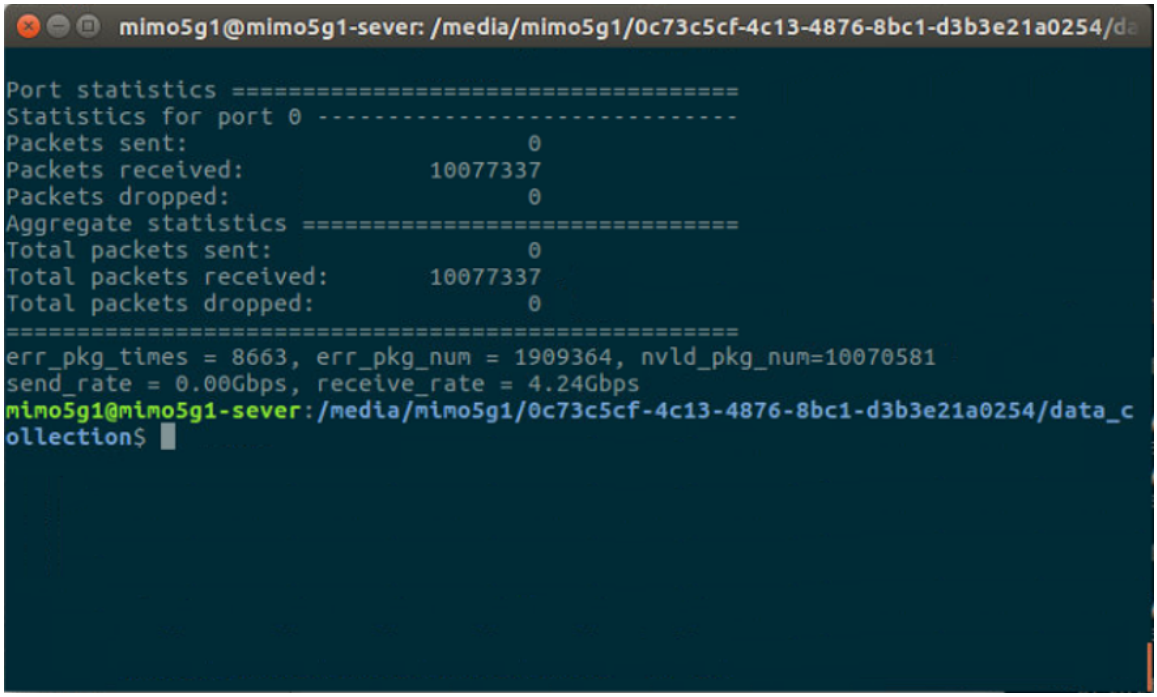


图 1: 向 ssd 写入

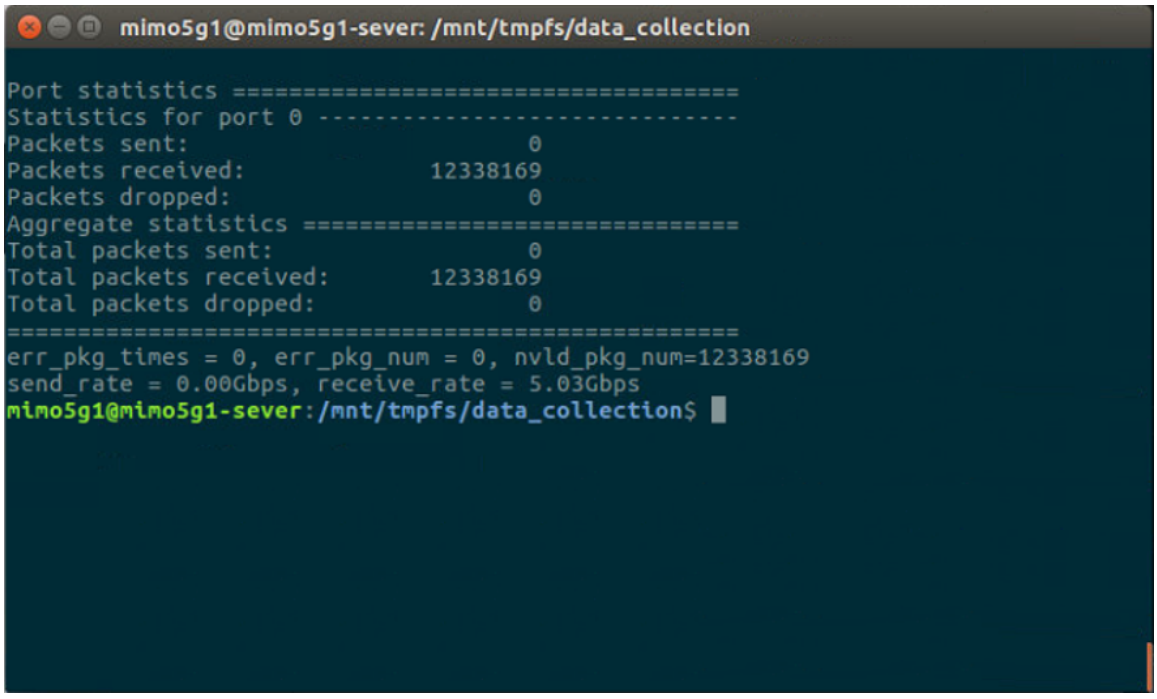


图 2: 向 tempfs 写入

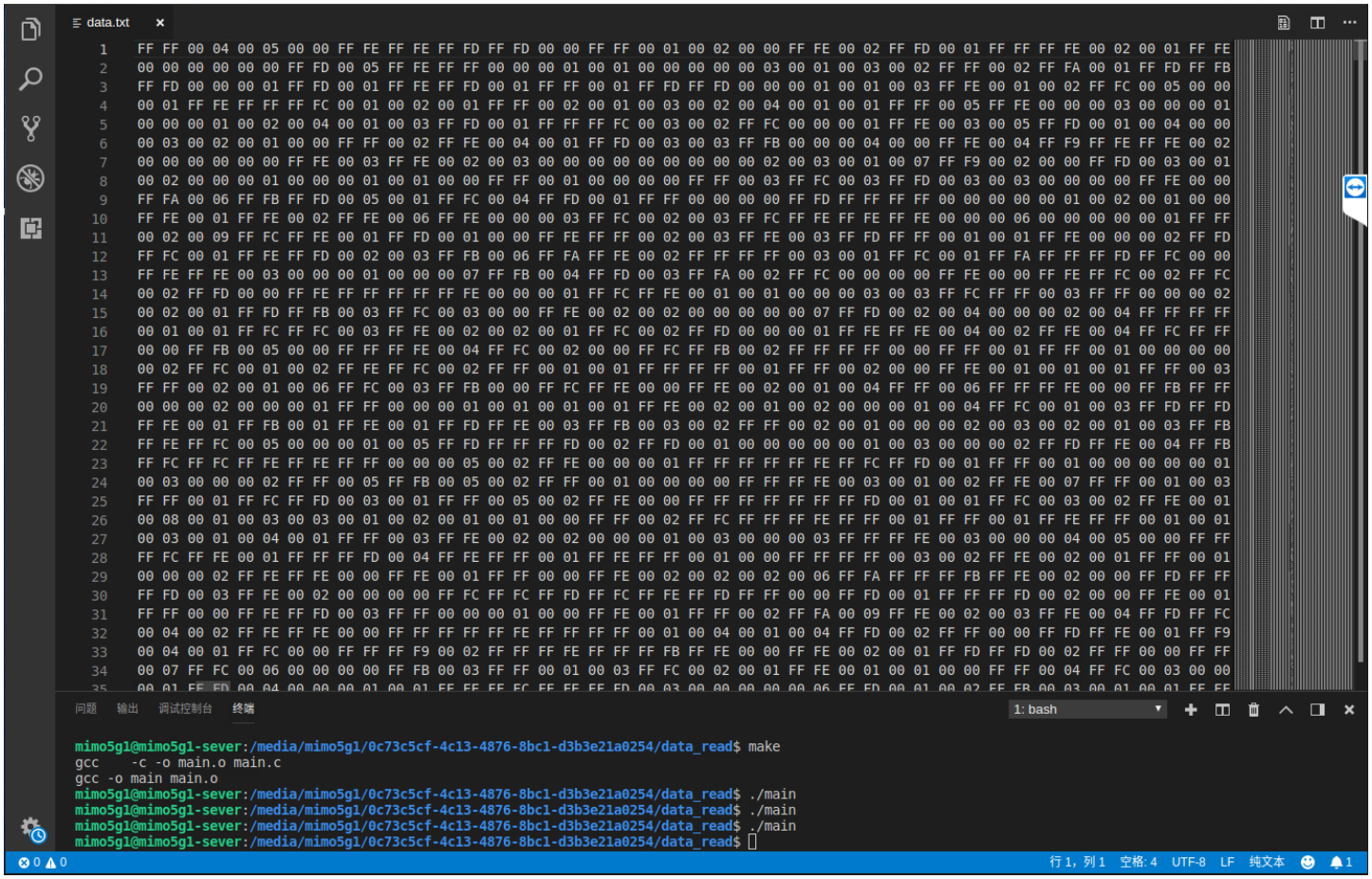

data.txt
FF FF 00 04 00 05 00 00 FF FE FF FE FF FD FF FD 00 00 FF FF 00 01 00 02 00 00 FF FE 00 02 FF FD 00 01 FF FF FF FE 00 02 00 01 FF FE
00 00 00 00 00 00 FF FD 00 05 FF FE FF FF 00 00 00 01 00 01 00 00 00 00 00 03 00 01 00 03 00 02 FF FF 00 02 FF FA 00 01 FF FD FF FB
FF FD 00 00 00 01 FF FD 00 01 FF FE FF FD 00 01 FF FF 00 01 FF FD FF FD 00 00 00 01 00 01 00 03 FF FE 00 01 00 02 FF FC 00 05 00 00
00 01 FF FE FF FF FF FC 00 01 00 02 00 01 FF FF 00 02 00 01 00 03 00 02 00 04 00 01 00 01 FF FF 00 05 FF FE 00 00 00 03 00 00 00 01
00 00 00 01 00 02 00 04 00 01 00 03 FF FD 00 01 FF FF FF FC 00 03 00 02 FF FC 00 00 00 01 FF FE 00 03 00 05 FF FD 00 01 00 04 00 00
00 03 00 02 00 01 00 00 FF FF 00 02 FF FE 00 04 00 01 FF FD 00 03 00 03 FF FB 00 00 00 04 00 00 FF FE 00 04 FF F9 FF FE FF FE 00 02
00 00 00 00 00 00 FF FE 00 03 FF FE 00 02 00 03 00 00 00 00 00 00 00 00 00 02 00 03 00 01 00 07 FF F9 00 02 00 00 FF FD 00 03 00 01
00 02 00 00 00 01 00 00 00 01 00 01 00 00 FF FF 00 01 00 00 00 00 FF FF 00 03 FF FC 00 03 FF FD 00 03 00 03 00 00 00 00 FF FE 00 00
FF FA 00 06 FF FB FF FD 00 05 00 01 FF FC 00 04 FF FD 00 01 FF FF 00 00 00 00 FF FD FF FF FF FF 00 00 00 00 00 01 00 02 00 01 00 00
