uProcessor Term Project: FFT & MVM optimization

10조: 이승훈, 박진호

건국대학교

전기전자공학부

MVM optimization result

```
Measured Accuracy: NSR(dB) = -inf

-----Benchmarking Start-----
Case 0: MVM Reference
Nr,         Max,         Min,     Average,    Fltr Avg,  Fltr_Avg(ms)
10,    66955963,    66934245,    66942855,    66942293,       200.827
Case 1: MVM Optimization
Nr,         Max,         Min,     Average,    Fltr Avg,  Fltr_Avg(ms)
10,    28289387,    28271313,    28279085,    28278769,        84.836
----Benchmarking Complete----

Optimized MVM is x2.37 faster than Reference
```

- What's applied?
  - Loop interchange
  - Matrix tiling

Loop optimization: Loop interchange

```
1  // original code
2  for (i = 0; i < NSC; i++) // i: 0 -> 2048
3    for (n = 0; n < NOFDM; n++) // n: 0 -> 10
4      for (j = 0; j < NTX; j++) // j: 0 -> 8
5        for (k = 0; k < NRX; k++) // k: 0 -> 8
6
```

<Original code>

*0	*1	*8	*64	*2048	*16384
+0	+1	+2	+2	+3	+3

Table: Column access weights

```
1  // loop interchanged
2  for (n = 0; n < NOFDM; n++) // n: 0 -> 10
3    for (i = 0; i < NSC; i++) // i: 0 -> 2048
4      for (k = 0; k < NRX; k++) // k: 0 -> 8
5        for (j = 0; j < NTX; j++) // j: 0 -> 8
6  {
7  tmp1 = n*16384 +                  i*8 + j;
8  tmp2 =                  i*64 + j*8 + k;
9  tmp3 = n*16384 + k*2048             + i;
10 out[tmp1].real +=
11           h_inv[tmp2].real * in[tmp3].real -
12           h_inv[tmp2].img  * in[tmp3].img;
13 out[tmp1].img  +=
14             h_inv[tmp2].img  * in[tmp3].real+
15             h_inv[tmp2].real * in[tmp3].img;
16 }
```

<Loop interchanged code>

-	n	i	j	k
tmp1	3	2	1	0
tmp2	0	2	2	1
tmp3	3	1	0	3
total	6	5	3	4

Table: Column Access cost for each variables

Loop optimization: Tiling

```
1  // loop interchanged
2  for (n = 0; n < NOFDM; n++) // n: 0 -> 10
3    for (i = 0; i < NSC; i++) // i: 0 -> 2048
4      for (k = 0; k < NRX; k++) // k: 0 -> 8
5        for (j = 0; j < NTX; j++) // j: 0 -> 8
6
```

<Loop interchanged code>

*0	*1	*8	*64	*2048	*16384
+0	+1	+2	+2	+3	+3

Table: Column access weights

```
1  // tiling applied
2  int TS = 4;
3  int kk, jj;
4  for (n = 0; n < NOFDM; n++) // n: 0 -> 10
5    for (i = 0; i < NSC; i++) // i: 0 -> 2048
6      for (k = 0; k < NRX/TS; k++ ) // k: 0 ->
         2
7      {
8        for (j = 0; j < NTX/TS; j++) // j: 0
        -> 2
9        {
10         for(kk = k*TS;kk<k*TS+TS;kk++) // kk
         : 0 -> 4, 4->8
11         {
12           for(jj=j*TS;jj<j*TS+TS;jj++)
13           {
14 ...
15 }}}}
```

<Matrix tiling applied code>

-	n	i	j	k
tmp1	3	2(3)	1(2)	0
tmp2	0	2(3)	2(3)	1(2)
tmp3	3	1(2)	0	3
total	6	5(8)	3(5)	4(5)

Table: Column Access cost for each variables