NUMA - 2

산업용병렬처리특론

정내훈

2020년도 1학기

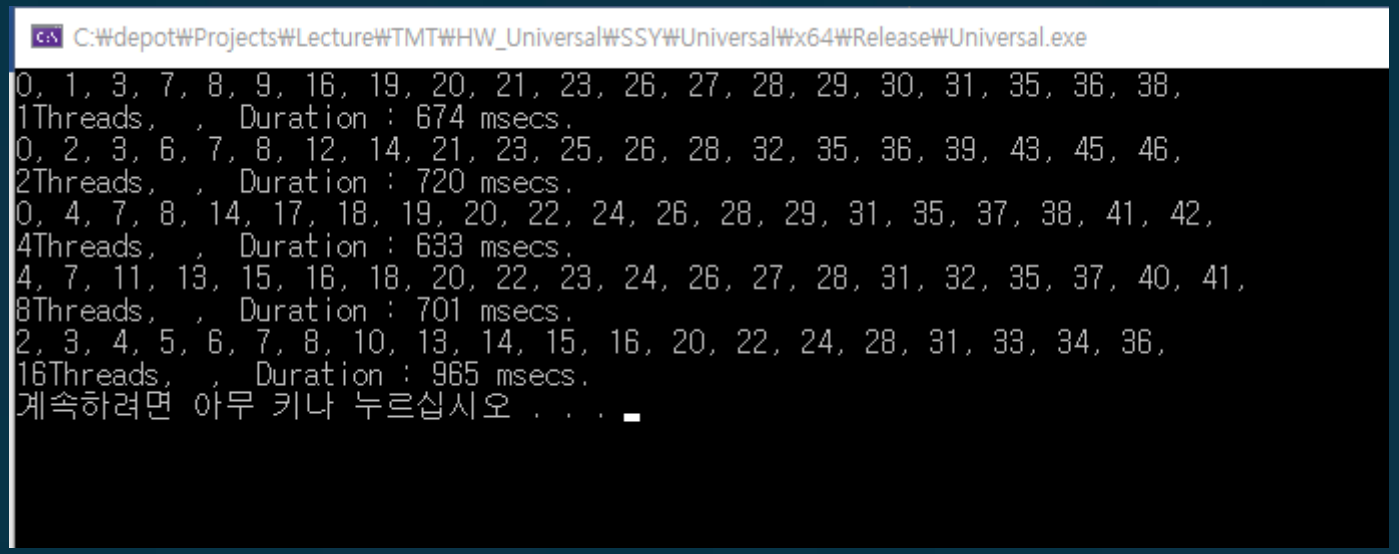
한국산업기술대학교 스마트팩토리융합학과

목차

- 과제리뷰
- NUMA 구현 환경
- NUMA Aware Programming
- 5차 과제