FF FE 00 01 FF FE 00 02 FF FE 00 06 FF FE 00 00 00 03 FF FC 00 02 00 03 FF FC FF FE FF FE FF FE 00 00 00 06 00 00 00 00 00 01 FF FF
00 02 00 09 FF FC FF FE 00 01 FF FD 00 01 00 00 FF FE FF FF 00 02 00 03 FF FE 00 03 FF FD FF FF 00 01 00 01 FF FE 00 00 00 02 FF FD
FF FC 00 01 FF FE FF FD 00 02 00 03 FF FB 00 06 FF FA FF FE 00 02 FF FF FF FF 00 03 00 01 FF FC 00 01 FF FA FF FF FF FD FF FC 00 00
FF FE FF FE 00 03 00 00 00 01 00 00 00 07 FF FB 00 04 FF FD 00 03 FF FA 00 02 FF FC 00 00 00 00 FF FE 00 00 FF FE FF FC 00 02 FF FC
00 02 FF FD 00 00 FF FE FF FF FF FF FF FE 00 00 00 01 FF FC FF FE 00 01 00 01 00 00 00 03 00 03 FF FC FF FF 00 03 FF FF 00 00 00 02
00 02 00 01 FF FD FF FB 00 03 FF FC 00 03 00 00 FF FE 00 02 00 02 00 00 00 00 00 07 FF FD 00 02 00 04 00 00 00 02 00 04 FF FF FF FF
00 01 00 01 FF FC FF FC 00 03 FF FE 00 02 00 02 00 01 FF FC 00 02 FF FD 00 00 00 01 FF FE FF FE 00 04 00 02 FF FE 00 04 FF FC FF FF
00 00 FF FB 00 05 00 00 FF FF FF FE 00 04 FF FC 00 02 00 00 FF FC FF FB 00 02 FF FF FF FF 00 00 FF FF 00 01 FF FF 00 01 00 00 00 00
00 02 FF FC 00 01 00 02 FF FE FF FC 00 02 FF FF 00 01 00 01 FF FF FF FF 00 01 FF FF 00 02 00 00 FF FE 00 01 00 01 00 01 FF FF 00 03
