

1. (p8) T=2,4,8,16

<p>T=2</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 6227894791, 6227579718, 6227682534, 6227668854, 18683005.342 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 3916693952, 3914432447, 3916181816, 3916336471, 11749008.648 ----Benchmarking Complete----</pre>	<p>T=4</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 6227694862, 6227497750, 6227598011, 6227595938, 18682786.594 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 2862475052, 2861960862, 2862079611, 2862045025, 8586134.514 ----Benchmarking Complete----</pre>
<p>T=8</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 6226719075, 6226393916, 6226575627, 6226580410, 18679740.010 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 2452099313, 2451791037, 2451938545, 2451936887, 7355810.181 ----Benchmarking Complete----</pre>	<p>T=16</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 6228228453, 6227985019, 6228109259, 6228109890, 18684328.450 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr_Avg(us) 10, 2462132291, 2462014393, 2462083079, 2462085514, 7386256.060 ----Benchmarking Complete----</pre>

○속도변화 이유

Tiling을 하지 않은 Case 0의 경우 실행 시 보드의 캐쉬의 상태에 따라 조금의 차이가 있지만 대부분 비슷한 수준의 속도를 보였다. 이는 기존의 일반적인 matrix의 내적계산법을 통해 진행하였으므로, 32KB 한 라인당 8개 word를 가지는 L1 Cache에서 b에 한 colum에 해당하는 숫자를 가져올 때 계속해서 miss가 나게 된다. 따라서, L1 Cache에 없는 데이터를 가져오기 위해 L2 Cache, DDR까지 접근하여 가져와야 하므로 이에 따라 많은 시간이 더 걸리게 된다.

하지만, Tiling을 한 Case 1의 경우 비슷한 지역의 숫자들을 묶어두었기 때문에 한 번 계산 시 Cache위에 올려둔 숫자들을 최대한 miss가 적게 나게 사용할 수 있다. T가 증가할수록 실행시간이 줄어들었다가 T가 8일 때 가장 좋은 실행시간을 나타낸다. 이는, L1 Cache의 캐시라인 하나 당 8개의 word가 올라가는 특성을 모두 활용했기 때문이다. Tiling을 활용한 계산에서 tiling된 ra의 연속된 숫자 64개를 각 8개씩 8line에 올려두고 똑같이 rb의 연속된 숫자 64개를 8개씩 8line에 올려 두면 해당 데이터만으로 Tiling 계산을 위한 한 iteration시 활용되는 모든 데이터가 L1 Cache위에 올려져 있으므로, iteration이 바뀔때만 Cache에 miss가 나게 되어 가장 효율이 좋은 결과를 얻을 수 있게 된다. 만일 T가 16으로 증가한다면, ra와 rb의 계산 중에, L1 Cache에 다 담을 수 없는 범위의 데이터와 계산이 들어가게 되므로 그 과정에서 조금의 시간이 더 걸리게 된다.

