NUMA - 3

산업용병렬처리특론

정내훈

2020년도 1학기

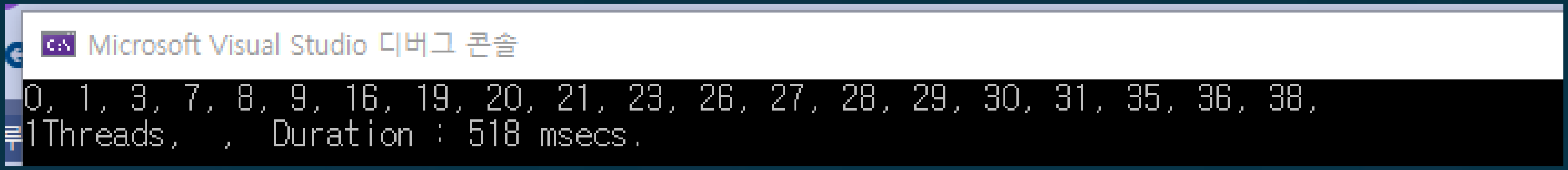
한국산업기술대학교 스마트팩토리융합학과

목차

- CX 알고리즘 구현
- 과제 리뷰

구현

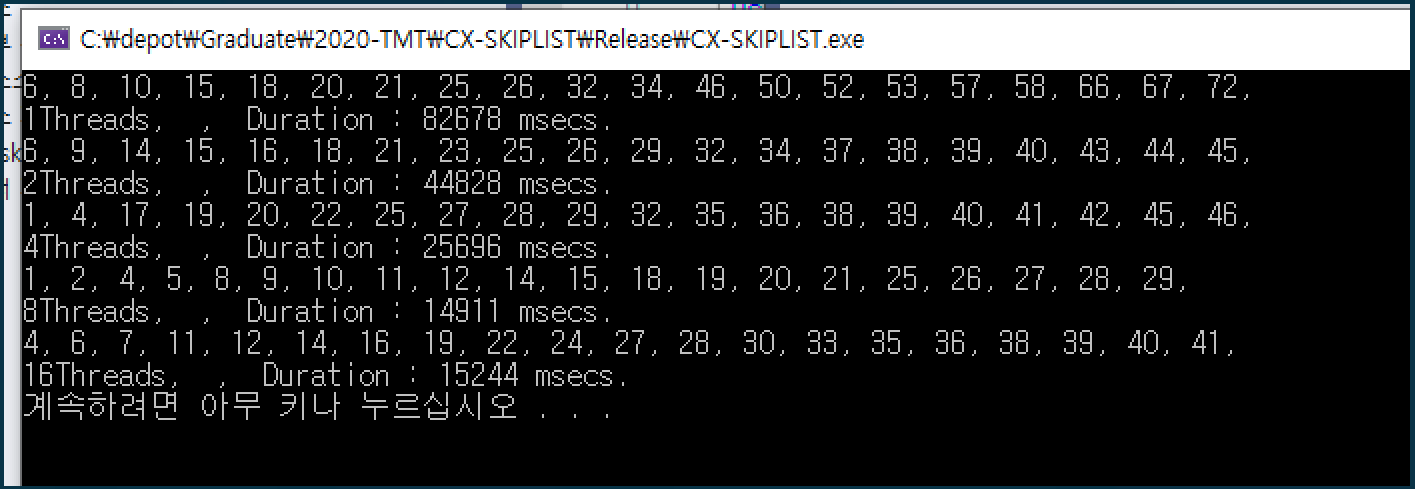
- 싱글 쓰레드 스킵리스트의 성능
  - Intel® Core™ i7-7700 CPU @3.60GHz
    - Quad Core with Hyperthread, logical 8 core



```
0, 1, 3, 7, 8, 9, 16, 19, 20, 21, 23, 26, 27, 28, 29, 30, 31, 35, 36, 38,
1Threads,   ,  Duration : 518 msecs.
```

구현

- Lock Free Universal의 성능
  - 교재 그대로 구현
  - 400만회 벤치마킹 => 4만회
    - 반복 실행 특성상 실행 시간은 횟수의 제곱으로 증가.

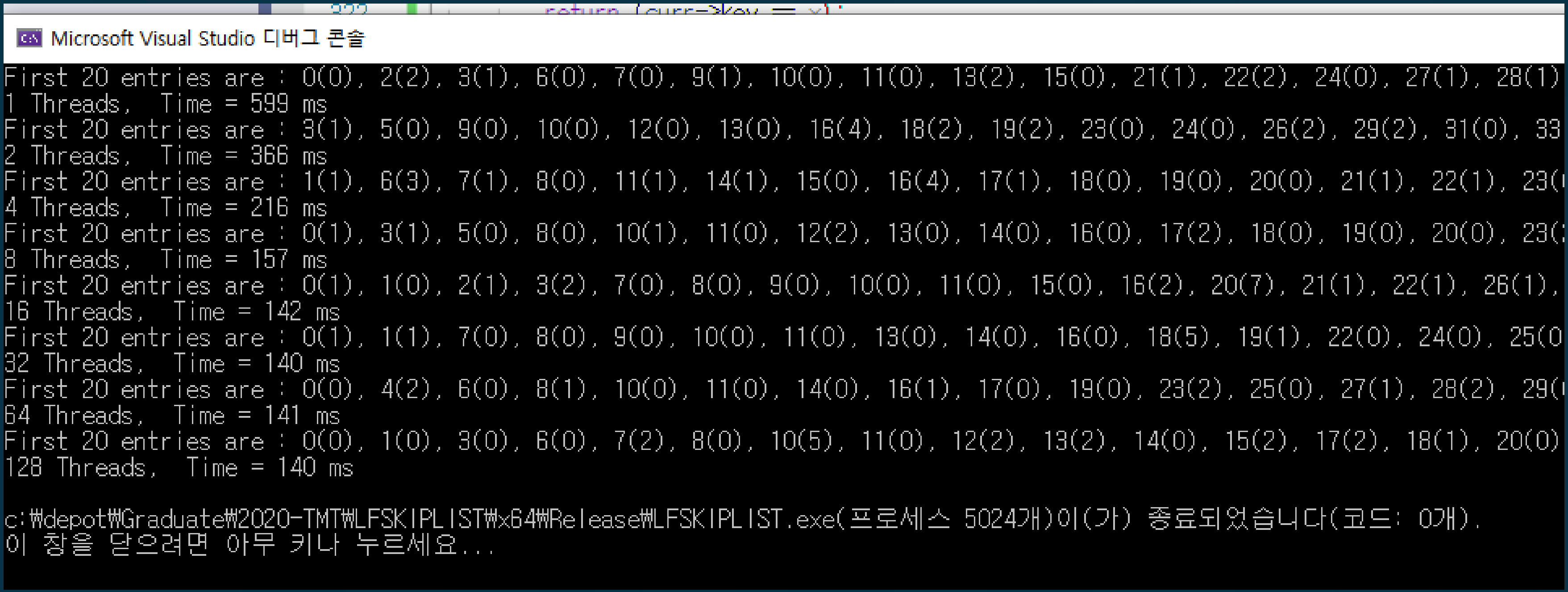


C:₩depot₩Graduate₩2020-TMT₩CX-SKIPLIST₩Release₩CX-SKIPLIST.exe

```
6, 8, 10, 15, 18, 20, 21, 25, 26, 32, 34, 46, 50, 52, 53, 57, 58, 66, 67, 72,
1Threads,  ,  Duration : 82678 msecs.
6, 9, 14, 15, 16, 18, 21, 23, 25, 26, 29, 32, 34, 37, 38, 39, 40, 43, 44, 45,
2Threads,  ,  Duration : 44828 msecs.
1, 4, 17, 19, 20, 22, 25, 27, 28, 29, 32, 35, 36, 38, 39, 40, 41, 42, 45, 46,
4Threads,  ,  Duration : 25696 msecs.
1, 2, 4, 5, 8, 9, 10, 11, 12, 14, 15, 18, 19, 20, 21, 25, 26, 27, 28, 29,
8Threads,  ,  Duration : 14911 msecs.
4, 6, 7, 11, 12, 14, 16, 19, 22, 24, 27, 28, 30, 33, 35, 36, 38, 39, 40, 41,
16Threads,  ,  Duration : 15244 msecs.
계속하려면 아무 키나 누르십시오 . . .
```