FF FF 00 02 00 01 00 06 FF FC 00 03 FF FB 00 00 FF FC FF FE 00 00 FF FE 00 02 00 01 00 04 FF FF 00 06 FF FF FF FE 00 00 FF FB FF FF
00 00 00 02 00 00 00 01 FF FF 00 00 00 01 00 01 00 01 00 01 FF FE 00 02 00 01 00 02 00 00 00 01 00 04 FF FC 00 01 00 03 FF FD FF FD
FF FE 00 01 FF FB 00 01 FF FE 00 01 FF FD FF FE 00 03 FF FB 00 03 00 02 FF FF 00 02 00 01 00 00 00 02 00 03 00 02 00 01 00 03 FF FB
FF FE FF FC 00 05 00 00 00 01 00 05 FF FD FF FF FF FD 00 02 FF FD 00 01 00 00 00 00 00 01 00 03 00 00 00 02 FF FD FF FE 00 04 FF FB
FF FC FF FC FF FE FF FE FF FF 00 00 00 05 00 02 FF FE 00 00 00 01 FF FF FF FF FF FE FF FC FF FD 00 01 FF FF 00 01 00 00 00 00 00 01
00 03 00 00 00 02 FF FF 00 05 FF FB 00 05 00 02 FF FF 00 01 00 00 00 00 FF FF FF FE 00 03 00 01 00 02 FF FE 00 07 FF FF 00 01 00 03
FF FF 00 01 FF FC FF FD 00 03 00 01 FF FF 00 05 00 02 FF FE 00 00 FF FF FF FF FF FF FD 00 01 00 01 FF FC 00 03 00 02 FF FE 00 01
00 08 00 01 00 03 00 03 00 01 00 02 00 01 00 01 00 00 FF FF 00 02 FF FC FF FF FF FE FF FF 00 01 FF FF 00 01 FF FE FF FF 00 01 00 01
00 03 00 01 00 04 00 01 FF FF 00 03 FF FE 00 02 00 02 00 00 00 01 00 03 00 00 00 03 FF FF FF FE 00 03 00 00 00 04 00 05 00 00 FF FF
FF FC FF FE 00 01 FF FF FF FD 00 04 FF FE FF FF 00 01 FF FE FF FF 00 01 00 00 FF FF FF FF 00 03 00 02 FF FE 00 02 00 01 FF FF 00 01