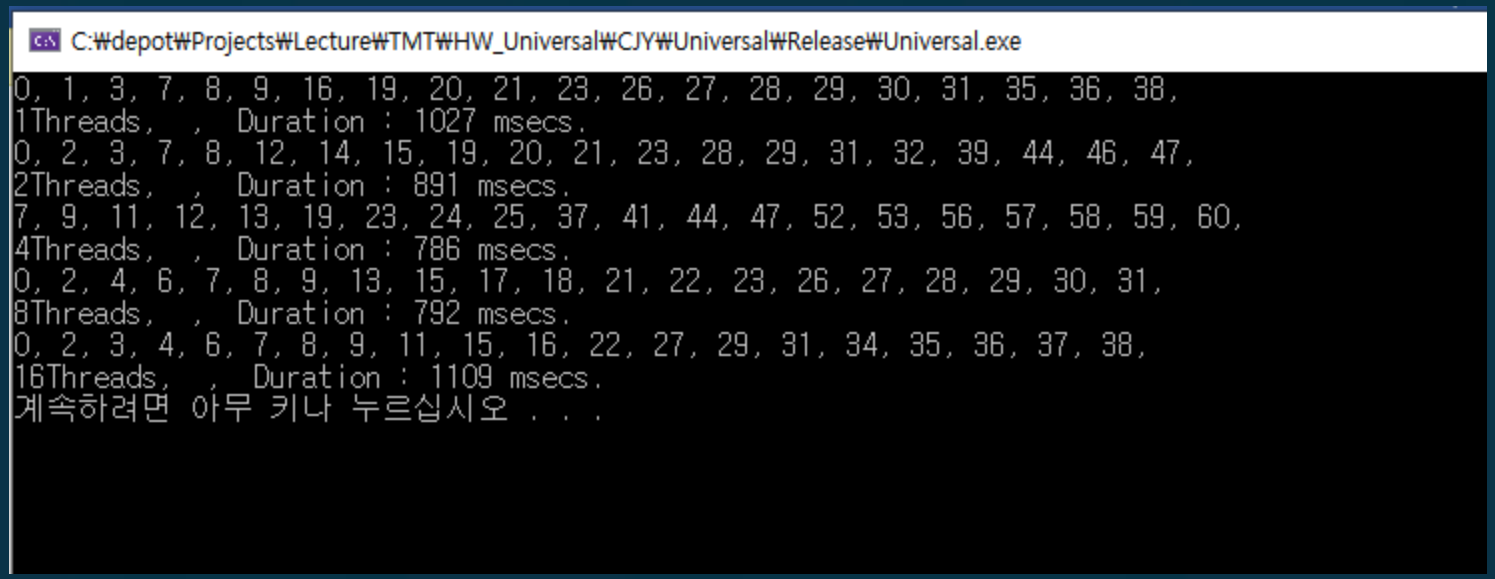
SSY

- read only 제거
- thread 마다 로컬 객체 재사용
- 메모리 재사용 - 미흡



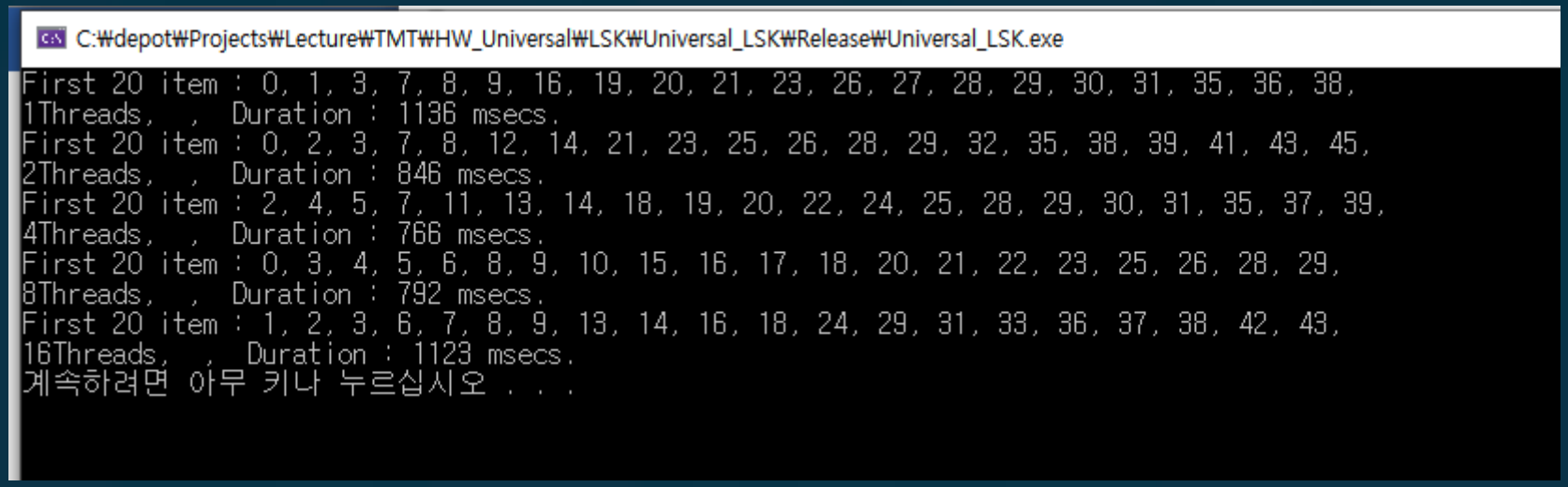
CJY