구현

- 비교를 위한 Lock Free Skiplist



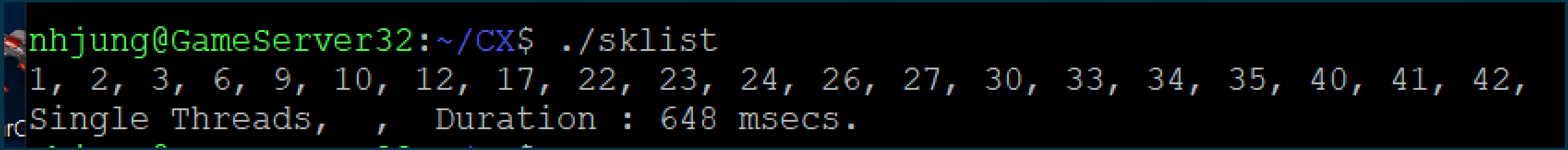
```
First 20 entries are : 0(0), 2(2), 3(1), 6(0), 7(0), 9(1), 10(0), 11(0), 13(2), 15(0), 21(1), 22(2), 24(0), 27(1), 28(1)
1 Threads,  Time = 599 ms
First 20 entries are : 3(1), 5(0), 9(0), 10(0), 12(0), 13(0), 16(4), 18(2), 19(2), 23(0), 24(0), 26(2), 29(2), 31(0), 33
2 Threads,  Time = 366 ms
First 20 entries are : 1(1), 6(3), 7(1), 8(0), 11(1), 14(1), 15(0), 16(4), 17(1), 18(0), 19(0), 20(0), 21(1), 22(1), 23(
4 Threads,  Time = 216 ms
First 20 entries are : 0(1), 3(1), 5(0), 8(0), 10(1), 11(0), 12(2), 13(0), 14(0), 16(0), 17(2), 18(0), 19(0), 20(0), 23(
8 Threads,  Time = 157 ms
First 20 entries are : 0(1), 1(0), 2(1), 3(2), 7(0), 8(0), 9(0), 10(0), 11(0), 15(0), 16(2), 20(7), 21(1), 22(1), 26(1),
16 Threads,  Time = 142 ms
First 20 entries are : 0(1), 1(1), 7(0), 8(0), 9(0), 10(0), 11(0), 13(0), 14(0), 16(0), 18(5), 19(1), 22(0), 24(0), 25(0
32 Threads,  Time = 140 ms
First 20 entries are : 0(0), 4(2), 6(0), 8(1), 10(0), 11(0), 14(0), 16(1), 17(0), 19(0), 23(2), 25(0), 27(1), 28(2), 29(
64 Threads,  Time = 141 ms
First 20 entries are : 0(0), 1(0), 3(0), 6(0), 7(2), 8(0), 10(5), 11(0), 12(2), 13(2), 14(0), 15(2), 17(2), 18(1), 20(0)
128 Threads,  Time = 140 ms

c:₩depot₩Graduate₩2020-TMT₩LFSKIPLIST₩x64₩Release₩LFSKIPLIST.exe(프로세스 5024개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

LINUX

- 잠깐!!!
- 우리는 NUMA를 하고 있고, NUMA machine에서 돌려야.
- 환경
  - Linux 5.3.0-51-generic #44~18.04.2-Ubuntu SMP
  - Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
  - 4 CPU, 8 core per CPU, 16 logical thread per CPU
  - gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)

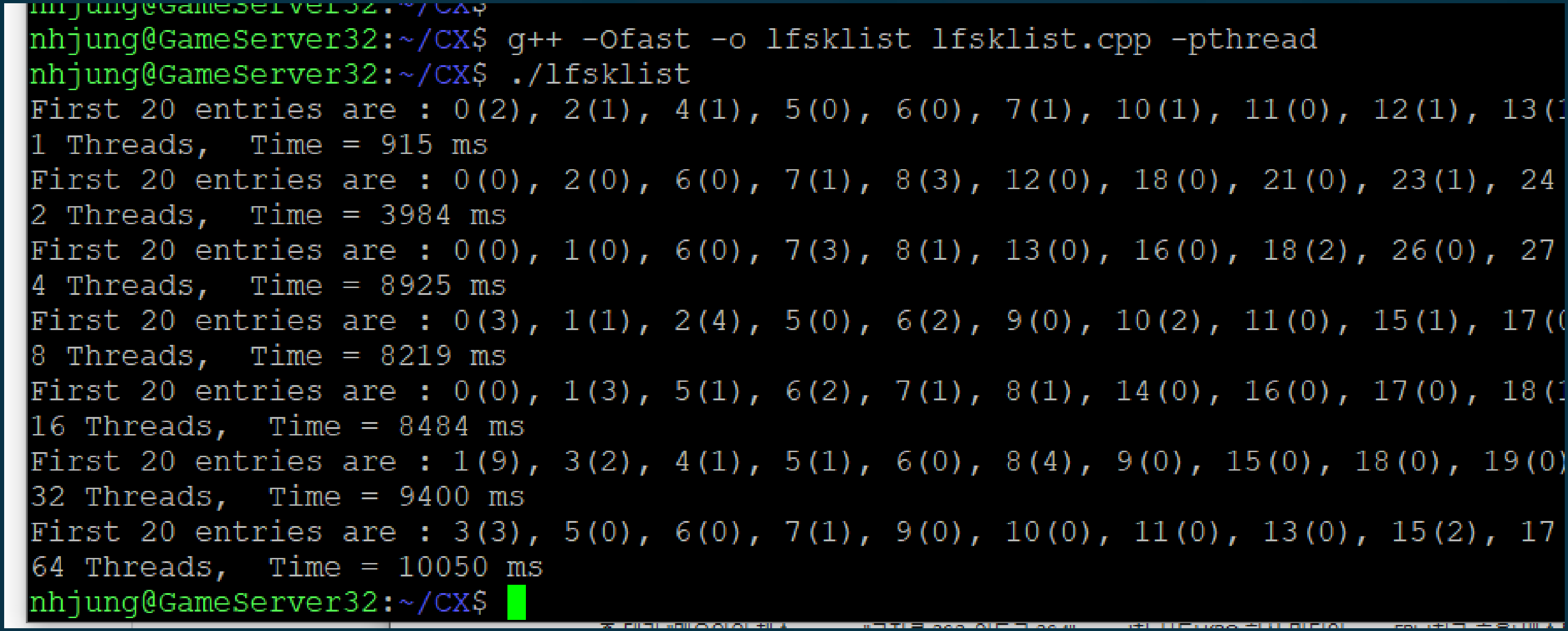
LINUX

- Single Thread Skiplist



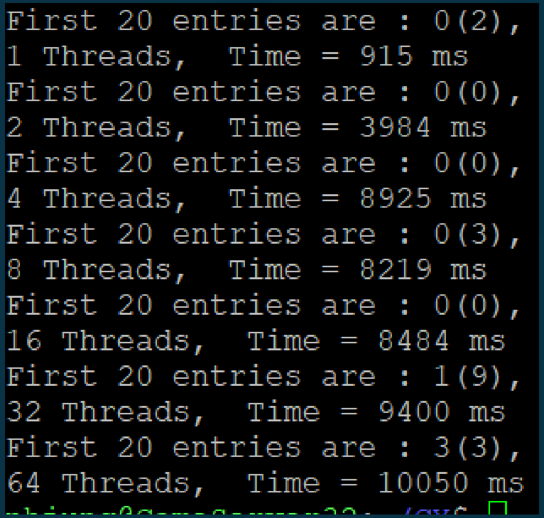
```
nhjung@GameServer32:~/CX$ ./sklist
1, 2, 3, 6, 9, 10, 12, 17, 22, 23, 24, 26, 27, 30, 33, 34, 35, 40, 41, 42,
Single Threads,   ,  Duration : 648 msecs.
```

- 비교용 LF SkipList



```
nhjung@GameServer32:~/CX$
nhjung@GameServer32:~/CX$ g++ -Ofast -o lfsklist lfsklist.cpp -pthread
nhjung@GameServer32:~/CX$ ./lfsklist
First 20 entries are : 0(2), 2(1), 4(1), 5(0), 6(0), 7(1), 10(1), 11(0), 12(1), 13(
1 Threads,  Time = 915 ms
First 20 entries are : 0(0), 2(0), 6(0), 7(1), 8(3), 12(0), 18(0), 21(0), 23(1), 24
2 Threads,  Time = 3984 ms
First 20 entries are : 0(0), 1(0), 6(0), 7(3), 8(1), 13(0), 16(0), 18(2), 26(0), 27
4 Threads,  Time = 8925 ms
First 20 entries are : 0(3), 1(1), 2(4), 5(0), 6(2), 9(0), 10(2), 11(0), 15(1), 17(
8 Threads,  Time = 8219 ms
First 20 entries are : 0(0), 1(3), 5(1), 6(2), 7(1), 8(1), 14(0), 16(0), 17(0), 18(
16 Threads,  Time = 8484 ms
First 20 entries are : 1(9), 3(2), 4(1), 5(1), 6(0), 8(4), 9(0), 15(0), 18(0), 19(0)
32 Threads,  Time = 9400 ms
First 20 entries are : 3(3), 5(0), 6(0), 7(1), 9(0), 10(0), 11(0), 13(0), 15(2), 17
64 Threads,  Time = 10050 ms
nhjung@GameServer32:~/CX$ 
```

LINUX

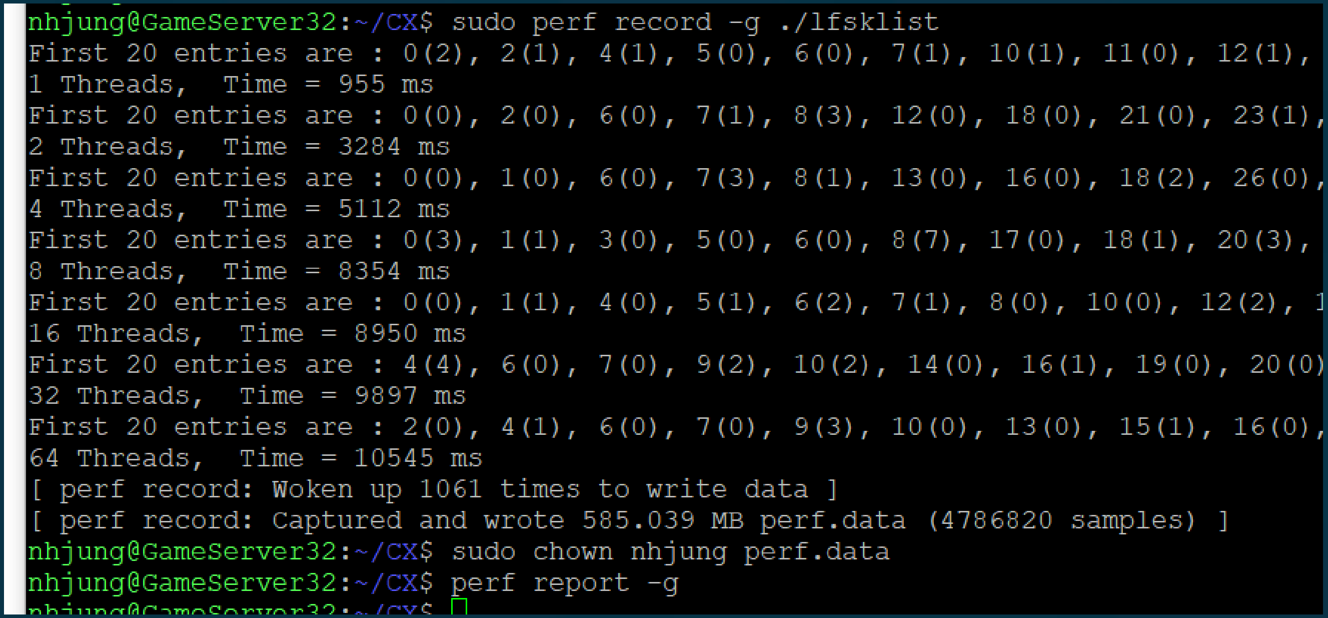