## 2. p9.(Optimization O2)

(1) T = 2

<pre> 90      unsigned mat_mult_tiling(unsigned int uiParam0, unsigned int 001008fc:  mov     r0, #1 00100900:  pop     {r4,r5,r6,r7,r8,pc} 00100904:  .byte 0x1c 00100905:  .byte 0x4b 00100906:  .byte 0x21 00100907:  .byte 0x00 00100908:  .byte 0x1c 00100909:  .byte 0x4b 0010090a:  .byte 0x41 0010090b:  .byte 0x00 91      { 92          int io,jo,ko,ii,ki,ji; 93          int *rresult, *rb, *ra; 94 mat_mult_tiling: 0010090c:  ldr     r3, [pc, #+208] 00100910:  push    {r4,r5,r6,r7,r8,r9,r10,r11,lr} 00100914:  sub     sp, sp, #20 00100918:  str     r3, [sp, #+12] 0010091c:  ldr     r3, [pc, #+196] 00100920:  str     r3, [sp, #+8] 00100924:  ldr     r10, [pc, #+192] 102          rresult[ji] += ra[ki] * rb[ji]; 103 104      return 1; 00100928:  ldr     r9, [sp, #+8] 91      { 92          int io,jo,ko,ii,ki,ji; 93          int *rresult, *rb, *ra; 94 0010092c:  ldr     r4, [sp, #+12] 00100930:  sub     lr, r10, #1048576 00100934:  sub     r11, r9, #4096 102          rresult[ji] += ra[ki] * rb[ji]; 103 104      return 1; 00100938:  ldr     r8, [lr] 0010093c:  mov     r3, r11 00100940:  ldr     r7, [lr, #+4] 00100944:  mov     r12, r4 00100948:  ldr     r6, [lr, #+2048] </pre>	<pre> 0010094c:  ldr     r5, [lr, #+2052] 00100950:  str     lr, [sp, #+4] 00100954:  ldr     r2, [r3, #-2044] 00100958:  ldr     r0, [r12, #-2048] 0010095c:  add     r12, r12, #2048 00100960:  ldr     lr, [r3, #-2048] 00100964:  ldr     r1, [r12, #-4092] 00100968:  mla     r2, r0, r7, r2 0010096c:  mla     r0, r0, r8, lr 00100970:  mla     r2, r5, r1, r2 00100974:  mla     r0, r6, r1, r0 00100978:  str     r2, [r3, #-2044] 0010097c:  str     r0, [r3, #-2048] 00100980:  add     r3, r3, #2048 98          for(ii = 0, rresult = &amp;result2[i0][j0], ra = &amp;a[i0][k0]; 99              ii &lt; T; ii++, rresult +=N, ra += N) 100              for(ki = 0, rb = &amp;b[k0][j0]; ki &lt; T; ki++, rb += N) 101                  for(ji = 0; ji &lt; T; ji++) 00100984:  cmp     r9, r3 00100988:  bne     -60 ; addr=0x00100954: mat_mult_tiling + 0x00000048 0010098c:  ldr     lr, [sp, #+4] 00100990:  add     r4, r4, #8 00100994:  add     lr, lr, #4096 97          for(ko = 0; ko &lt; N; ko += T) 00100998:  cmp     r10, lr 0010099c:  bne     -108 ; addr=0x00100938: mat_mult_tiling + 0x0000002c 96          for(jo = 0; jo &lt; N; jo += T) 001009a0:  ldr     r3, [pc, #+72] 001009a4:  add     r10, r10, #8 001009a8:  add     r9, r9, #8 001009ac:  cmp     r3, r10 001009b0:  bne     -140 ; addr=0x0010092c: mat_mult_tiling + 0x00000020 001009b4:  ldr     r2, [sp, #+8] 001009b8:  ldr     r3, [sp, #+12] 001009bc:  add     r2, r2, #4096 001009c0:  str     r2, [sp, #+8] 001009c4:  add     r3, r3, #4096 95          for(io = 0; io &lt; N; io += T) 001009c8:  ldr     r2, [pc, #+36] 001009cc:  str     r3, [sp, #+12] 001009d0:  cmp     r2, r3 001009d4:  bne     -184 ; addr=0x00100924: mat_mult_tiling + 0x00000018 </pre>
--	--