- 메모리 재사용
- local object 재사용, read_only 별도 취급



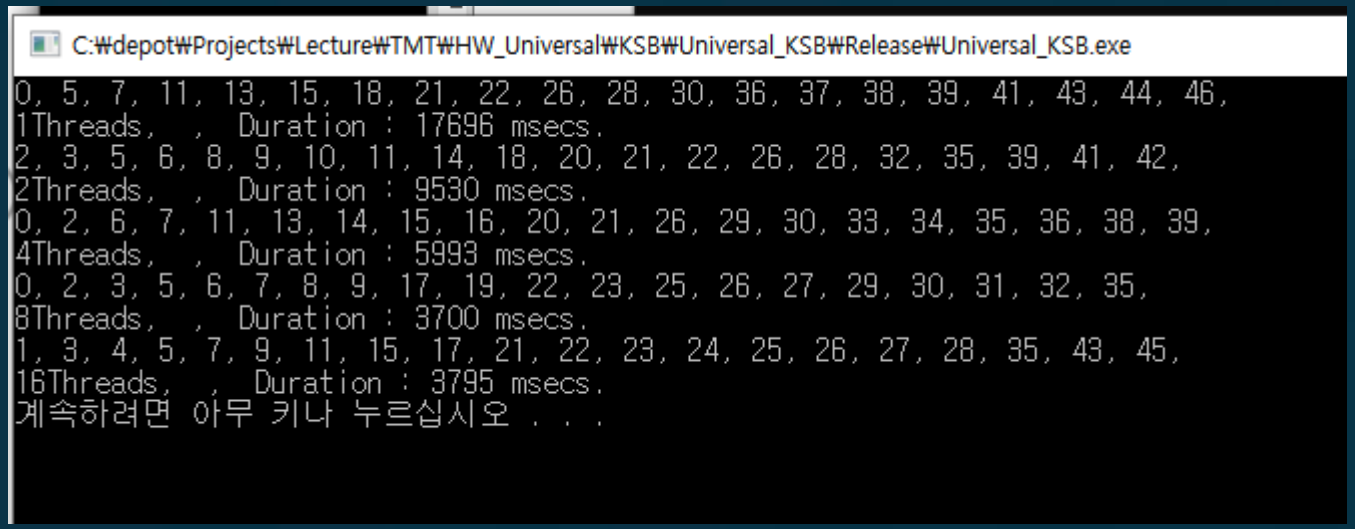
LSK

- 메모리 재사용 없음
- read only invocation 별도처리
- local object 재사용.