```
First 20 entries are : 0(2),
1 Threads,  Time = 915 ms
First 20 entries are : 0(0),
2 Threads,  Time = 3984 ms
First 20 entries are : 0(0),
4 Threads,  Time = 8925 ms
First 20 entries are : 0(3),
8 Threads,  Time = 8219 ms
First 20 entries are : 0(0),
16 Threads,  Time = 8484 ms
First 20 entries are : 1(9),
32 Threads,  Time = 9400 ms
First 20 entries are : 3(3),
64 Threads,  Time = 10050 ms
```

- 응????? 왜???
- 원인을 알아보자

LINUX

- perf 명령어 사용
  - 예전에는 gperf 사용, 리누스 토발즈가 perf강력추천
    - 그런데 멀티쓰레드에서는 gperf가 더 좋아 보임

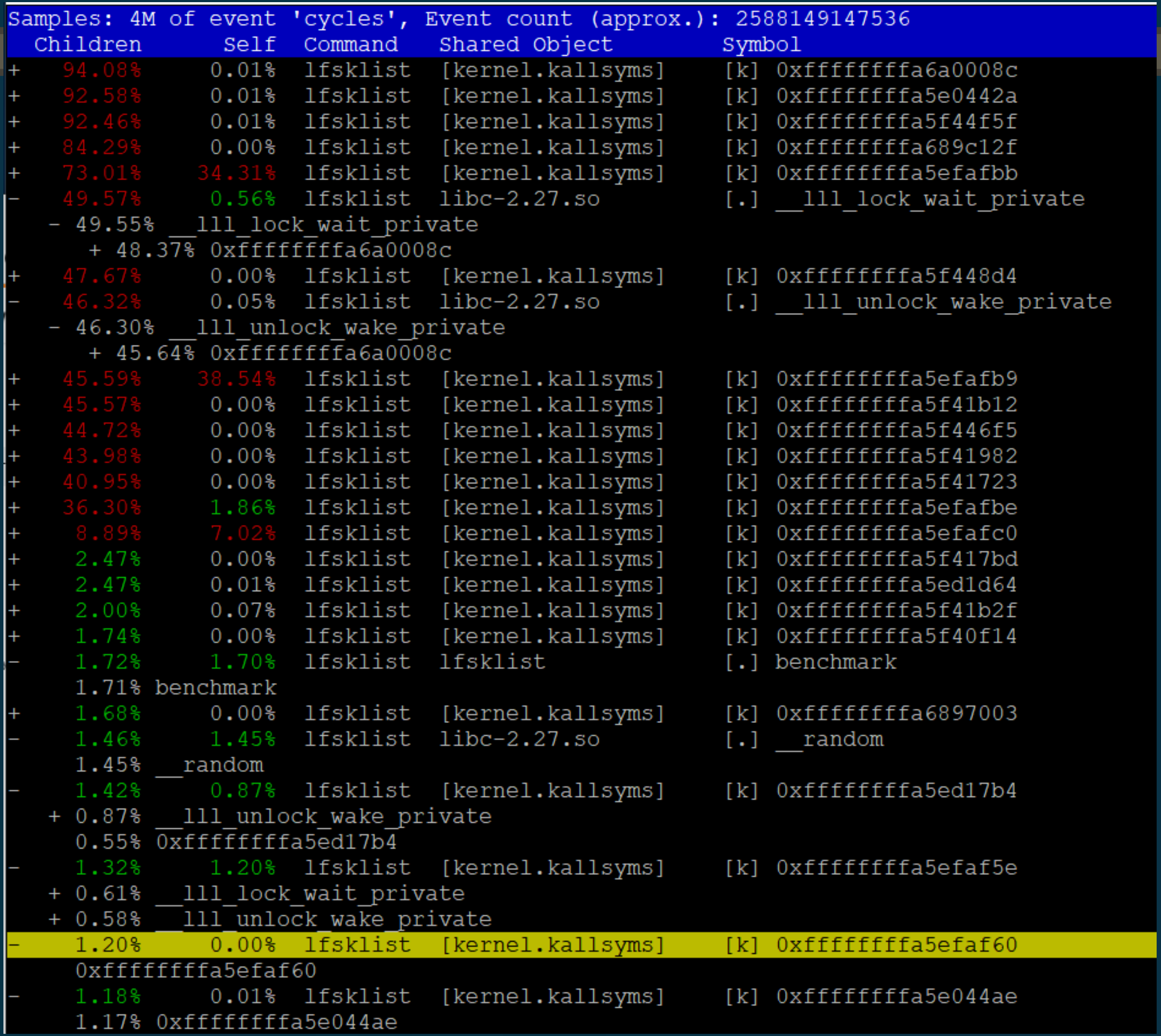


```
nhjung@GameServer32:~/CX$ sudo perf record -g ./lfsklist
First 20 entries are : 0(2), 2(1), 4(1), 5(0), 6(0), 7(1), 10(1), 11(0), 12(1),
1 Threads,  Time = 955 ms
First 20 entries are : 0(0), 2(0), 6(0), 7(1), 8(3), 12(0), 18(0), 21(0), 23(1),
2 Threads,  Time = 3284 ms
First 20 entries are : 0(0), 1(0), 6(0), 7(3), 8(1), 13(0), 16(0), 18(2), 26(0),
4 Threads,  Time = 5112 ms
First 20 entries are : 0(3), 1(1), 3(0), 5(0), 6(0), 8(7), 17(0), 18(1), 20(3),
8 Threads,  Time = 8354 ms
First 20 entries are : 0(0), 1(1), 4(0), 5(1), 6(2), 7(1), 8(0), 10(0), 12(2), 1
16 Threads,  Time = 8950 ms
First 20 entries are : 4(4), 6(0), 7(0), 9(2), 10(2), 14(0), 16(1), 19(0), 20(0)
32 Threads,  Time = 9897 ms
First 20 entries are : 2(0), 4(1), 6(0), 7(0), 9(3), 10(0), 13(0), 15(1), 16(0),
64 Threads,  Time = 10545 ms
[ perf record: Woken up 1061 times to write data ]
[ perf record: Captured and wrote 585.039 MB perf.data (4786820 samples) ]
nhjung@GameServer32:~/CX$ sudo chown nhjung perf.data
nhjung@GameServer32:~/CX$ perf report -g
nhjung@GameServer32:~/CX$
```

https://stackoverflow.com/questions/2229336/linux-application-profiling
http://www.brendangregg.com/perf.html
https://easyperf.net/blog/2019/10/05/Performance-Analysis-Of-MT-apps

LINUX

- perf



```
Samples: 4M of event 'cycles', Event count (approx.): 2588149147536
  Children      Self  Command   Shared Object         Symbol
+   94.08%     0.01%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa6a0008c
+   92.58%     0.01%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5e0442a
+   92.46%     0.01%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f44f5f
+   84.29%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa689c12f
+   73.01%    34.31%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efafbb
-   49.57%     0.56%  lfsklist  libc-2.27.so          [.] __lll_lock_wait_private
   - 49.55% __lll_lock_wait_private
      + 48.37% 0xffffffffa6a0008c
+   47.67%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f448d4
-   46.32%     0.05%  lfsklist  libc-2.27.so          [.] __lll_unlock_wake_private
   - 46.30% __lll_unlock_wake_private
      + 45.64% 0xffffffffa6a0008c
+   45.59%    38.54%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efafb9
+   45.57%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f41b12
+   44.72%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f446f5
+   43.98%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f41982
+   40.95%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f41723
+   36.30%     1.86%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efafbe
+    8.89%     7.02%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efafc0
+    2.47%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f417bd
+    2.47%     0.01%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5ed1d64
+    2.00%     0.07%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f41b2f
+    1.74%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5f40f14
-    1.72%     1.70%  lfsklist  lfsklist              [.] benchmark
     1.71% benchmark
+    1.68%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa6897003
-    1.46%     1.45%  lfsklist  libc-2.27.so          [.] __random
     1.45% __random
-    1.42%     0.87%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5ed17b4
   + 0.87% __lll_unlock_wake_private
     0.55% 0xffffffffa5ed17b4
-    1.32%     1.20%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efaf5e
   + 0.61% __lll_lock_wait_private
   + 0.58% __lll_unlock_wake_private
-    1.20%     0.00%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5efaf60
     0xffffffffa5efaf60
-    1.18%     0.01%  lfsklist  [kernel.kallsyms]     [k] 0xffffffffa5e044ae
     1.17% 0xffffffffa5e044ae
```

LINUX

- perf

```
49.55% __lll_lock_wait_private
45.64% __lll_unlock_wake_private
 1.71% benchmark
```

- mutex 쓴 적이 없는데???
- 원인 : rand() 함수
  - 내부적으로 standard library lock을 사용.
  - rand()함수가 reentrant하지 않기 때문.
    - seed update
  - https://brooker.co.za/blog/2014/12/06/random.html
  - rand()를 사용한 모든 멀티쓰레드 프로그램에 큰 문제!!!!

LINUX

- rand()문제의 해결책
  - "man 3 rand" 에 나옴

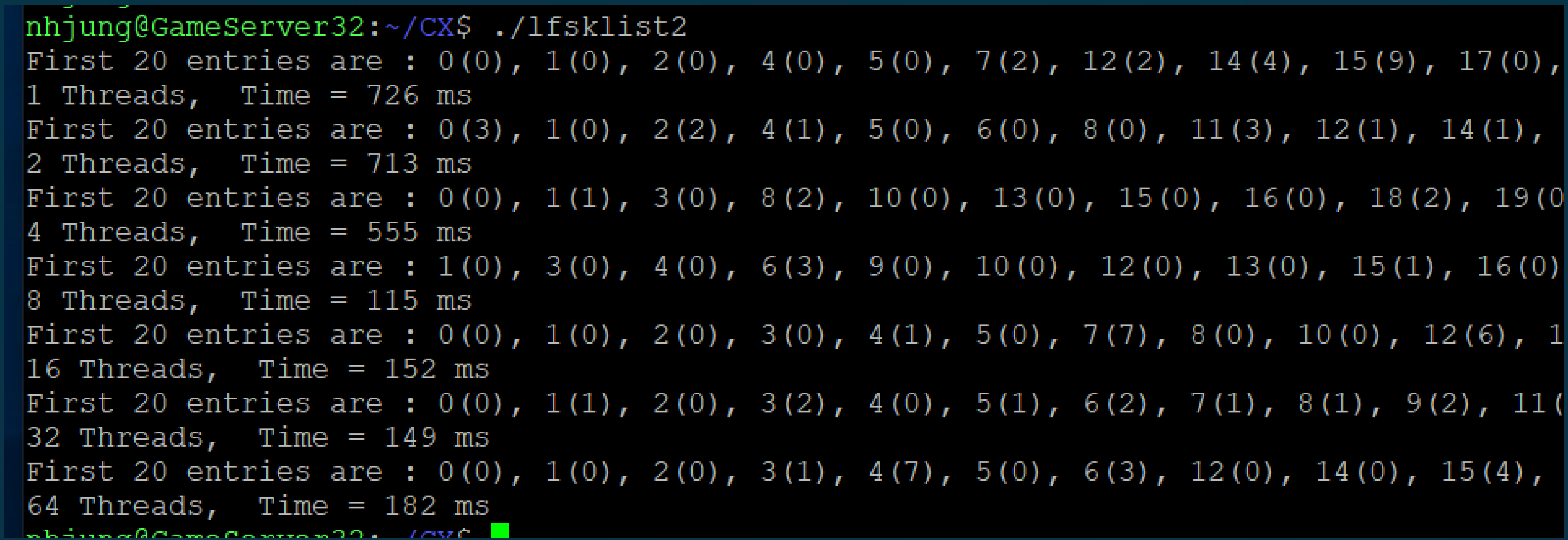
```
static unsigned long next = 1;

/* RAND_MAX assumed to be 32767 */
int myrand(void) {
   next = next * 1103515245 + 12345;
   return((unsigned)(next/65536) % 32768);
}
```

  - 우리는

```
thread_local unsigned long g_next = 1;