(2) T = 4

<pre> 90      unsigned mat_mult_tiling(unsigned int uiParam0, unsigned int 001008fc:  mov     r0, #1 00100900:  pop     {r4,r5,r6,r7,r8,pc} 00100904:  .byte 0x1c 00100905:  .byte 0x4b 00100906:  .byte 0x21 00100907:  .byte 0x00 00100908:  .byte 0x1c 00100909:  .byte 0x4b 0010090a:  .byte 0x41 0010090b:  .byte 0x00 91      { 92          int io,jo,ko,ii,ki,ji; 93          int *rresult, *rb, *ra; 94 mat_mult_tiling: 0010090c:  ldr     r3, [pc, #+216] 00100910:  push    {r4,r5,r6,r7,r8,r9,r10,r11,lr} 00100914:  sub     sp, sp, #20 00100918:  str     r3, [sp, #+12] 0010091c:  ldr     r3, [pc, #+204] 00100920:  str     r3, [sp, #+8] 00100924:  ldr     r11, [pc, #+200] 00100928:  ldr     r3, [sp, #+8] 0010092c:  str     r3, [sp, #+4] 00100930:  ldr     r10, [sp, #+12] 00100934:  sub     r8, r11, #1048576 00100938:  ldr     r7, [sp, #+4] 0010093c:  add     r9, r10, #8192 00100940:  mov     r6, r10 00100944:  sub     r4, r6, #16 00100948:  mov     lr, r8 0010094c:  add     r12, r7, #16 00100950:  ldr     r0, [r4, #+4]! 00100954:  mov     r1, lr 00100958:  mov     r2, r7 102          rresult[ji] += ra[ki] * rb[ji]; 103 104      return 1; 0010095c:  ldr     r5, [r2, #+4] 00100960:  ldr     r3, [r1, #+4]! 00100964:  mla     r3, r3, r0, r5 </pre>	<pre> 00100968:  str     r3, [r2, #+4]! 101          for(ji = 0; ji &lt; T; ji++) 0010096c:  cmp     r12, r2 00100970:  bne     -28 ; addr=0x0010095c: mat_mult_tiling + 0x00000050 100          for(ki = 0, rb = &amp;b[k0][j0]; ki &lt; T; ki++, rb += N) 00100974:  cmp     r6, r4 00100978:  add     lr, lr, #2048 0010097c:  bne     -52 ; addr=0x00100950: mat_mult_tiling + 0x00000044 00100980:  add     r6, r6, #2048 00100984:  add     r7, r7, #2048 98          for(ii = 0, rresult = &amp;result2[i0][j0], ra = &amp;a[i0][k0]; 99              ii &lt; T; ii++, rresult +=N, ra += N) 00100988:  cmp     r9, r6 0010098c:  bne     -80 ; addr=0x00100944: mat_mult_tiling + 0x00000038 00100990:  add     r8, r8, #8192 00100994:  add     r10, r10, #16 97          for(ko = 0; ko &lt; N; ko += T) 00100998:  cmp     r11, r8 0010099c:  bne     -108 ; addr=0x00100938: mat_mult_tiling + 0x0000002c 001009a0:  ldr     r3, [sp, #+4] 001009a4:  add     r11, r11, #16 001009a8:  add     r3, r3, #16 001009ac:  str     r3, [sp, #+4] 96          for(jo = 0; jo &lt; N; jo += T) 001009b0:  ldr     r3, [pc, #+64] 001009b4:  cmp     r3, r11 001009b8:  bne     -144 ; addr=0x00100930: mat_mult_tiling + 0x00000024 001009bc:  ldr     r2, [sp, #+8] 001009c0:  ldr     r3, [sp, #+12] 001009c4:  add     r2, r2, #8192 001009c8:  str     r2, [sp, #+8] 001009cc:  add     r3, r3, #8192 95          for(io = 0; io &lt; N; io += T) 001009d0:  ldr     r2, [pc, #+36] 001009d4:  str     r3, [sp, #+12] 001009d8:  cmp     r2, r3 001009dc:  bne     -192 ; addr=0x00100924: mat_mult_tiling + 0x00000018 105      } </pre>
---	--