00 00 00 02 FF FE FF FE 00 00 FF FE 00 01 FF FF 00 00 FF FE 00 02 00 02 00 06 FF FA FF FF FB FF FE 00 02 00 00 FF FD FF FF
FF FD 00 03 FF FE 00 02 00 00 00 00 FF FC FF FC FF FD FF FC FF FE FF FD FF FF 00 00 FF FD 00 01 FF FF FF FD 00 02 00 00 FF FE 00 01
FF FF 00 00 FF FE FF FD 00 03 FF FF 00 00 00 01 00 00 FF FE 00 01 FF FF 00 02 FF FA 00 09 FF FE 00 02 00 03 FF FE 00 04 FF FD FF FC
00 04 00 02 FF FE FF FE 00 00 FF FF FF FF FF FE FF FF FF FF 00 01 00 04 00 01 00 04 FF FD 00 02 FF FF 00 00 FF FD FF FE 00 01 FF F9
00 04 00 01 FF FC 00 00 FF FF FF F9 00 02 FF FF FF FE FF FF FF FB FF FE 00 00 FF FE 00 02 00 01 FF FD FF FD 00 02 FF FF 00 00 FF FF
00 07 FF FC 00 06 00 00 00 00 FF FB 00 03 FF FF 00 01 00 03 FF FC 00 02 00 01 FF FE 00 01 00 01 00 00 FF FF 00 04 FF FC 00 03 00 00
问题 输出 调试控制台 终端
1: bash
mimo5g1@mimo5g1-sever:/media/mimo5g1/0c73c5cf-4c13-4876-8bc1-d3b3e21a0254/data_read$ make
gcc -c -o main.o main.c
gcc -o main main.o
mimo5g1@mimo5g1-sever:/media/mimo5g1/0c73c5cf-4c13-4876-8bc1-d3b3e21a0254/data_read$ ./main
mimo5g1@mimo5g1-sever:/media/mimo5g1/0c73c5cf-4c13-4876-8bc1-d3b3e21a0254/data_read$ ./main
mimo5g1@mimo5g1-sever:/media/mimo5g1/0c73c5cf-4c13-4876-8bc1-d3b3e21a0254/data_read$ ./main
mimo5g1@mimo5g1-sever:/media/mimo5g1/0c73c5cf-4c13-4876-8bc1-d3b3e21a0254/data_read$
行 1，列 1 空格: 4 UTF-8 LF 纯文本