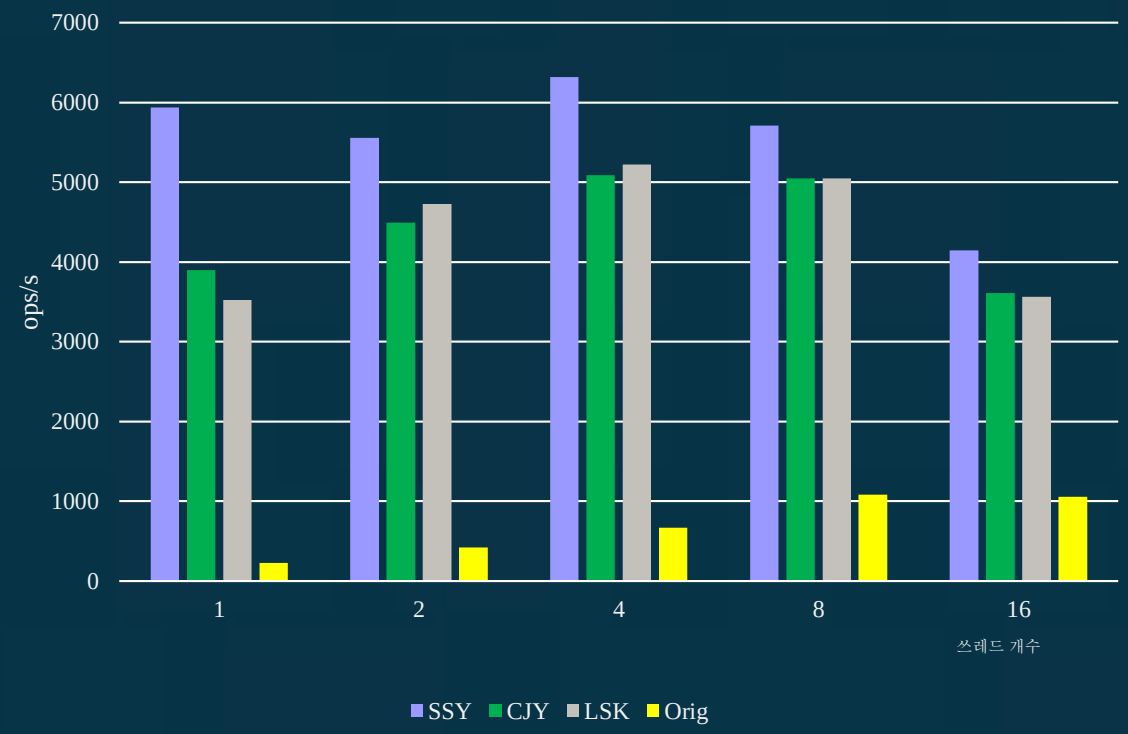
KSB

- 컴파일 오류


C:₩depot₩Projects₩Lecture₩TMT₩HW_Universal₩KSB₩Universal_KSB₩Release₩Universal_KSB.exe
0, 5, 7, 11, 13, 15, 18, 21, 22, 26, 28, 30, 36, 37, 38, 39, 41, 43, 44, 46,
1Threads, , Duration : 17696 msecs.
2, 3, 5, 6, 8, 9, 10, 11, 14, 18, 20, 21, 22, 26, 28, 32, 35, 39, 41, 42,
2Threads, , Duration : 9530 msecs.
0, 2, 6, 7, 11, 13, 14, 15, 16, 20, 21, 26, 29, 30, 33, 34, 35, 36, 38, 39,
4Threads, , Duration : 5993 msecs.
0, 2, 3, 5, 6, 7, 8, 9, 17, 19, 22, 23, 25, 26, 27, 29, 30, 31, 32, 35,
8Threads, , Duration : 3700 msecs.
1, 3, 4, 5, 7, 9, 11, 15, 17, 21, 22, 23, 24, 25, 26, 27, 28, 35, 43, 45,
16Threads, , Duration : 3795 msecs.
계속하려면 아무 키나 누르십시오 . . .

성능비교

- 컴파일 오류


7000
6000
5000
4000
3000
2000
1000
0
ops/s
1
2
4
8
16
쓰레드 개수
SSY
CJY
LSK
Orig

평가

- 최적화를 적용한 결과 성능이 많이 개선되었다.
  - 하지만 single thread version보다 나을 것이 없다.
    - 작업의 병렬성이 없다.
      - Add(734)를 호출했다면, 모든 쓰레드에서 독자적으로 Add(734)가 호출된다.
  - 멀티쓰레드 성능개선이 없다.
    - log의 next에서의 잦은 충돌 때문인 것으로 보인다.
  - 최적화 이전 버전에서 병렬성을 보였던 이유?
    - log의 next에서의 충돌로 인한 영향이 미미했다.

개선

- 병렬성 문제 해결?
  - “A Wait Free Universal Construct for Large Objects”, PPoPP '20: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel ProgrammingFebruary 2020
  - thread마다 object를 두지 않고, object의 pool을 작성하여 교대로 사용.
  - rw-lock을 두어 read-only-method는 병렬 수행

개선

- 병렬성 문제 해결?
  - “A Wait Free Universal Construct for Large Objects”, PPoPP '20: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel ProgrammingFebruary 2020
  - thread마다 object를 두지 않고, object의 pool을 작성하여 교대로 사용.
  - rw-lock을 두어 read-only-method는 병렬 수행

개선

- CX Algorithm

```
struct Combined {
   seqObject s_o;
   rwlock      lck;
   seqNode * ptr;
}

Combined combs[NUM_COM];
```

```
Apply(invocation &inv)
{
   if (is_read_only(inv)){
      max_seq = get_max_seq()
      for (auto &c : combs) {
         if (c.try_rwlock_read()) {
            if (max_seq == c.seq)
               return c.apply(inv);
         }
      }
   }
   for (auto &c : combs) {
       if (c.try_rwlock_write()) {
          combs.update();
          return c.apply(inv)
       }
    }
  }
}
```

개선

- CX Algorithm
  - log가 무한히 길어질 수 없으므로 log의 길이가 어느 정도 이상이 되면 log를 줄이고 combs들 중 ptr가 사용 불능이 되는 것들을 초기화 한다.
  - 초기화된 combs의 update()에서는 적절한 combs의 원소를 copy한 후 update()한다.
  - 이를 위해 seqObject는 deepcopy()메소드를 가지고 있어야 한다.