(3) T = 8

<pre> 90      unsigned mat_mult_tiling(unsigned int uiParam0, unsigned int 001008fc:  mov     r0, #1 00100900:  pop     {r4,r5,r6,r7,r8,pc} 00100904:  .byte 0x1c 00100905:  .byte 0x4b 00100906:  .byte 0x21 00100907:  .byte 0x00 00100908:  .byte 0x1c 00100909:  .byte 0x4b 0010090a:  .byte 0x41 0010090b:  .byte 0x00 91      { 92      int io,jo,ko,ii,ki,ji; 93      int *rresult, *rb, *ra; 94 mat_mult_tiling: 0010090c:  push    {r4,r5,r6,r7,r8,r9,r10,r11,lr} 00100910:  sub     sp, sp, #20 00100914:  ldr     r11, [pc, #+204] 00100918:  ldr     r3, [pc, #+204] 0010091c:  str     r3, [sp, #+4] 00100920:  ldr     r10, [pc, #+200] 00100924:  ldr     r3, [sp, #+4] 00100928:  str     r11, [sp, #+12] 0010092c:  str     r3, [sp, #+8] 00100930:  ldr     r11, [sp, #+12] 00100934:  sub     r8, r10, #1048576 00100938:  ldr     r6, [sp, #+8] 0010093c:  add     r7, r11, #16384 00100940:  mov     r5, r11 00100944:  sub     r4, r5, #32 00100948:  mov     lr, r8 0010094c:  add     r12, r6, #32 00100950:  ldr     r0, [r4, #+4]! 00100954:  mov     r1, lr 00100958:  mov     r2, r6 102             rresult[ji] += ra[ki] * rb[ji]; 103 104      return 1; 0010095c:  ldr     r9, [r2, #+4] 00100960:  ldr     r3, [r1, #+4]! 00100964:  mla     r3, r3, r0, r9 </pre>	<pre> 00100968:  str     r3, [r2, #+4]! 101             for(ji = 0; ji &lt; T; ji++) 0010096c:  cmp     r12, r2 00100970:  bne     -28 ; addr=0x0010095c: mat_mult_tiling + 0x00000050 100             for(ki = 0, rb = &amp;b[k0][j0]; ki &lt; T; ki++, rb += N) 00100974:  cmp     r5, r4 00100978:  add     lr, lr, #2048 0010097c:  bne     -52 ; addr=0x00100950: mat_mult_tiling + 0x00000044 00100980:  add     r5, r5, #2048 00100984:  add     r6, r6, #2048 98             for(ii = 0, rresult = &amp;result2[i0][j0], ra = &amp;a[i0][k0]; 99             ii &lt; T; ii++, rresult +=N, ra += N) 00100988:  cmp     r7, r5 0010098c:  bne     -80 ; addr=0x00100944: mat_mult_tiling + 0x00000038 00100990:  add     r8, r8, #16384 00100994:  add     r11, r11, #32 97             for(ko = 0; ko &lt; N; ko += T) 00100998:  cmp     r10, r8 0010099c:  bne     -108 ; addr=0x00100938: mat_mult_tiling + 0x0000002c 001009a0:  ldr     r3, [sp, #+8] 001009a4:  add     r10, r10, #32 001009a8:  add     r3, r3, #32 001009ac:  str     r3, [sp, #+8] 96             for(jo = 0; jo &lt; N; jo += T) 001009b0:  ldr     r3, [pc, #+60] 001009b4:  cmp     r3, r10 001009b8:  bne     -144 ; addr=0x00100930: mat_mult_tiling + 0x00000024 001009bc:  ldr     r3, [sp, #+4] 001009c0:  ldr     r11, [sp, #+12] 001009c4:  add     r3, r3, #16384 001009c8:  str     r3, [sp, #+4] 001009cc:  add     r11, r11, #16384 95             for(io = 0; io &lt; N; io += T) 001009d0:  ldr     r3, [pc, #+32] 001009d4:  cmp     r3, r11 001009d8:  bne     -192 ; addr=0x00100920: mat_mult_tiling + 0x00000014 105      } </pre>
---	--