/* RAND_MAX assumed to be 32767 */
int rand_mt(void) {
   g_next = g_next * 1103515245 + 12345;
   return((unsigned)(g_next/65536) % 32768);
}
```

LINUX

- LF Skiplist 성능

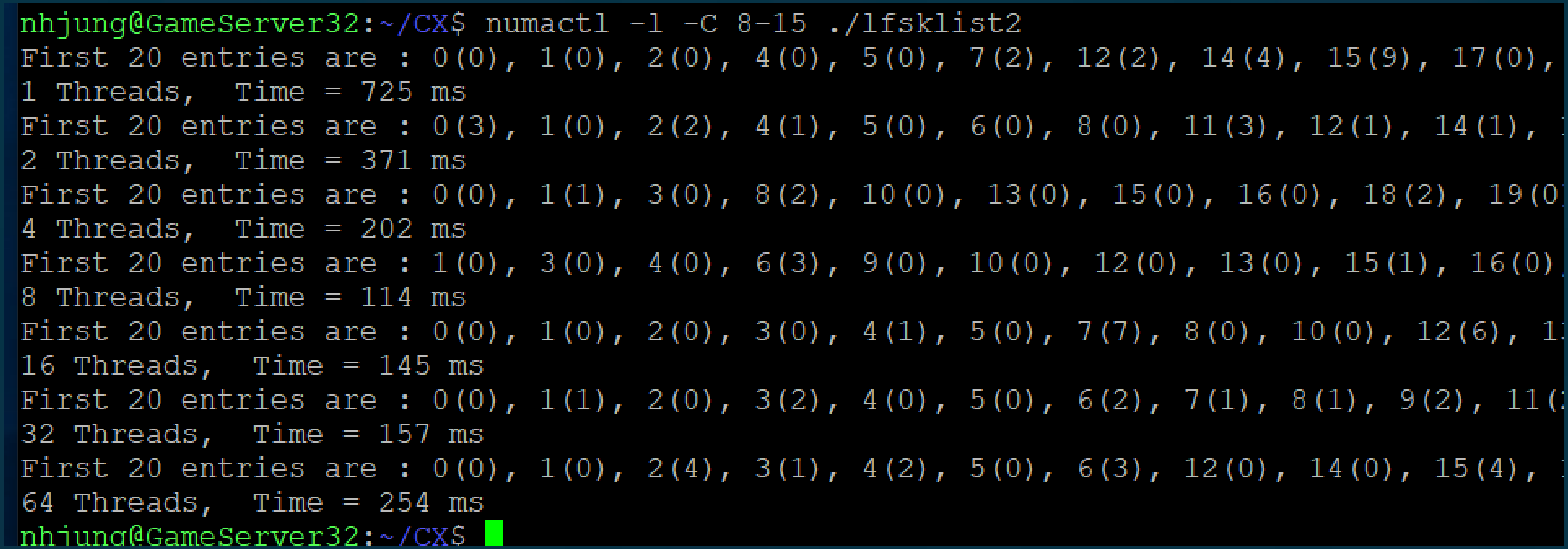


```
nhjung@GameServer32:~/CX$ ./lfsklist2
First 20 entries are : 0(0), 1(0), 2(0), 4(0), 5(0), 7(2), 12(2), 14(4), 15(9), 17(0),
1 Threads,  Time = 726 ms
First 20 entries are : 0(3), 1(0), 2(2), 4(1), 5(0), 6(0), 8(0), 11(3), 12(1), 14(1),
2 Threads,  Time = 713 ms
First 20 entries are : 0(0), 1(1), 3(0), 8(2), 10(0), 13(0), 15(0), 16(0), 18(2), 19(0
4 Threads,  Time = 555 ms
First 20 entries are : 1(0), 3(0), 4(0), 6(3), 9(0), 10(0), 12(0), 13(0), 15(1), 16(0)
8 Threads,  Time = 115 ms
First 20 entries are : 0(0), 1(0), 2(0), 3(0), 4(1), 5(0), 7(7), 8(0), 10(0), 12(6), 1
16 Threads,  Time = 152 ms
First 20 entries are : 0(0), 1(1), 2(0), 3(2), 4(0), 5(1), 6(2), 7(1), 8(1), 9(2), 11(
32 Threads,  Time = 149 ms
First 20 entries are : 0(0), 1(0), 2(0), 3(1), 4(7), 5(0), 6(3), 12(0), 14(0), 15(4),
64 Threads,  Time = 182 ms
nhjung@GameServer32:~/CX$ 
```

- 2개 쓰레드에서 성능향상 미비
  - default가 thread를 Node에 고르게 배정

LINUX

- 한 노드에서만 실행

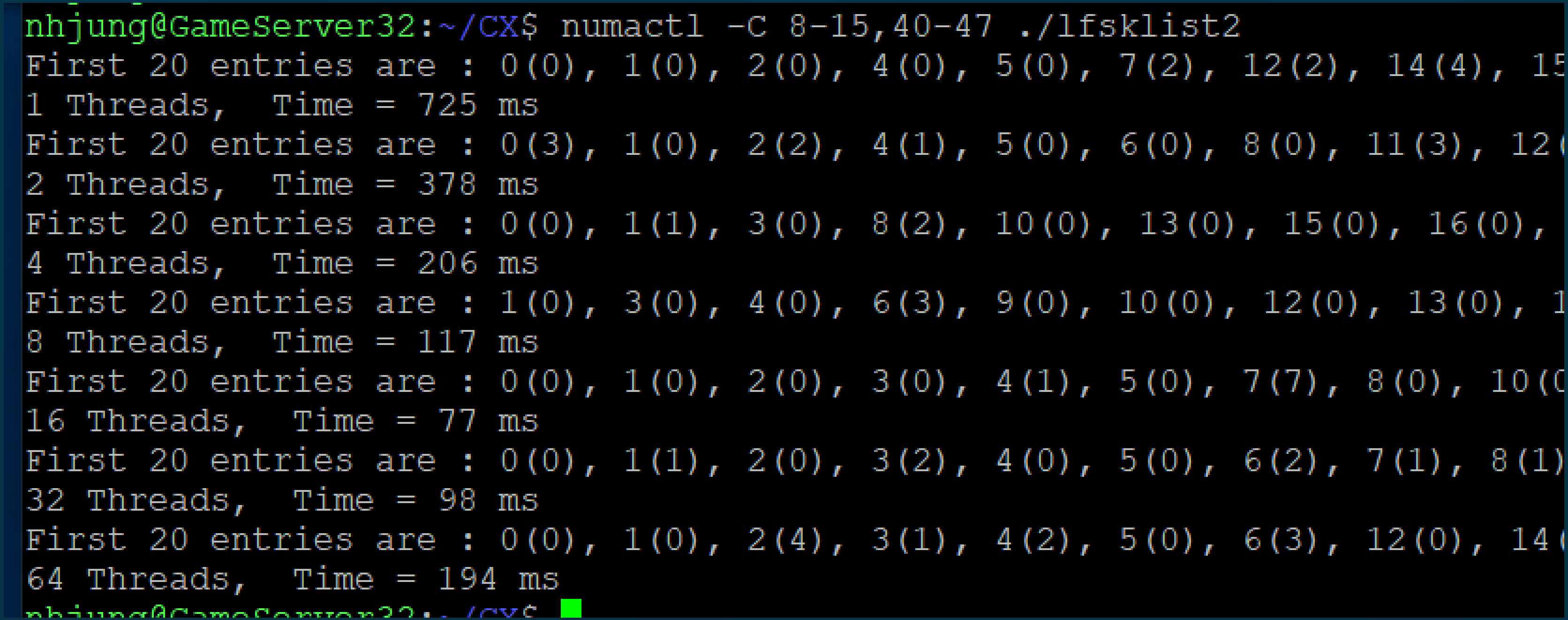


```
nhjung@GameServer32:~/CX$ numactl -l -C 8-15 ./lfsklist2
First 20 entries are : 0(0), 1(0), 2(0), 4(0), 5(0), 7(2), 12(2), 14(4), 15(9), 17(0),
1 Threads,  Time = 725 ms
First 20 entries are : 0(3), 1(0), 2(2), 4(1), 5(0), 6(0), 8(0), 11(3), 12(1), 14(1),
2 Threads,  Time = 371 ms
First 20 entries are : 0(0), 1(1), 3(0), 8(2), 10(0), 13(0), 15(0), 16(0), 18(2), 19(0
4 Threads,  Time = 202 ms
First 20 entries are : 1(0), 3(0), 4(0), 6(3), 9(0), 10(0), 12(0), 13(0), 15(1), 16(0)
8 Threads,  Time = 114 ms
First 20 entries are : 0(0), 1(0), 2(0), 3(0), 4(1), 5(0), 7(7), 8(0), 10(0), 12(6), 1
16 Threads,  Time = 145 ms
First 20 entries are : 0(0), 1(1), 2(0), 3(2), 4(0), 5(0), 6(2), 7(1), 8(1), 9(2), 11(
32 Threads,  Time = 157 ms
First 20 entries are : 0(0), 1(0), 2(4), 3(1), 4(2), 5(0), 6(3), 12(0), 14(0), 15(4),
64 Threads,  Time = 254 ms
nhjung@GameServer32:~/CX$
```

– 8개 쓰레드까지 성능 향상

LINUX

- 한 노드(+Hyperthread) 실행

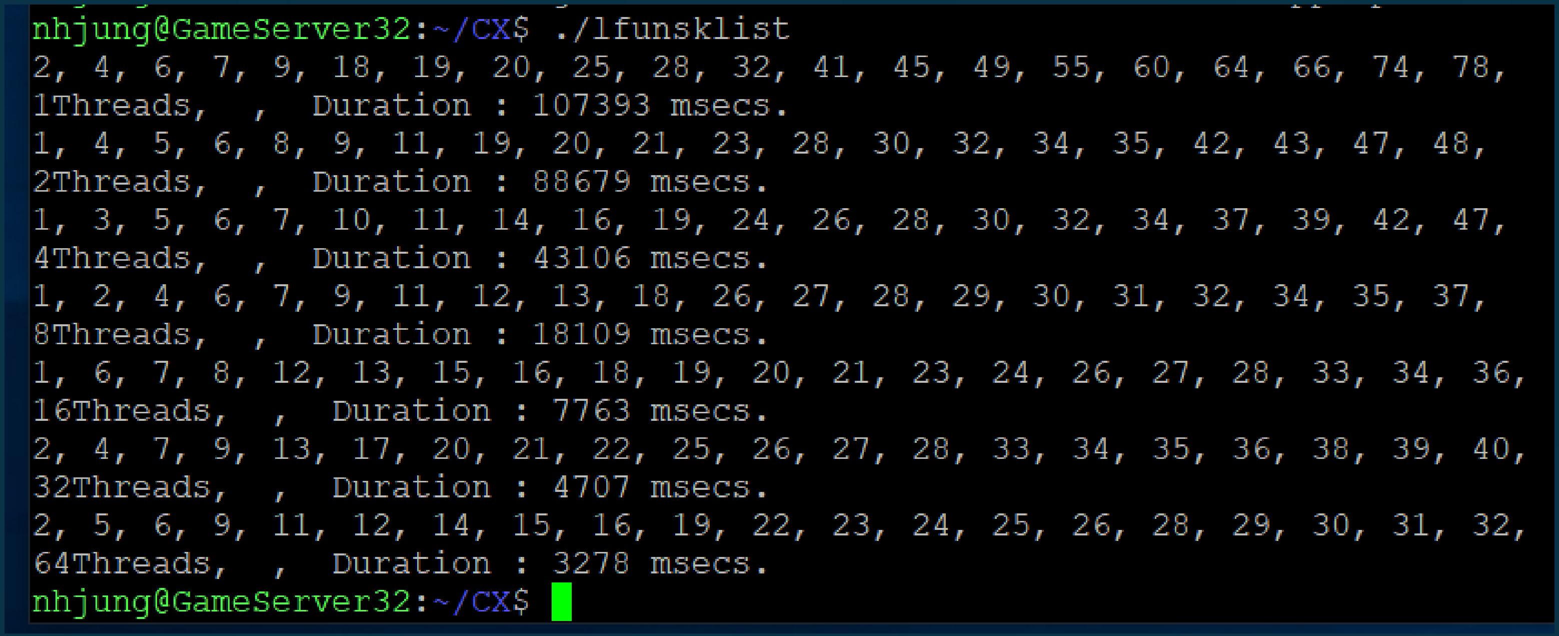