5주차 과제

- Universal Constructor를 사용하여 구현한 Lock-Free Skiplist의 성능 개선
  - 제출 E-Class에 제출
    - 최적화 기법들을 적용하시오
      - Invocation호출 최소화, Read Only Invocation제거
    - CX 알고리즘을 적용하시오
    - 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
    - 4월 23일 목요일 오후 1시까지.

목차

- 과제리뷰
- NUMA 구현 환경
- NUMA Aware Programming
- 5차 과제

구현

- 테스트 환경
  - LINUX, Ubuntu 18.04.1
  - gcc 7.5.0
  - Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz (family: 0x6, model: 0x2d, stepping: 0x7)
  - NUMA : 8Core X 4 CPU, Hyperthreading
  - 256GB 메모리
  - 접속방법 : ssh로 210.93.61.41에 접속
    - id요청은 nhjung골뱅이kpu.ac.kr로 (id/passwd) email을 보낼것.

구현

- 참조
  - https://lunatine.net/2016/07/14/numa-with-linux/
  - 운영체제에서의 NUMA관리가 잘 정리되어 있음.
  - https://frankdenneman.nl/2016/07/06/introduction-2016-numa-deep-dive-series/
  - 여러가지 많은 자료.
  - https://www.youtube.com/watch?v=YXYw8I8ZVYw
    - 유료라서 시청불가

구현

- LINUX
  - NUMA 컴퓨터에서 운영체제가 실행되는 것에 더해서 여러가지 기능들이 추가됨
    - Kernel 2.5부터 NUMA기능들이 들어가기 시작해서 지금은 default로 NUMA 메모리 관리 기능이 지원됨.

```
nhjung@GameServer32:~$ sysctl kernel.numa_balancing
kernel.numa_balancing = 1
```

    - 메모리 할당이 프로세스가 실행되는 노드에서 이루어지고
    - 어떤 노드에 부하가 집중되면 프로세스와 함께 메모리도 다른 노드로 자동 이전된다.

구현

- numactl
  - 현재 NUMA상태를 보여주는 명령어

```
nhjung@GameServer32:~$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
59 60 61 62 63
cpubind: 0 1 2 3
nodebind: 0 1 2 3
membind: 0 1 2 3
```

구현

- numactl
  - 현재 NUMA상태를 보여주는 명령어

```
nhjung@GameServer32:~$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
59 60 61 62 63
cpubind: 0 1 2 3
nodebind: 0 1 2 3
membind: 0 1 2 3
```

구현

- numactl
  - 현재 NUMA상태를 보여주는 명령어

```
nhjung@GameServer32:~$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 32 33 34 35 36 37 38 39
node 0 size: 64411 MB
node 0 free: 63439 MB
node 1 cpus: 8 9 10 11 12 13 14 15 40 41 42 43 44 45 46 47
node 1 size: 64508 MB
node 1 free: 64306 MB
node 2 cpus: 16 17 18 19 20 21 22 23 48 49 50 51 52 53 54 55
node 2 size: 64508 MB
node 2 free: 64298 MB
node 3 cpus: 24 25 26 27 28 29 30 31 56 57 58 59 60 61 62 63
node 3 size: 64483 MB
node 3 free: 63520 MB
node distances:
node   0   1   2   3
  0:  10  21  30  21
  1:  21  10  21  30
  2:  30  21  10  21
  3:  21  30  21  10
```

구현

- numstats
  - 현재 NUMA상태를 보여주는 명령어

```
nhjung@GameServer32:~$ numastat
                           node0           node1           node2           node3
numa_hit                1266433          183977          154698         6209282
numa_miss                     0               0               0               0
numa_foreign                  0               0               0               0
interleave_hit            21223           21387           21206           21377
local_node              1265187          158932          130021         6186463
other_node                 1246           25045           24677           22819
```

구현

- Linux의 NUMA 기능 테스트

```
constexpr long SIZE = 32ULL * 1024 * 1024 * 1024;
struct MEM {
   char buf[SIZE];
};


int main()
{
   MEM *p = new MEM;
   while(true)
      for (long i=0;i<SIZE;i += 4096) p->buf[i] = 0;
}
```

  - 위의 프로그램을 실행 후
  - numastat –p <프로그램이름>을 실행.