图 3: 转码结果

3 低时延 AVX2-LDPC 译码实现

3.1 设计系统时遇到的问题

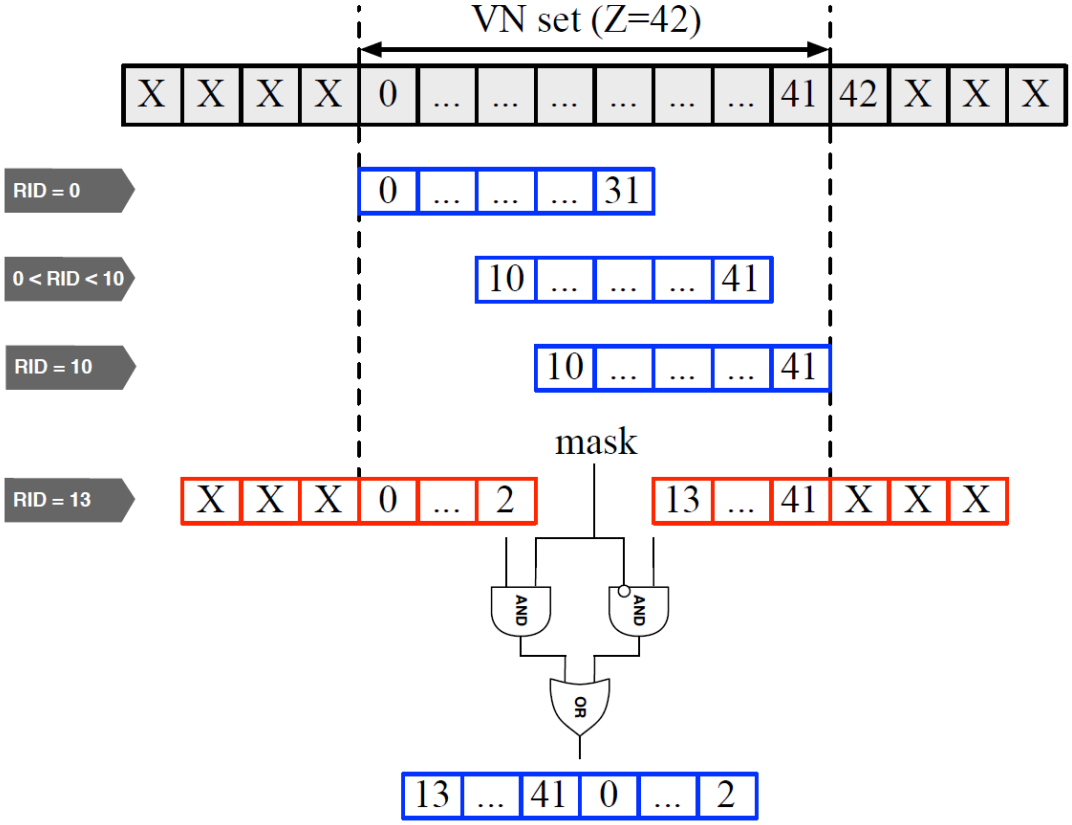


图 4: 论文提到的 VN 接入方式

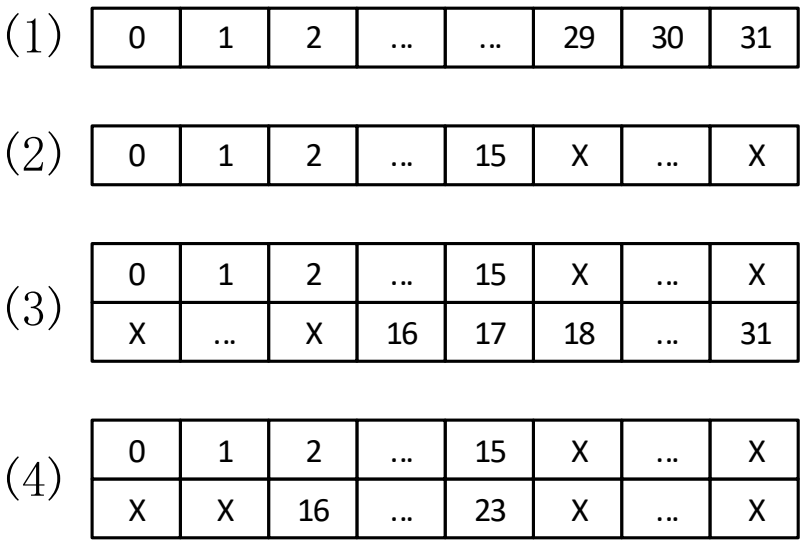


图 5: 实际所有的 VN 接入方式

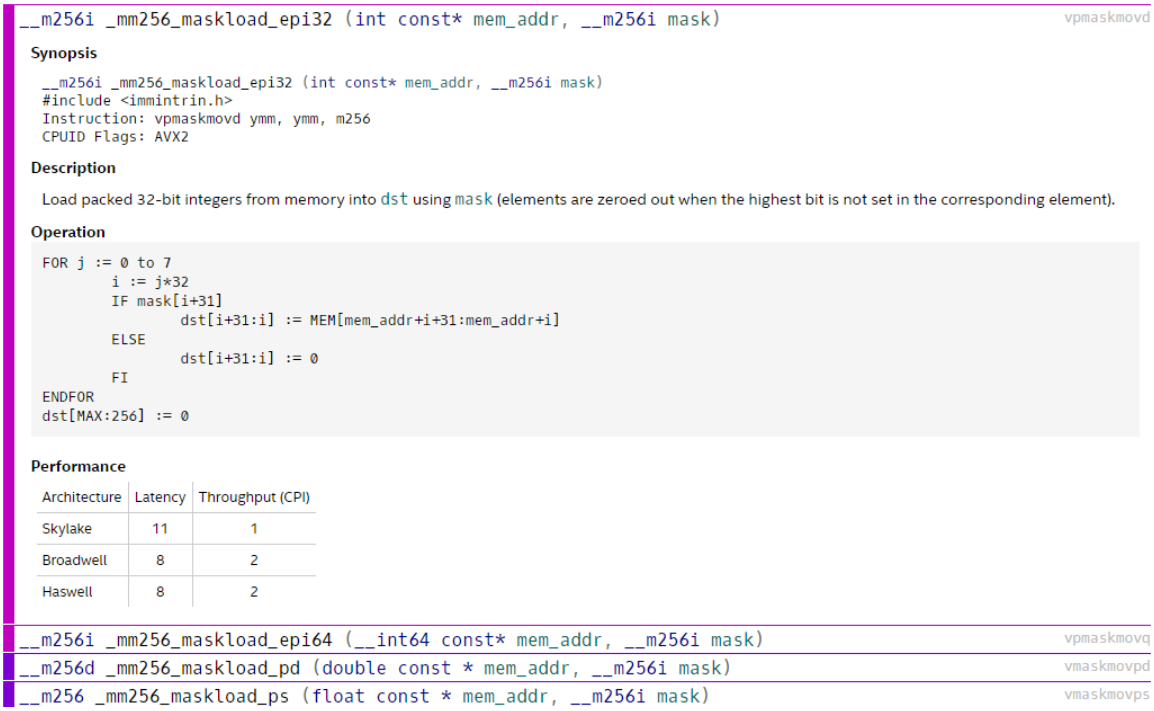
3.2 设计的数据结构

```c
typedef struct nr15_ldpc_simd_t
{
        …………

        /* Low-Latency Decoder Parameters */
        int32_t units;          // floor(Z_c/REG_SIZE)
        int32_t whole_degree;   // sum of degree of every hbg_row_d
        int8_t* degree;         // number of connective check nodes
                                // (length:hbg_row_d)
        __m256i* cn_msg_avx2;   // avx2 message from cn to vn
                                // (length:whole_degree*units)
        __m256i* vn_msg_avx2;   // temp avx2 message from vn to cn(length:19)
        int8_t* llr_fixed;      // fixed llr info(length:2*REG_SIZE+Nd)
        int8_t* llr_addr_flag;  // flag for access type
                                // (0:no mask;1:1 mask;2:2 mask)
                                // (length:whole_degree*units)
        int8_t** llr_addr;      // llr address(length:whole_degree*units)
        int8_t** llr_addr_pre;  // extra llr address for flag=2
                                //(length:whole_degree)
        __m256i* mask;          // mask1 for flag=2(length:whole_degree)
        __m256i* mask_pre;      // mask2 for flag=2(length:whole_degree)
        __m256i endmask;        // mask for flag=1

} nr15_ldpc_simd_t;
```

3.3 使用掩码时遇到的问题



__m256i _mm256_maskload_epi32 (int const* mem_addr, __m256i mask) vpmaskmovd

Synopsis

```
__m256i _mm256_maskload_epi32 (int const* mem_addr, __m256i mask)
#include <immintrin.h>
Instruction: vpmaskmovd ymm, ymm, m256
CPUID Flags: AVX2
```

Description

Load packed 32-bit integers from memory into dst using mask (elements are zeroed out when the highest bit is not set in the corresponding element).

Operation

```
FOR j := 0 to 7
        i := j*32
        IF mask[i+31]
                dst[i+31:i] := MEM[mem_addr+i+31:mem_addr+i]
        ELSE
                dst[i+31:i] := 0
        FI
ENDFOR
dst[MAX:256] := 0
```