```
nhjung@GameServer32:~/CX$ numactl -C 8-15,40-47 ./lfsklist2
First 20 entries are : 0(0), 1(0), 2(0), 4(0), 5(0), 7(2), 12(2), 14(4), 15
1 Threads,  Time = 725 ms
First 20 entries are : 0(3), 1(0), 2(2), 4(1), 5(0), 6(0), 8(0), 11(3), 12(
2 Threads,  Time = 378 ms
First 20 entries are : 0(0), 1(1), 3(0), 8(2), 10(0), 13(0), 15(0), 16(0),
4 Threads,  Time = 206 ms
First 20 entries are : 1(0), 3(0), 4(0), 6(3), 9(0), 10(0), 12(0), 13(0), 1
8 Threads,  Time = 117 ms
First 20 entries are : 0(0), 1(0), 2(0), 3(0), 4(1), 5(0), 7(7), 8(0), 10(0
16 Threads,  Time = 77 ms
First 20 entries are : 0(0), 1(1), 2(0), 3(2), 4(0), 5(0), 6(2), 7(1), 8(1)
32 Threads,  Time = 98 ms
First 20 entries are : 0(0), 1(0), 2(4), 3(1), 4(2), 5(0), 6(3), 12(0), 14(
64 Threads,  Time = 194 ms
nhjung@GameServer32:~/CX$
```

- 16개 쓰레드까지 성능 향상

만능 머신

- 교재에 있는 LFUniversal 그대로 적용
- 루프 횟수 400만 -> 4만
  - 실행시간은 횟수의 제곱.



```
nhjung@GameServer32:~/CX$ ./lfunsklist
2, 4, 6, 7, 9, 18, 19, 20, 25, 28, 32, 41, 45, 49, 55, 60, 64, 66, 74, 78,
1Threads,  ,  Duration : 107393 msecs.
1, 4, 5, 6, 8, 9, 11, 19, 20, 21, 23, 28, 30, 32, 34, 35, 42, 43, 47, 48,
2Threads,  ,  Duration : 88679 msecs.
1, 3, 5, 6, 7, 10, 11, 14, 16, 19, 24, 26, 28, 30, 32, 34, 37, 39, 42, 47,
4Threads,  ,  Duration : 43106 msecs.
1, 2, 4, 6, 7, 9, 11, 12, 13, 18, 26, 27, 28, 29, 30, 31, 32, 34, 35, 37,
8Threads,  ,  Duration : 18109 msecs.
1, 6, 7, 8, 12, 13, 15, 16, 18, 19, 20, 21, 23, 24, 26, 27, 28, 33, 34, 36,
16Threads,  ,  Duration : 7763 msecs.
2, 4, 7, 9, 13, 17, 20, 21, 22, 25, 26, 27, 28, 33, 34, 35, 36, 38, 39, 40,
32Threads,  ,  Duration : 4707 msecs.
2, 5, 6, 9, 11, 12, 14, 15, 16, 19, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32,
64Threads,  ,  Duration : 3278 msecs.
nhjung@GameServer32:~/CX$ 
```

만능 머신

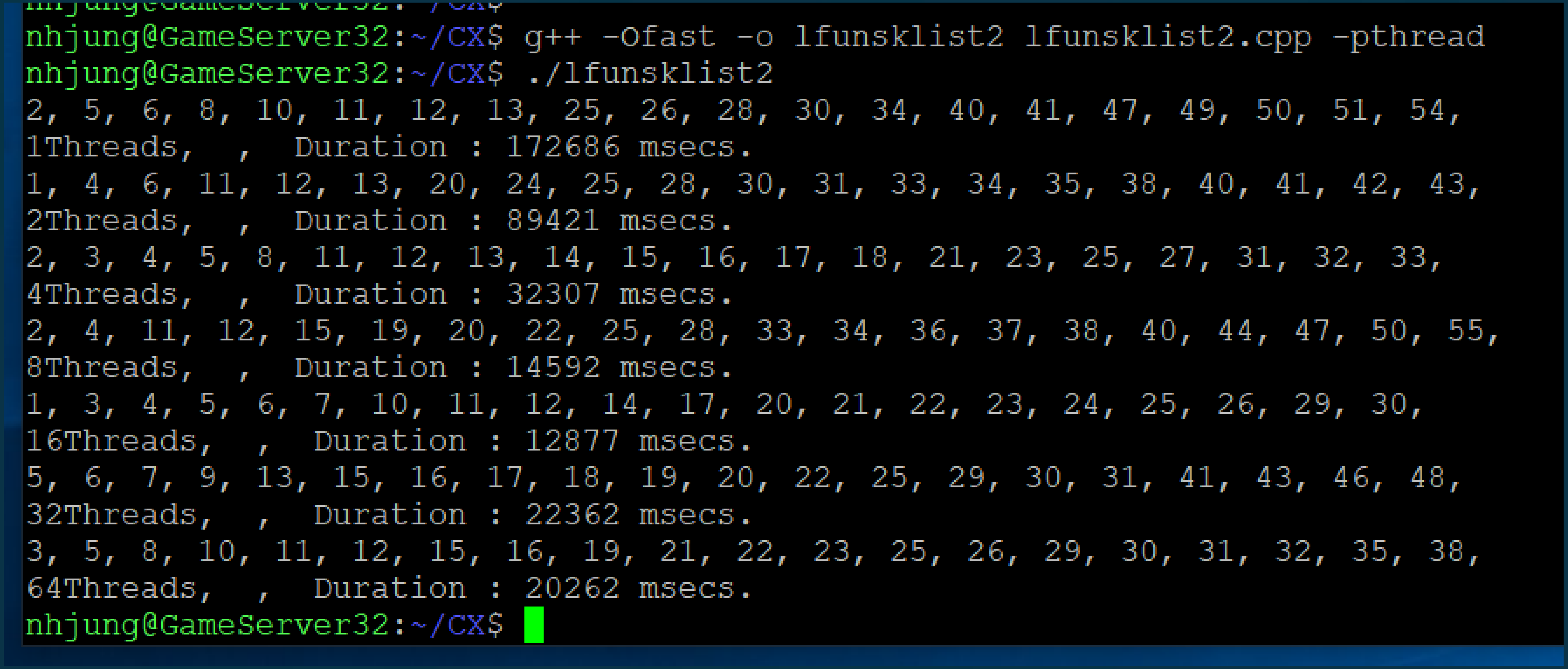
- 최적화
  - 객체 재사용. 쓰레드마다 객체 따로 둠

```
thread_local SeqObject local_object;
thread_local NODE* local_tail;
```

```
Response Apply(const Invocation& invoc) {
   NODE* prefer = new NODE{ invoc };
   while (prefer->seq == 0) {
…
   }

   NODE* curr = local_tail->next;
   while (curr != prefer) {
      local_object.Apply(curr->invoc);
      curr = curr->next;
   }
   local_tail = curr;
   return local_object.Apply(curr->invoc);
}
```

만능 머신

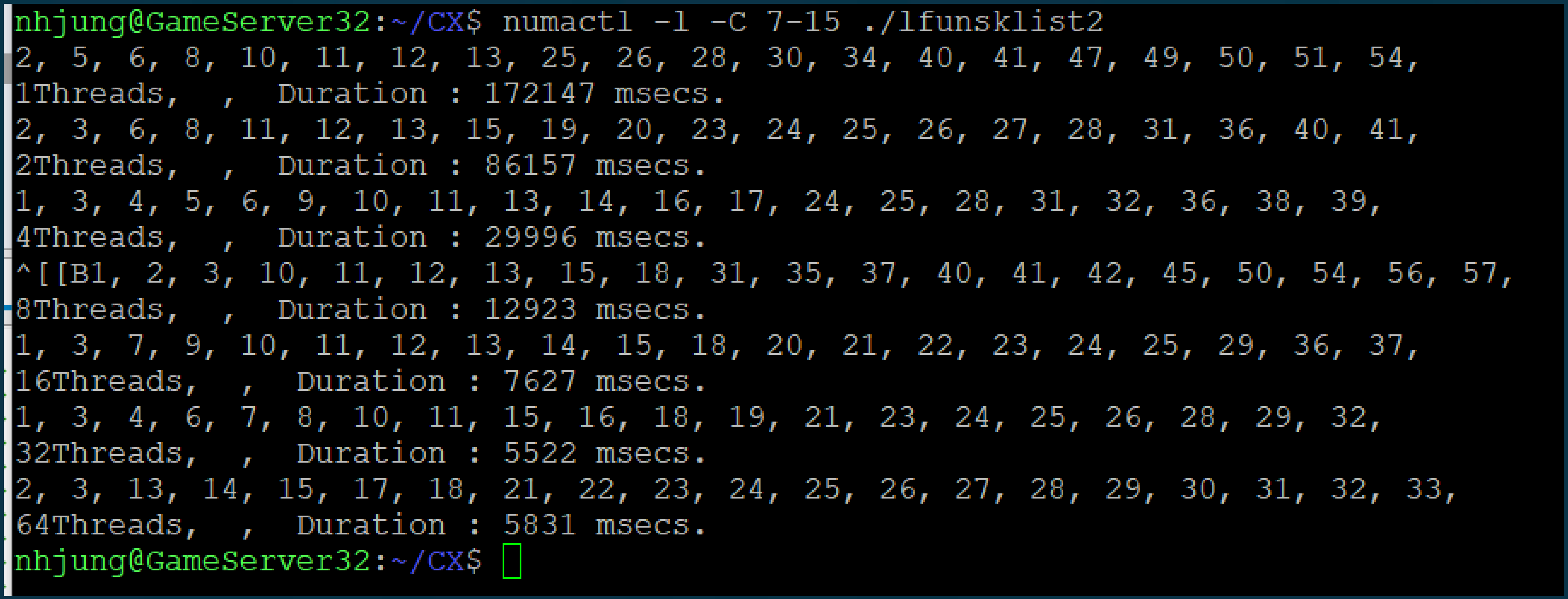
- 최적화
  - 객체 재사용. (루프횟수 다시 400만으로)



```
nhjung@GameServer32:~/CX$
nhjung@GameServer32:~/CX$ g++ -Ofast -o lfunsklist2 lfunsklist2.cpp -pthread
nhjung@GameServer32:~/CX$ ./lfunsklist2
2, 5, 6, 8, 10, 11, 12, 13, 25, 26, 28, 30, 34, 40, 41, 47, 49, 50, 51, 54,
1Threads,  ,  Duration : 172686 msecs.
1, 4, 6, 11, 12, 13, 20, 24, 25, 28, 30, 31, 33, 34, 35, 38, 40, 41, 42, 43,
2Threads,  ,  Duration : 89421 msecs.
2, 3, 4, 5, 8, 11, 12, 13, 14, 15, 16, 17, 18, 21, 23, 25, 27, 31, 32, 33,
4Threads,  ,  Duration : 32307 msecs.
2, 4, 11, 12, 15, 19, 20, 22, 25, 28, 33, 34, 36, 37, 38, 40, 44, 47, 50, 55,
8Threads,  ,  Duration : 14592 msecs.
1, 3, 4, 5, 6, 7, 10, 11, 12, 14, 17, 20, 21, 22, 23, 24, 25, 26, 29, 30,
16Threads,  ,  Duration : 12877 msecs.
5, 6, 7, 9, 13, 15, 16, 17, 18, 19, 20, 22, 25, 29, 30, 31, 41, 43, 46, 48,
32Threads,  ,  Duration : 22362 msecs.
3, 5, 8, 10, 11, 12, 15, 16, 19, 21, 22, 23, 25, 26, 29, 30, 31, 32, 35, 38,
64Threads,  ,  Duration : 20262 msecs.
nhjung@GameServer32:~/CX$
```

만능 머신

- NUMA로 인한 성능 저하 측정
  - 객체 재사용. Remote Memory Access 최소화