Memory Latency 측정

- 인텔의 mlc(Memory Latency Checker)사용
  - https://software.intel.com/en-us/articles/intelr-memory-latency-checker
  - Machine Status Register에 접근 가능해야 한다.
    - prefetch를 disable시키기 위해
    - mlc도 당연히 root로 실행되어야 한다.

```
nhjung@GameServer32:~/NUMA/Linux$ sudo modprobe msr
nhjung@GameServer32:~/NUMA/Linux$ sudo ./mlc
```

Memory Latency 측정

- 결과

```
nhjung@GameServer32:~/NUMA/Linux$ sudo ./mlc
Intel(R) Memory Latency Checker - v3.8
Measuring idle latencies (in ns)...
                Numa node
Numa node            0       1       2       3
       0          70.6   246.0   253.8   245.5
       1         245.7    70.2   242.2   252.3
       2         253.6   244.8    70.2   244.3
       3         237.8   252.2   244.8    70.2
```

  - Remote Node의 Latency는 약 3.5배

Memory Latency 측정

- 결과

```
Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is
enabled
Using Read-only traffic type
                Numa node
Numa node            0       1       2       3
       0         28467.3  4224.8  4110.3  4180.6
       1          4130.6 27943.0  4263.4  4090.4
       2          4094.5  4197.6 28568.7  4192.9
       3          4286.8  4101.5  4156.6 28591.5
```

- Bandwidths의 차이는 7배 (Latency의 2배)

Memory Latency 측정

- 결과

```
Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency        33.1
Local Socket L2->L2 HITM latency        37.6
Remote Socket L2->L2 HITM latency (data address homed in writer socket)
                      Reader Numa Node
Writer Numa Node     0       1       2       3
            0        -   152.8   208.1   148.6
            1    150.1       -   151.9   206.7
            2    206.9   150.2       -   152.5
            3    151.4   207.5   151.2       -
Remote Socket L2->L2 HITM latency (data address homed in reader socket)
                      Reader Numa Node
Writer Numa Node     0       1       2       3
            0        -   206.1   233.4   205.2
            1    202.1       -   202.2   237.4
            2    231.4   206.8       -   205.3
            3    201.8   237.8   201.9       -
```

- HITM : modified cache line 전송
- writer node에서 reader node로 전송

목차

- 과제리뷰
- NUMA 구현 환경
- NUMA Aware Programming
- 5차 과제

libnuma

- apt-get install libnuma-dev

```
#include <numa.h>
#include <iostream>
using namespace std;

int main()
{
   if (numa_available() < 0)
      cout << "No NUMA!\n";
   else
      cout << "This is NUMA.\n";
}
```