Performance

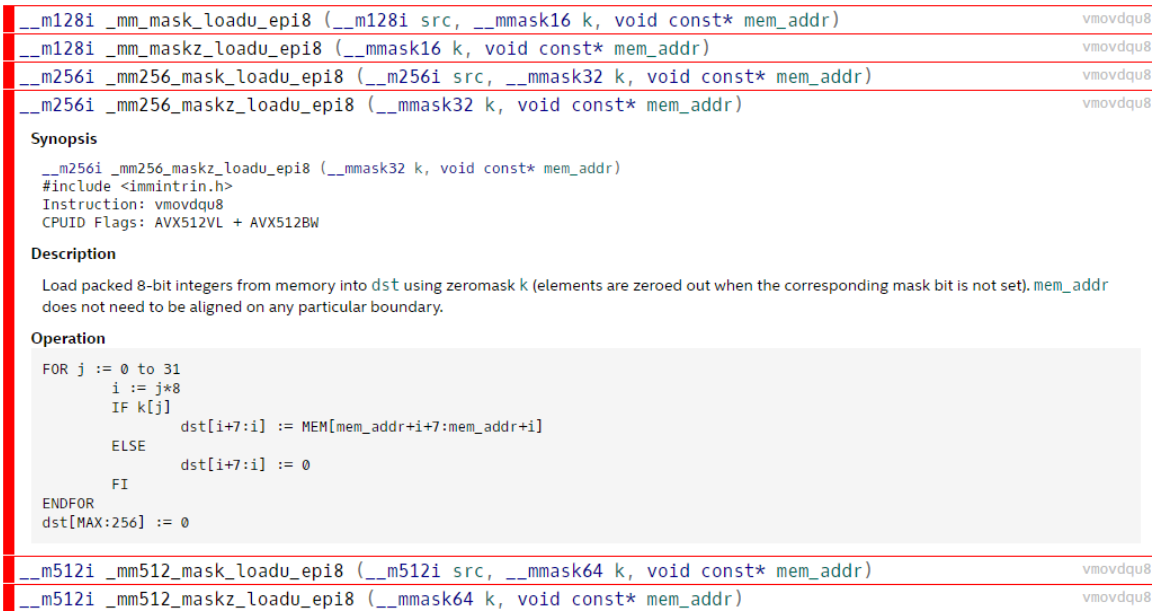
Architecture	Latency	Throughput (CPI)
Skylake	11	1
Broadwell	8	2
Haswell	8	2

__m256i _mm256_maskload_epi64 (__int64 const* mem_addr, __m256i mask) vpmaskmovq

__m256d _mm256_maskload_pd (double const * mem_addr, __m256i mask) vmaskmovpd

__m256 _mm256_maskload_ps (float const * mem_addr, __m256i mask) vmaskmovps

图 6: AVX2 mask load 相关函数



```
__m128i _mm_mask_loadu_epi8 (__m128i src, __mmask16 k, void const* mem_addr)        vmovdqu8
__m128i _mm_maskz_loadu_epi8 (__mmask16 k, void const* mem_addr)                    vmovdqu8
__m256i _mm256_mask_loadu_epi8 (__m256i src, __mmask32 k, void const* mem_addr)     vmovdqu8
__m256i _mm256_maskz_loadu_epi8 (__mmask32 k, void const* mem_addr)                 vmovdqu8
```

Synopsis

```
__m256i _mm256_maskz_loadu_epi8 (__mmask32 k, void const* mem_addr)
#include <immintrin.h>
Instruction: vmovdqu8
CPUID Flags: AVX512VL + AVX512BW
```

Description

Load packed 8-bit integers from memory into dst using zeromask k (elements are zeroed out when the corresponding mask bit is not set). mem_addr does not need to be aligned on any particular boundary.

Operation

```
FOR j := 0 to 31
        i := j*8
        IF k[j]
                dst[i+7:i] := MEM[mem_addr+i+7:mem_addr+i]
        ELSE
                dst[i+7:i] := 0
        FI
ENDFOR
dst[MAX:256] := 0
```

```
__m512i _mm512_mask_loadu_epi8 (__m512i src, __mmask64 k, void const* mem_addr)     vmovdqu8
__m512i _mm512_maskz_loadu_epi8 (__mmask64 k, void const* mem_addr)                 vmovdqu8
```

图 7: AVX-512 mask load 相关函数

3.4 当前掩码处理方式

3.4.1 取数据加掩码方式

```
if (*pflag1 == 2)
{
        vllr1 = VECTOR_LOAD(*p_indice_nod1);
        vllr2 = VECTOR_LOAD(*p_indice_nod_pre1);
        mask1 = VECTOR_LOAD(p_mask1);
        mask2 = VECTOR_LOAD(p_maskpre1);
        vllrm1 = VECTOR_AND(vllr1, mask1);
        vllrm2 = VECTOR_AND(vllr2, mask2);
        vllr = VECTOR_OR(vllrm1, vllrm2);

        p_mask1++;
        p_maskpre1++;
        p_indice_nod_pre1++;
}
else
        vllr = VECTOR_LOAD(*p_indice_nod1);
```