```
nhjung@GameServer32:~/CX$ numactl -l -C 7-15 ./lfunsklist2
2, 5, 6, 8, 10, 11, 12, 13, 25, 26, 28, 30, 34, 40, 41, 47, 49, 50, 51, 54,
1Threads,  ,  Duration : 172147 msecs.
2, 3, 6, 8, 11, 12, 13, 15, 19, 20, 23, 24, 25, 26, 27, 28, 31, 36, 40, 41,
2Threads,  ,  Duration : 86157 msecs.
1, 3, 4, 5, 6, 9, 10, 11, 13, 14, 16, 17, 24, 25, 28, 31, 32, 36, 38, 39,
4Threads,  ,  Duration : 29996 msecs.
^[[B1, 2, 3, 10, 11, 12, 13, 15, 18, 31, 35, 37, 40, 41, 42, 45, 50, 54, 56, 57,
8Threads,  ,  Duration : 12923 msecs.
1, 3, 7, 9, 10, 11, 12, 13, 14, 15, 18, 20, 21, 22, 23, 24, 25, 29, 36, 37,
16Threads,  ,  Duration : 7627 msecs.
1, 3, 4, 6, 7, 8, 10, 11, 15, 16, 18, 19, 21, 23, 24, 25, 26, 28, 29, 32,
32Threads,  ,  Duration : 5522 msecs.
2, 3, 13, 14, 15, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
64Threads,  ,  Duration : 5831 msecs.
nhjung@GameServer32:~/CX$ 
```

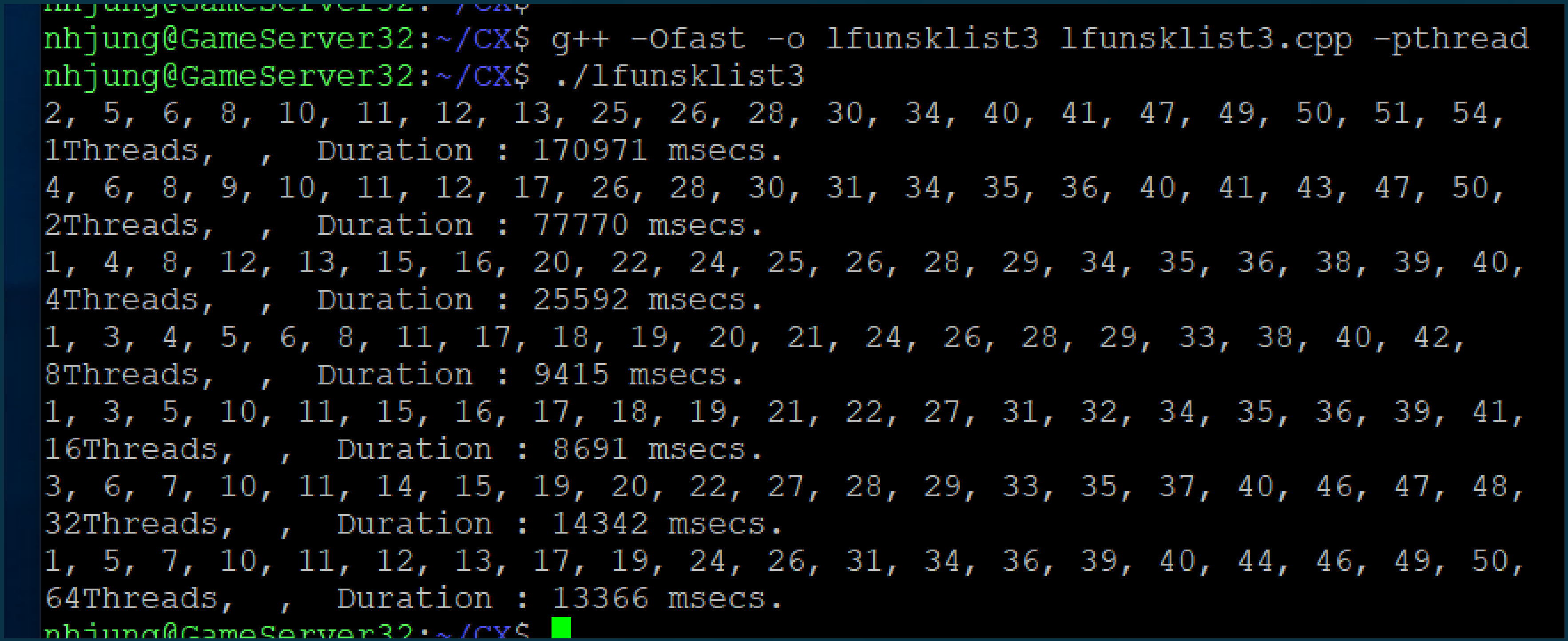
만능 머신

- 최적화
  - Read Only Method를 invocation list에서 제외

```
Response ROApply(const Invocation& invoc) {
   NODE* before = GetMaxNODE();

   while (nullptr != local_tail->next) {
      if (before == local_tail) break;
      local_tail = local_tail->next;
      local_object.Apply(local_tail->invoc);
   }
   return local_object.Apply(invoc);
}
```

만능 머신

- 최적화
  - Read Only Method를 invocation list에서 제외



```
nhjung@GameServer32:~/CX$
nhjung@GameServer32:~/CX$ g++ -Ofast -o lfunsklist3 lfunsklist3.cpp -pthread
nhjung@GameServer32:~/CX$ ./lfunsklist3
2, 5, 6, 8, 10, 11, 12, 13, 25, 26, 28, 30, 34, 40, 41, 47, 49, 50, 51, 54,
1Threads,  ,  Duration : 170971 msecs.
4, 6, 8, 9, 10, 11, 12, 17, 26, 28, 30, 31, 34, 35, 36, 40, 41, 43, 47, 50,
2Threads,  ,  Duration : 77770 msecs.
1, 4, 8, 12, 13, 15, 16, 20, 22, 24, 25, 26, 28, 29, 34, 35, 36, 38, 39, 40,
4Threads,  ,  Duration : 25592 msecs.
1, 3, 4, 5, 6, 8, 11, 17, 18, 19, 20, 21, 24, 26, 28, 29, 33, 38, 40, 42,
8Threads,  ,  Duration : 9415 msecs.
1, 3, 5, 10, 11, 15, 16, 17, 18, 19, 21, 22, 27, 31, 32, 34, 35, 36, 39, 41,
16Threads,  ,  Duration : 8691 msecs.
3, 6, 7, 10, 11, 14, 15, 19, 20, 22, 27, 28, 29, 33, 35, 37, 40, 46, 47, 48,
32Threads,  ,  Duration : 14342 msecs.
1, 5, 7, 10, 11, 12, 13, 17, 19, 24, 26, 31, 34, 36, 39, 40, 44, 46, 49, 50,
64Threads,  ,  Duration : 13366 msecs.
nhjung@GameServer32:~/CX$
```

CX

- 최적화
  - 객체 마다 Object를 갖고 있지 말고 Pool로 관리하자.
    - 다른 쓰레드에서 Update한 객체를 가져다 사용하자.
      - 모든 객체가 모든 업데이트를 할 필요가 없음.
    - Read Only Method를 실행한다면 Object를 공유 할 수 있다.
  - State
    - 0 : FREE
    - 1 : EXCLUSIVE ACCESS
    - 2 : SHARED ACCESS => 01bit : state, 2-7bit : share count

7주차 수업

- 지금 까지
  - Univesal Lockfree Skiplist의 제작
  - 최적화
    - Read Only Method는 History에 넣지 않기
    - 쓰레드별로 가장 최근에 업데이트한 Skiplist객체 유지하기
- ToDo : CX 알고리즘 구현 – 1단계
  - Update되는 SkipList객체를 thread별로 따로 두지 않고 Pool에서 관리
    - thread별로 따로 두는 것은 병렬성이 0%임

CX

- 자료구조
  - local_objects에 객체 Pool구성
  - MAX_THREAD만큼 존재하는 이유는 최악의 경우 대비
    - 모든 쓰레드가 Add를 동시에 호출하는 경우

```
atomic_int object_state[MAX_THREAD];
SeqObject local_objects[MAX_THREAD];
NODE* local_tail[MAX_THREAD];
```

CX

- 알고리즘

```
Response Apply(const Invocation& invoc) {
   NODE* prefer = new NODE{ invoc };
   Node History Update();

   index = Get_FREE_Object();

   Update_Object(I, prefer);
   local_tail[index] = curr;
   Response res = local_objects[index].Apply(curr->invoc);
   object_state[index] = ST_FREE;
   return res;
}
```

CX

- free Object를 찾는 알고리즘
  - 찾은 객체가 너무 최신인 경우를 제외해야 한다.

```
int index = 0;
while (true) {
   while (ST_FREE != object_state[index]) {
      index++;
      index = index % MAX_THREAD;
   }
   int old_state = ST_FREE;
   if (true == atomic_compare_exchange_strong(
                 &object_state[index], &old_state, ST_EXCLUSIVE)) {
      if (prefer->seq > local_tail[index]->seq) break;
      object_state[index] = ST_FREE;
   }
   index++;
   index = index % MAX_THREAD;
}
```

너무 최신이면 그냥 끝내면 되지 않는가? => res를 저장해야 한다.

CX

- ReadOnly Method의 경우
  - 우선 공유 가능한 Object가 있는지 검사
    - shared 상태이면서 seq가 before보다 최신이어야 함
  - Shared Counter를 관리해야 한다.