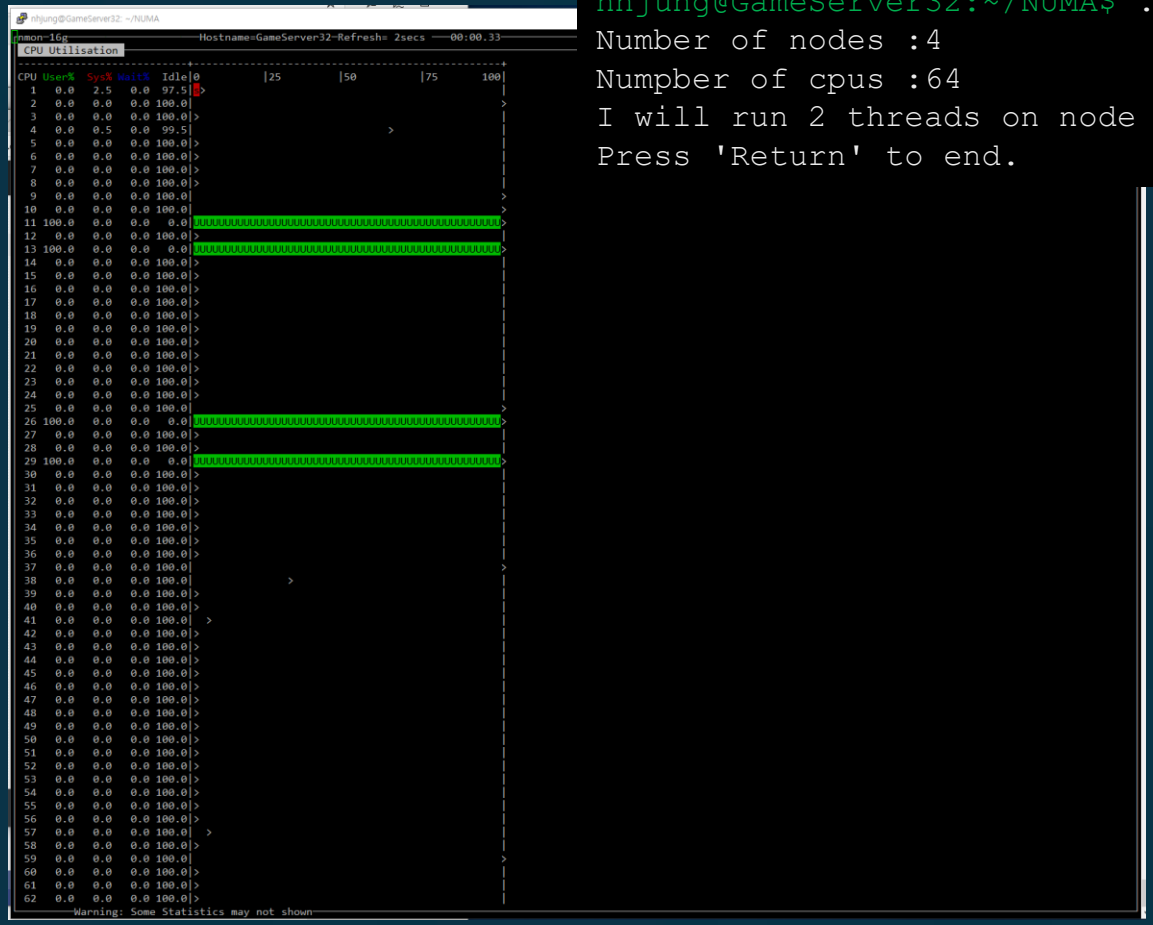
libnuma

- Pinning to node

```
 1 #include <numa.h>
 2 #include <iostream>
 3 #include <thread>
 4
 5 using namespace std;
 6
 7 void run_on_node(int node)
 8 {
 9     numa_run_on_node(node);
10     while(true);
11 }
12
13 int main()
14 {
15     cout << "Number of nodes :" << numa_num_configured_nodes() << endl;
16     cout << "Numpber of cpus :" << numa_num_configured_cpus() << endl;
17     cout << "I will run 2 threads on node #1 and #3i each\n";
18     thread t1 {run_on_node, 1};
19     thread t2 {run_on_node, 1};
20     thread t3 {run_on_node, 3};
21     thread t4 {run_on_node, 3};
22     cout << "Press \'Return\' to end." << std::endl;
23     cin.get();
24 }
```