3.4.2 存数据加掩码方式

```
if (*pflag2 == 2)
{
        vllr1 = VECTOR_LOAD(*p_indice_nod2);
        vllr2 = VECTOR_LOAD(*p_indice_nod_pre2);
        mask1 = VECTOR_LOAD(p_mask2);
        mask2 = VECTOR_LOAD(p_maskpre2);
        vllrm1 = VECTOR_AND(mask1, v2llr);
        vllrm2 = VECTOR_AND(mask2, v2llr);
```

```
        vllro1 = VECTOR_ANDNOT(mask1, vllr1);
        vllro2 = VECTOR_ANDNOT(mask2, vllr2);
        vllr1 = VECTOR_OR(vllrm1, vllro1);
        vllr2 = VECTOR_OR(vllrm2, vllro2);
        VECTOR_STORE(*p_indice_nod2, vllr1);
        VECTOR_STORE(*p_indice_nod_pre2, vllr2);

        p_mask2++;
        p_maskpre2++;
        p_indice_nod_pre2++;
}
else if (*pflag2 == 1)
{
        vllrm1 = VECTOR_AND(endmask, v2llr);
        vllr = VECTOR_LOAD(*p_indice_nod2);
        vllro1 = VECTOR_ANDNOT(endmask, vllr);
        vllr = VECTOR_OR(vllrm1, vllro1);
        VECTOR_STORE(*p_indice_nod2, vllr);
}
else
        VECTOR_STORE(*p_indice_nod2, v2llr);
```

3.5 系统测试

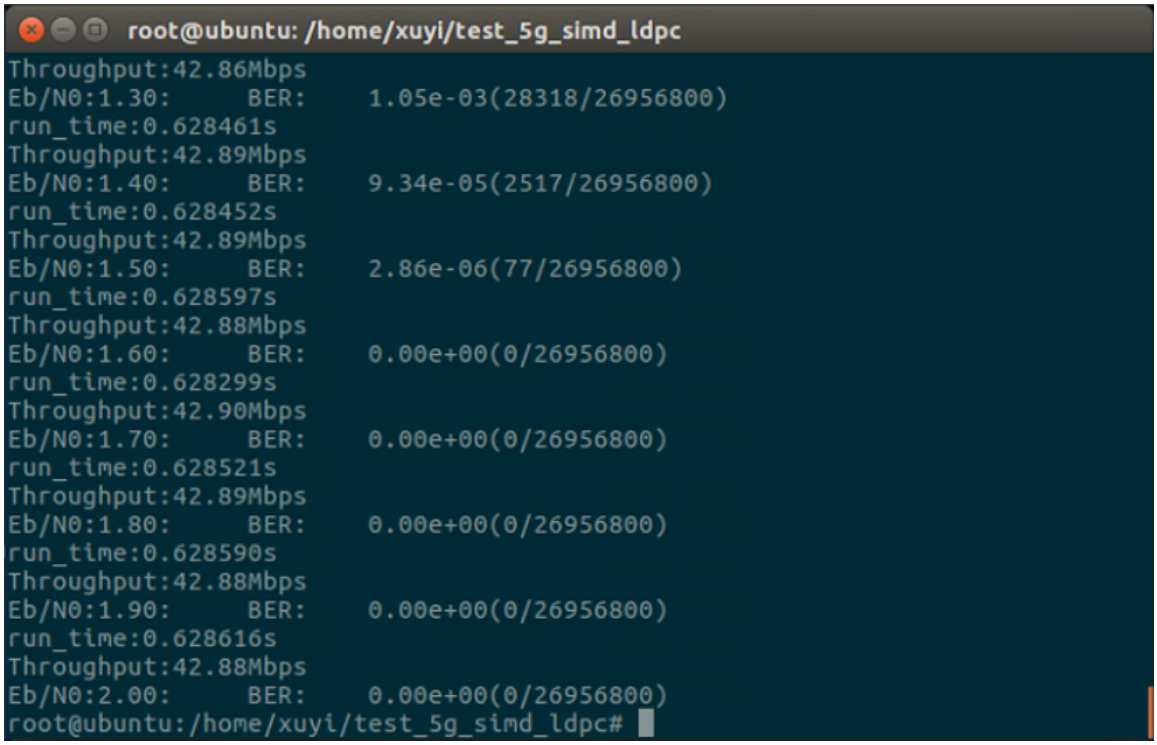


图 8: 运行结果

表 1: High-Thoughput 和 Low-Latency 系统性能对比 (Intel® Xeon® CPU E7-8867 V4 @2.40GHz)

scheduling	High-Thoughput	Low-Latency
Throughput	59.19Mbps	43.02Mbps
Latency(K=8448)	4.5543ms	0.1958ms

4 改写仿真报告