```
int index = 0;
for (index = 0; index < MAX_THREAD; ++index) {
   int old_state = object_state[index];
   if (ST_SHARE != (old_state & 0x3)) continue;
   if (before->seq > local_tail[index]->seq) continue;
   if (true == ACES(&object_state[index], &old_state, old_state + 4)) {
      Response res = local_objects[index].Apply(invoc);
      old_state = object_state[index];
      while (true) {
         if ((ST_SHARE + 4) == old_state) {
            if (true == ACES(&object_state[index], &old_state, ST_FREE))
               return res;
         } else
            if (true == ACES(&object_state[index], &old_state, old_state - 4)
               return res;
      }
   }
   else index--;
}
```

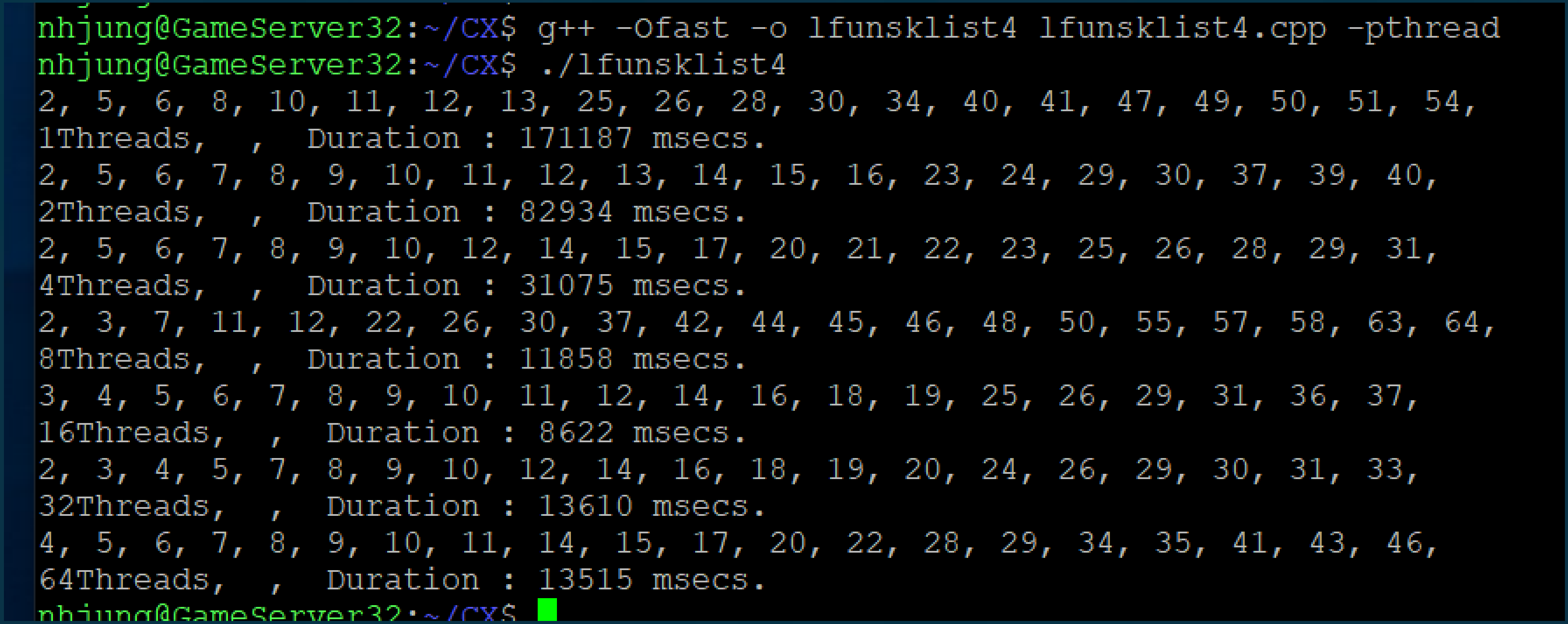
CX

- ReadOnly Method의 경우
  - 공유 가능한 객체가 없으면 FREE 객체를 사용
  - Update 해야 하므로 Exclusive 로 상태 변경

```
index = 0;
while (true) {
    while (ST_FREE != object_state[index]) {
        index++;
        index = index % MAX_THREAD;
    }
    int old_state = ST_FREE;
    if (true == ACES(&object_state[index], &old_state, ST_EXCLUSIVE))
        break;
}
```

만능 머신

- 최적화 결과
  - 별 차이 없음.



```
nhjung@GameServer32:~/CX$ g++ -Ofast -o lfunsklist4 lfunsklist4.cpp -pthread
nhjung@GameServer32:~/CX$ ./lfunsklist4
2, 5, 6, 8, 10, 11, 12, 13, 25, 26, 28, 30, 34, 40, 41, 47, 49, 50, 51, 54,
1Threads,  ,  Duration : 171187 msecs.
2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 23, 24, 29, 30, 37, 39, 40,
2Threads,  ,  Duration : 82934 msecs.
2, 5, 6, 7, 8, 9, 10, 12, 14, 15, 17, 20, 21, 22, 23, 25, 26, 28, 29, 31,
4Threads,  ,  Duration : 31075 msecs.
2, 3, 7, 11, 12, 22, 26, 30, 37, 42, 44, 45, 46, 48, 50, 55, 57, 58, 63, 64,
8Threads,  ,  Duration : 11858 msecs.
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 19, 25, 26, 29, 31, 36, 37,
16Threads,  ,  Duration : 8622 msecs.
2, 3, 4, 5, 7, 8, 9, 10, 12, 14, 16, 18, 19, 20, 24, 26, 29, 30, 31, 33,
32Threads,  ,  Duration : 13610 msecs.
4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 17, 20, 22, 28, 29, 34, 35, 41, 43, 46,
64Threads,  ,  Duration : 13515 msecs.
nhjung@GameServer32:~/CX$
```

만능 머신

- 최적화 결과
  - 별 차이 없음.

	LF	Universal (1/100 work)	Universal-Opt	Universal-Opt-Pool	
1	726	107393	170971	171187	
2	713	88679	77770	82934	
4	555	43106	25592	31075	
8	115	18109	9415	11858	
16	152	7763	8691	8622	
32	149	4707	14342	13610	
64	182	3278	13366	13515	

CX

- 분석
  - N개의 객체가 있다. 객체의 개수가 줄지 않았다.
    - 작업량이 줄지 않았다.
  - 오래된 객체를 최신 객체로 업데이트 할 때 부하가 크다.
    - Skiplist의 문제
    - 최신 객체를 Copy하는 것과 수 많은 Invocation을 적용하는 것의 비용차이 계산이 필요.
      - 효율적인 Copy 메소드 구현 필요.
      - 노드의 개수가 만 단위가 넘는다면?
    - Range를 1000이 아닌 다른 숫자들을 해봐야 한다.

CX

- CX최적화
  - ROApply 최적화
    - Apply에서 Update된 최신 Object를 CurObjectIdx가 가리키도록 한다.
    - ROApply는 CurObject를 사용하면 된다.
  - Apply 변경
    - 수행 후 CurObjectIdx를 변경
  - ABA 방지를 위해 seq와 idx의 합성을 사용.

CX

- CX최적화

```
Response ROApply(const Invocation& invoc) {
   int idx;
   int old_state;
   while (true) {
      idx = cur_obj & 0x7f;
      old_state = object_state[idx];
      if (ST_SHARE == (old_state & 0x3))
         if (true == ACES(&object_state[idx], &old_state, old_state + 4)) break;
      if (ST_FREE == (old_state & 0x3))
         if (true == ACES(&object_state[idx], &old_state, ST_SHARE + 4)) break;
   }
   Response res = local_objects[idx].Apply(invoc);
   old_state = object_state[idx];
   while (true) {
      if (ST_SHARE + 4 == old_state) {
         if (true == ACES(&object_state[idx], &old_state, ST_FREE)) break;
      } else
         if (true == ACES(&object_state[idx], &old_state, old_state - 4)) break;
   }
   return res;
}
```

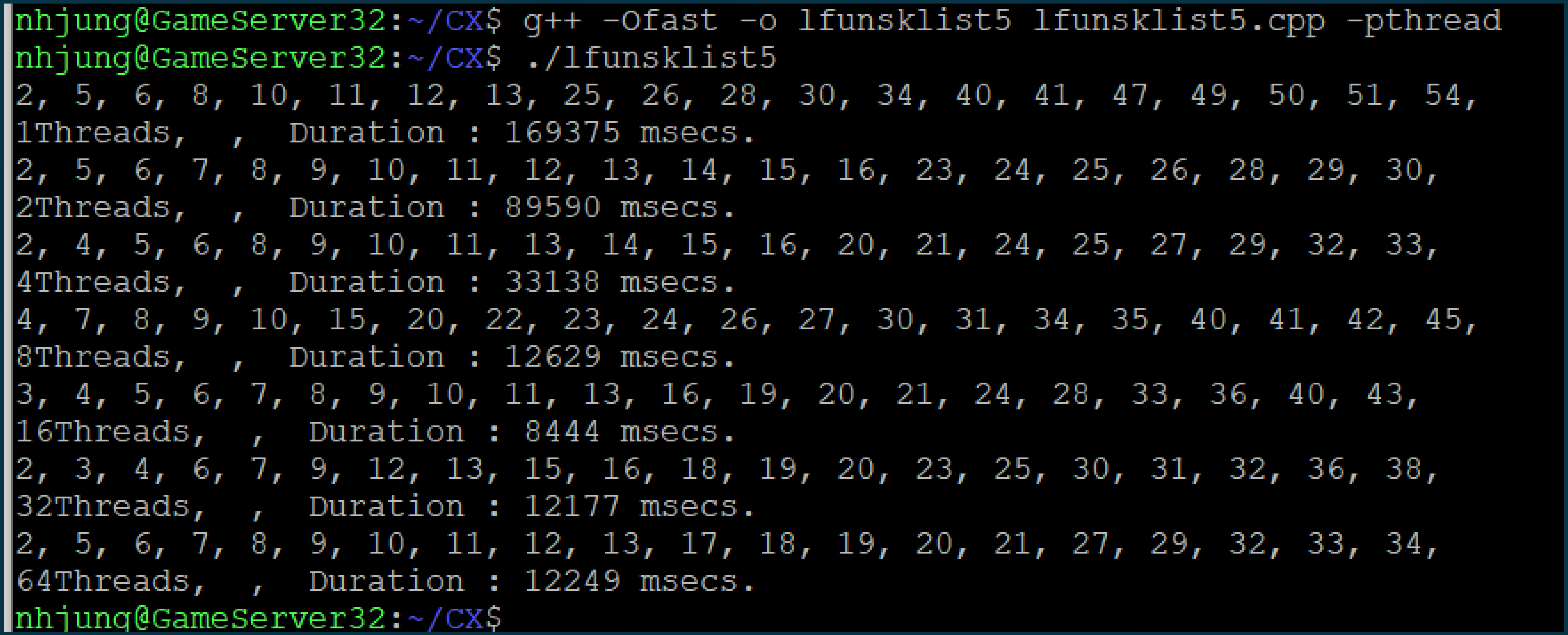
CX

- CX최적화

```
Response Apply(const Invocation& invoc) {
   NODE* prefer = new NODE{ invoc };
   // Add Prefer
   int index = 0;
   while (true) {
      while (ST_FREE != object_state[index])
         index = (index + 1) % MAX_THREAD;
      int old_state = ST_FREE;
      if (true == ACES(&object_state[index], &old_state, ST_EXCLUSIVE)) {
         if (prefer->seq > local_tail[index]->seq) break;
         object_state[index] = ST_FREE;
      }
      index = (index + 1) % MAX_THREAD;
   }
   // Advance object
   while (true) {
      long long new_obj = (prefer->seq << 7) + index;
      long long old_obj = cur_obj;
      if ((new_obj >> 7) < (old_obj >> 7)) break;
      if (true == ACES(&cur_obj, &old_obj, new_obj)) break;
   }
   return res;
}
```

CX

- 성능