(4) T = 16

<pre> 90      unsigned mat_mult_tiling(unsigned int uiP 001008fc:  mov     r0, #1 00100900:  pop     {r4,r5,r6,r7,r8,pc} 00100904:  .byte 0x1c 00100905:  .byte 0x4b 00100906:  .byte 0x21 00100907:  .byte 0x00 00100908:  .byte 0x1c 00100909:  .byte 0x4b 0010090a:  .byte 0x41 0010090b:  .byte 0x00 91      { 92      int io,jo,ko,ii,ki,ji; 93      int *rresult, *rb, *ra; 94 mat_mult_tiling: 0010090c:  push    {r4,r5,r6,r7,r8,r9,r10,r11,lr} 00100910:  sub     sp, sp, #20 00100914:  ldr     r11, [pc, #+204] 00100918:  ldr     r3, [pc, #+204] 0010091c:  str     r3, [sp, #+4] 00100920:  ldr     r10, [pc, #+200] 00100924:  ldr     r3, [sp, #+4] 00100928:  str     r11, [sp, #+12] 0010092c:  str     r3, [sp, #+8] 00100930:  ldr     r9, [sp, #+12] 00100934:  sub     r8, r10, #1048576 00100938:  ldr     r6, [sp, #+8] 0010093c:  add     r7, r9, #32768 00100940:  mov     r5, r9 00100944:  sub     r4, r5, #64 00100948:  mov     lr, r8 0010094c:  add     r12, r6, #64 00100950:  ldr     r0, [r4, #+4]! 00100954:  mov     r1, lr 00100958:  mov     r2, r6 102             rresult[ji] + 103 104      return 1; 0010095c:  ldr     r11, [r2, #+4] 00100960:  ldr     r3, [r1, #+4]! 00100964:  mla     r3, r3, r0, r11 </pre>	<pre> 00100968:  str     r3, [r2, #+4]! 101             for(ji = 0; ji &lt; T; ji++) 0010096c:  cmp     r12, r2 00100970:  bne     -28 ; addr=0x0010095c: mat_mult_tiling + 0x00000050 100             for(ki = 0, rb = &amp;b[k0][j0]; ki &lt; T; ki++, rb + 00100974:  cmp     r5, r4 00100978:  add     lr, lr, #2048 0010097c:  bne     -52 ; addr=0x00100950: mat_mult_tiling + 0x00000044 00100980:  add     r5, r5, #2048 00100984:  add     r6, r6, #2048 98             for(ii = 0, rresult = &amp;result2[i0][j0], ra = &amp;a[i0] 99             ii &lt; T; ii++, rresult +=N, ra += N) 00100988:  cmp     r7, r5 0010098c:  bne     -80 ; addr=0x00100944: mat_mult_tiling + 0x00000038 00100990:  add     r8, r8, #32768 00100994:  add     r9, r9, #64 97             for(ko = 0; ko &lt; N; ko += T) 00100998:  cmp     r10, r8 0010099c:  bne     -108 ; addr=0x00100938: mat_mult_tiling + 0x0000002c 001009a0:  ldr     r3, [sp, #+8] 001009a4:  add     r10, r10, #64 001009a8:  add     r3, r3, #64 001009ac:  str     r3, [sp, #+8] 96             for(jo = 0; jo &lt; N; jo += T) 001009b0:  ldr     r3, [pc, #+60] 001009b4:  cmp     r3, r10 001009b8:  bne     -144 ; addr=0x00100930: mat_mult_tiling + 0x00000024 001009bc:  ldr     r3, [sp, #+4] 001009c0:  ldr     r11, [sp, #+12] 001009c4:  add     r3, r3, #32768 001009c8:  str     r3, [sp, #+4] 001009cc:  add     r11, r11, #32768 95             for(io = 0; io &lt; N; io += T) 001009d0:  ldr     r3, [pc, #+32] 001009d4:  cmp     r3, r11 001009d8:  bne     -192 ; addr=0x00100920: mat_mult_tiling + 0x00000014 105      } </pre>
---	---

### (5) 결과 분석

최적화 옵션인 O2는 메모리공간과 속도에 대한 희생을 제외하고 최적화를 하는 옵션이다. T를 2로 두었을 경우와 T를 4이상으로 두었을 경우 assembly 코드 상에서 크게 달라진 점이 존재했다.