libnuma

- 결과

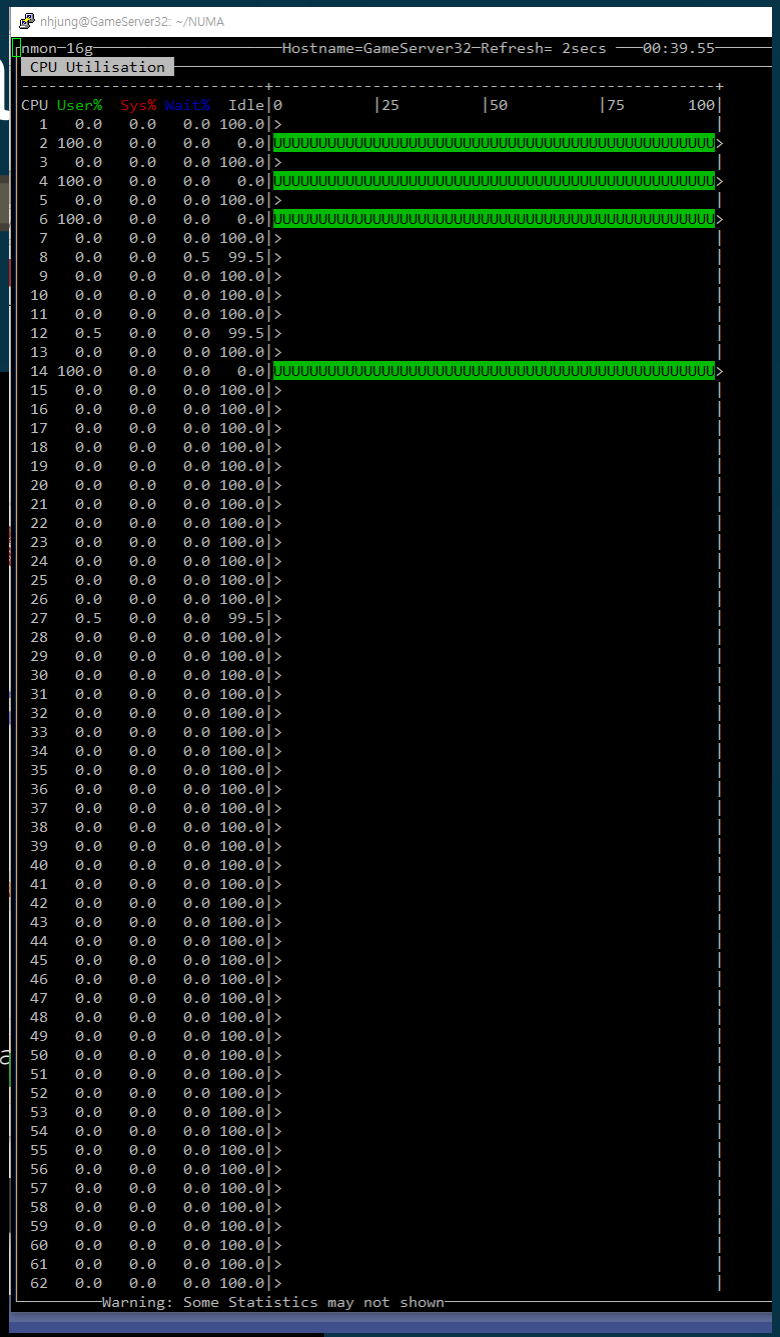



```
nhjung@GameServer32:~/NUMA$ !g++
g++ -std=c++11 -o numa_run numa_run.cpp -lnuma -pthread
nhjung@GameServer32:~/NUMA$ ./numa_run
Number of nodes :4
Numpber of cpus :64
I will run 2 threads on node #1 and #3 each
Press 'Return' to end.
```

libnuma

- Pinning to CPU

```
 1 #include <numa.h>
 2 #include <iostream>
 3 #include <thread>
 5 using namespace std;
 6
 7 void run_on_cpu(int cpuid)
 8 {
 9     struct bitmask *cpubuf;
10     cpubuf = numa_allocate_cpumask();
11     numa_bitmask_setbit(cpubuf, cpuid);
12     numa_sched_setaffinity(0, cpubuf);
13     while(true);
14     numa_free_cpumask(cpubuf);
15 }
16
17 int main()
18 {
19     cout << "I will run 4 threads on cpu #1, #3, #5 a
20     thread t1 {run_on_cpu, 1};
21     thread t2 {run_on_cpu, 3};
22     thread t3 {run_on_cpu, 5};
23     thread t4 {run_on_cpu, 13};
24     cout << "Press \'Return\' to end." << std::endl;
25     cin.get();
26 }
```



libnuma

- 메모리 할당
  - void *numa_alloc_onnode(size_t size, int node);
    - 해당 노드에 size만큼 메모리 할당
    - 실제로는 page 묶음 단위로 할당. 아래의 모든 함수에도 적용됨
    - 반드시 numa_free로 해제해야 함
  - void *numa_alloc_local(size_t size);
    - 자신의 노드에서 size만큼 할당
  - void *numa_alloc_interleaved(size_t size);
    - 여러 노드에 걸쳐 골고루 할당. page 묶음 보다 size가 작으면 한 node에서 할당
  - void numa_free(void *start, size_t size);
    - 일반 free와는 달리 size를 주어야 함.

libnuma

- 그밖에 많은 함수들이 있음.
- 자세한 것은 ‘man numa’

목차

- 과제리뷰
- NUMA 구현 환경
- NUMA Aware Programming
- 5차 과제

과제

- CX 알고리즘을 사용한 만능 머신 최적화.
  - 12페이지 참조