```
nhjung@GameServer32:~/CX$ g++ -Ofast -o lfunsklist5 lfunsklist5.cpp -pthread
nhjung@GameServer32:~/CX$ ./lfunsklist5
2, 5, 6, 8, 10, 11, 12, 13, 25, 26, 28, 30, 34, 40, 41, 47, 49, 50, 51, 54,
1Threads,  ,  Duration : 169375 msecs.
2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 23, 24, 25, 26, 28, 29, 30,
2Threads,  ,  Duration : 89590 msecs.
2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15, 16, 20, 21, 24, 25, 27, 29, 32, 33,
4Threads,  ,  Duration : 33138 msecs.
4, 7, 8, 9, 10, 15, 20, 22, 23, 24, 26, 27, 30, 31, 34, 35, 40, 41, 42, 45,
8Threads,  ,  Duration : 12629 msecs.
3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 16, 19, 20, 21, 24, 28, 33, 36, 40, 43,
16Threads,  ,  Duration : 8444 msecs.
2, 3, 4, 6, 7, 9, 12, 13, 15, 16, 18, 19, 20, 23, 25, 30, 31, 32, 36, 38,
32Threads,  ,  Duration : 12177 msecs.
2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17, 18, 19, 20, 21, 27, 29, 32, 33, 34,
64Threads,  ,  Duration : 12249 msecs.
nhjung@GameServer32:~/CX$
```

만능 머신

- 최적화 결과
  - 별 차이 없음.

	LF	Universal (1/100 work)	Universal-Opt	Universal-Opt-Pool	CX-1
1	726	107393	170971	171187	169375
2	713	88679	77770	82934	89590
4	555	43106	25592	31075	33138
8	115	18109	9415	11858	12629
16	152	7763	8691	8622	8444
32	149	4707	14342	13610	12177
64	182	3278	13366	13515	12249

CX with NUMA

- Combined Object를 NUMA node별로 별도 관리

15(7)주차 과제

- CX알고리즘를 사용하여 구현한 Lock-Free Skiplist의 성능 개선
  - 제출 E-Class에 제출
    - Combine Object의 Pool을 NUMA 노드별로 따로 관리해서, NUMA에서의 성능향상을 측정하시오.
    - 소스코드, 성능 벤치마크 결과(초당 Operation수, 메모리 사용량, Contains의 비율 변화), 구현기법 설명, 성능 분석
    - 지금 부터는 벤치마크를 NUMA machine에서 함.
      - 계정이 없는 수강생은 nhjung골뱅이kpu.ac.kr로 id/passwd를 적어서 신청할 것
  - 기한
    - 7월 1일 수요일 오후 1시까지.

최적화 한번더

- free Object를 찾는 알고리즘
  - 찾은 객체가 너무 최신인 경우를 제외해야 한다.

```
int index = 0;
while (true) {
   while (ST_FREE != object_state[index]) {
      index++;
      index = index % MAX_THREAD;
   }
   int old_state = ST_FREE;
   if (true == atomic_compare_exchange_strong(
                 &object_state[index], &old_state, ST_EXCLUSIVE)) {
      if (prefer->seq > local_tail[index]->seq) break;
      object_state[index] = ST_FREE;
   }
   index++;
   index = index % MAX_THREAD;
}
```

너무 최신이면 그냥 끝내면 되지 않는가? => res를 저장해야 한다.

최적화 한번더

```
class NODE
{
public:
        Invocation invoc;
        Consensus decideNext;
        NODE* next;
        volatile int seq;
        atomic_bool res;
```

```
NODE* curr = local_tail[index]->next;
distance_count[prefer->seq - curr->seq]++;
while (curr != prefer) {
        curr->res = local_objects[index].Apply(curr->invoc);
        curr = curr->next;
}
local_tail[index] = curr;
Response res = local_objects[index].Apply(curr->invoc);
prefer->res = res;
object_state[index] = ST_FREE;
```

최적화 한번더

```
int index = 0;
while (true) {
   while (ST_FREE != object_state[index]) {
      index++;
      index = index % MAX_THREAD;
   }
   int old_state = ST_FREE;
   if (true == atomic_compare_exchange_strong(
                 &object_state[index], &old_state, ST_EXCLUSIVE)) {
      if (prefer->seq > local_tail[index]->seq) break;
      object_state[index] = ST_FREE;
      return prefer->res;
   }
   index++;
   index = index % MAX_THREAD;
}
```

최적화 한번더

- 결과 : 별 차이 없음

NUMA 구현

- NUMA 자료 구조

```
struct NUMA_DATA {
        atomic_int object_state[MAX_THREAD];
        SeqObject local_objects[MAX_THREAD];
        NODE* local_tail[MAX_THREAD];
        atomic <long> cur_obj;
};

NUMA_DATA* numa_data[NUM_NUMA_NODES];
```

```
thread_local int thread_id;
thread_local int numa_id;
```

```
int get_numa_node_from_cpu(int i)
{
        i = i % 32;
        return i / 8;
}
```

NUMA 구현

- NUMA 자료 구조 할당

```
LFUniversal()
{
   tail = new NODE;
   tail->seq = 1;

   for (int i = 0; i < NUM_NUMA_NODES; ++i) {
      numa_data[i] = reinterpret_cast<NUMA_DATA *>(
                        numa_alloc_onnode(sizeof(NUMA_DATA), i));
   }
   for (int i = 0; i < MAX_THREAD; ++i) {
      head[i] = tail;
        int nu_node = get_numa_node_from_cpu(i);
        numa_data[nu_node]->object_state[i] = ST_FREE;
        numa_data[nu_node]->local_objects[i].Reset();
        numa_data[nu_node]->local_tail[i] = tail;
   }
   for (int i =0; i< NUM_NUMA_NODES;++i)
      numa_data[i]->cur_obj = (1 << 7) + get_first_cpu_from_node(i);
```

NUMA 구현

- 주의!

```
struct NUMA_DATA {
        atomic_int object_state[MAX_THREAD];
        SeqObject local_objects[MAX_THREAD];
        NODE* local_tail[MAX_THREAD];
        atomic <long> cur_obj;
};

NUMA_DATA* numa_data[NUM_NUMA_NODES];
```

```
numa_data[i] = reinterpret_cast<NUMA_DATA *>(
                        numa_alloc_onnode(sizeof(NUMA_DATA), i));
```

– SeqObject의 생성자가 호출되지 않는다.

NUMA 구현

- NUMA 자료 구조 접근

```
Response ROApply(const Invocation& invoc) {
        NUMA_DATA* my_node = numa_data[numa_id];
        int idx;
        int old_state;
        while (true) {
                idx = my_node->cur_obj & 0x7f;
                old_state = my_node->object_state[idx];
```

NUMA 구현

- Thread 초기화

```
void ThreadFunc(int num_thread, int tid)
{
        thread_id = tid;
        numa_id = get_numa_node_from_cpu(tid);
        run_on_cpu(tid);
…
```

NUMA 구현

- 성능 비교

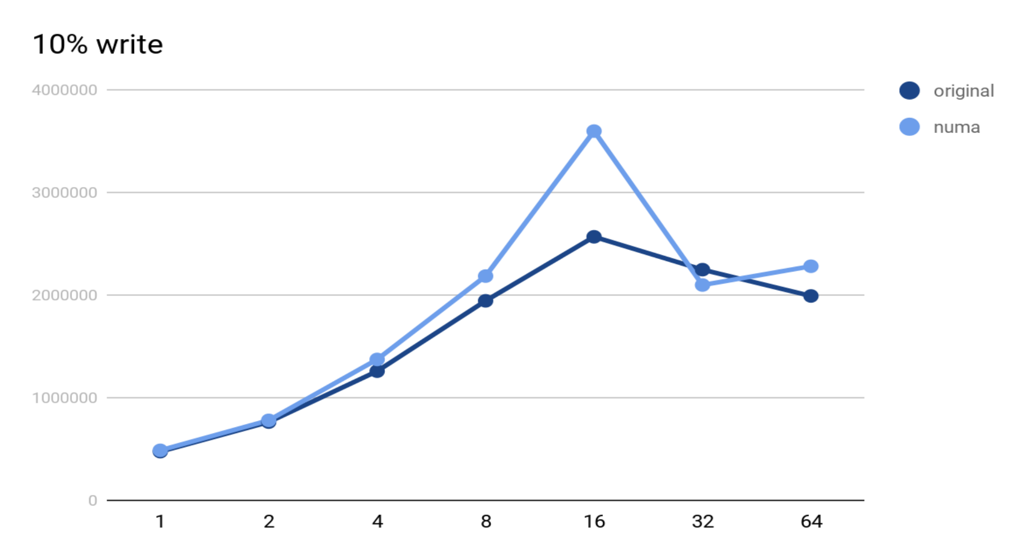
	LF	Universal-Opt	Universal-Opt-Pool	CX-1	CX_NUMA
1	726	170971	171187	169375	166375
2	713	77770	82934	89590	88600
4	555	25592	31075	33138	30728
8	115	9415	11858	12629	12286
16	152	8691	8622	8444	7202
32	149	14342	13610	12177	14701
64	182	13366	13515	12249	13146

NUMA 구현

- 성능 분석
  - NUMA를 신경써서 객체들을 분배했는데, 성능이 별로 좋아지지 않았다.
  - 원인?
    - Invocation List가 계속 Access되는데 이것의 NUMA Remote접근이 많아서 Object를 독립시키는 것으로는 부족
      - Invocation List를 Numa Node에 복제해야 하지 않는가? (논문 아이디어??) , Remote Memory Access와 복사 비용의 trade-off
    - 프로그램에 무엇인가 오류가 있어서 Object의 Node독립이 잘 되지 않는다.
      - 검증 필요. “NumaMMA: NUMA MeMory Analyzer”논문의 Tool을 가져와서 테스트 해보자.

CJY 과제 리뷰

- 성능 분석
  - Contains의 비율을 90%로 늘렸을 때 성능향상이 16thread에서 발생
    - 하지만 기대했던 32, 64thread에서는 성능향싱이 없음
    - 왜일까??



SSY 과제 리뷰

- 프로그램 오류
  - 최적화를 위해서 cur_obj 포인터를 사용해서 최신 객체에 대해 RO_Apply를 적용 시켰는데,
  - NUMA 노드별로 별도의 cur_obj가 존재해서 cur_obj가 아주 옛날 객체를 가리킬 수 있고, 이는 메모리 일관성을 어긴다.
  - 대책
    - RO_Apply에서 현재 seq를 Invocation list에서 얻은 후 그 seq보다 최신인 객체가 없으면, Free 객체를 하나 얻어서 Update한다.

NUMA 구현

- 성능 비교

	LF	Universal-Opt	CX-1	CX_NUMA	CX_NUMA_FIX
1	726	170971	169375	166375	
2	713	77770	89590	88600	
4	555	25592	33138	30728	
8	115	9415	12629	12286	10687
16	152	8691	8444	7202	7270
32	149	14342	12177	14701	14343
64	182	13366	12249	13146	13295

다음 주

- “NumaMMA: NUMA MeMory Analyzer”논문의 Tool을 가져와서 테스트 해보자.
- 한학기 동안 다루었던 주제들 리뷰.