먼저, tilling 하는 크기가 커짐에 따라,  $rresult = ra * rb$  에서 mla하는 assembly code가 4개에서 1개로 줄어들었다. 즉, T가 2였을 경우 한 iteration 당 12개에 해당하는 명령어를 사용하였다면, T가 4 이상인 경우에 4개의 명령어만 사용하도록 컴파일되게 설계되었다. 또한, branch 명령어의 경우 T=2인 경우 4개의 branch 명령어가, T가 4이상인 경우 6개의 branch명령어가 사용되었다. 이는 T=2의 경우 가장 안쪽에 해당하는 ji에 대한 반복문과 그 밖을 감싸는 rb에 대한 반복문을 branch 명령어를 사용하지 않고 assembly code안에서 동작을 구현했다면, T가 커짐에 따라서, 두 반복문에 해당되는 부분을 branch명령어를 통해 구현한 차이라고 생각한다.

두번째로, 크기가 커짐에 따라 add명령어에 사용되는 immediate들 또한 달라졌다. Ra와 rb의 주소를 계속해서 늘려 계산하기 위해 사용되는 부분에 4\*T에 해당되는 immediate가 사용되어 4bit int type 정수 표현을 사용했다.

### 3. (p10)L1CacheDisable, T=2,4,8,16

<p>T=2</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 4278215213, 4277206788, 4277340858, 4277248323, 12831744.131 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 2410796699, 2409693712, 2410217589, 2410210685, 7230631.583 ----Benchmarking Complete----</pre>	<p>T=4</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 4273379848, 4272628879, 4272947776, 4272933629, 12818800.050 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 3520385760, 3519955460, 3520007441, 3519966649, 10559899.258 ----Benchmarking Complete----</pre>
<p>T=8</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 4278174012, 4277206584, 4277330172, 4277240141, 12831719.585 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 3269014511, 3268617121, 3268972903, 3269012174, 9807035.882 ----Benchmarking Complete----</pre>	<p>T=16</p> <pre>----Benchmarking starting---- CPU_FREQ_HZ=666666687, TIMER_FREQ_HZ=333333343 Matrix size= 512 * 512 Algorithm validation success! Case 0: Non-cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 4276716395, 4276698375, 4276709118, 4276709551, 12830127.815 Case 1: Cache optimized matrix multiply Nr,      Max,      Min,      Average,  Fltr Avg,  Fltr Avg(us) 10, 3151986447, 3151732202, 3151759240, 3151734218, 9455202.037 ----Benchmarking Complete----</pre>

앞서와의 차이에는 L1Dcache를 사용하지 않는 것에 있다. ZYNQ는 L1Cache를 사용하지 않으면 L2 Cache로 이동하고 해당 Cache에도 데이터가 존재하지 않으면 직접 DDR로 이동하여 data를 가져오게 된다. L2Cache는 512KB로 구성되어 있으며, 8way-set associative Cache이다. L2 Cahce는 32bytes의 Line Size를 가지므로 L1 Cache와 동일하게 8Word단위로 구성된다.

Case 0와 같이 Tiling이 되지 않은 경우 Cache의 크기가 커져 한 cache에 올릴 수 있는 데이터가 많아져 기존의 L1 Cache만을 사용했을 때보다 main memory에 접근하는 횟수가 줄어들게 되어 시간이 조금 더 빨라짐을 확인 할 수 있었다.

Case 1의 경우 T가 2일 때 가장 좋은 실행시간을 보이며 T가 4이상이면 T의 크기가 커질수록 실행시간이 짧아짐을 확인 할 수 있었다. 기존의 L1 Cache와 동일하게 Cache Line의 크기가 32bytes이기 때문에 T가 8일 때 가장 좋은 실행시간을 예상했으나 예상과는 다른 결과가 나왔다. 가장 큰 이유로 생각해 볼 수 있는 것은, L2 Cache의 Cache Line에 따라 데이터의 접근이 다르다는 점이 있을 것으로 생각한다.