[MemOS/Challenges: 2nd week] NUMA-aware programming

2021. 8. 18

Baik Song An

ETRI

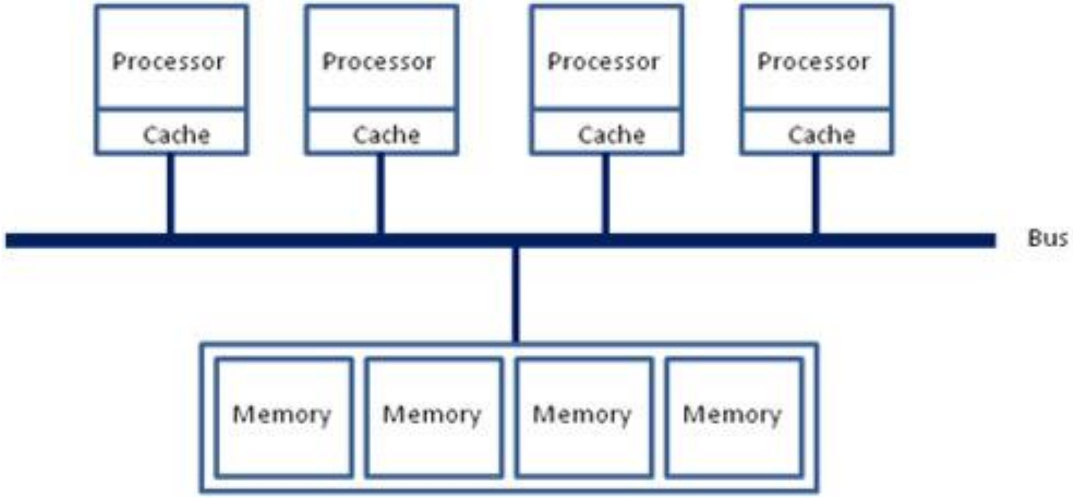
Contents

- NUMA-aware programming (recap)
- Example #1: memcpy() microbenchmark
- Example #2: PARSEC-blackscholes
- Example #3: SpMV/DMV (HPC workloads)

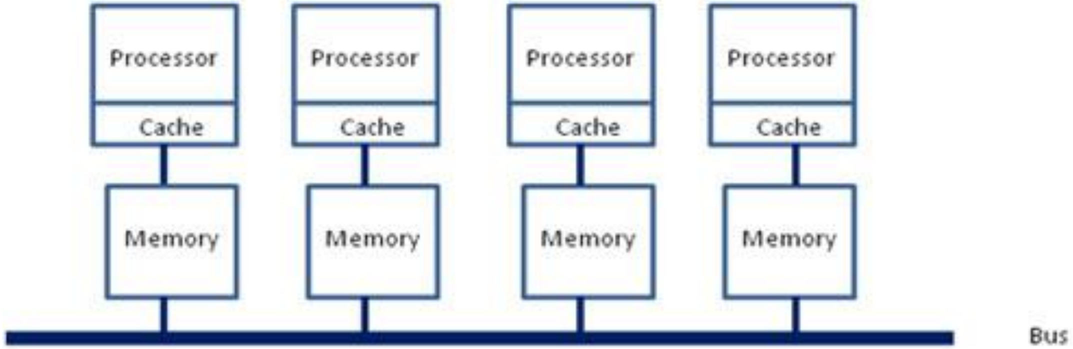
NUMA-aware programming

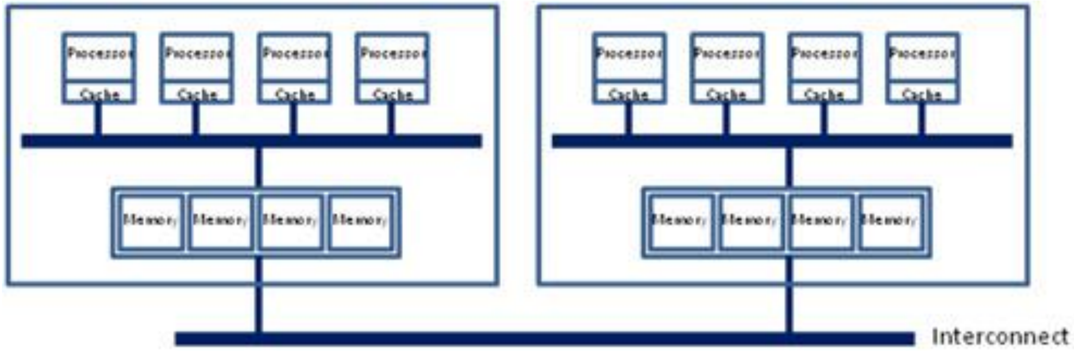
NUMA (Non-Uniform Memory Access)

- UMA



- NUMA





NUMA-aware programming

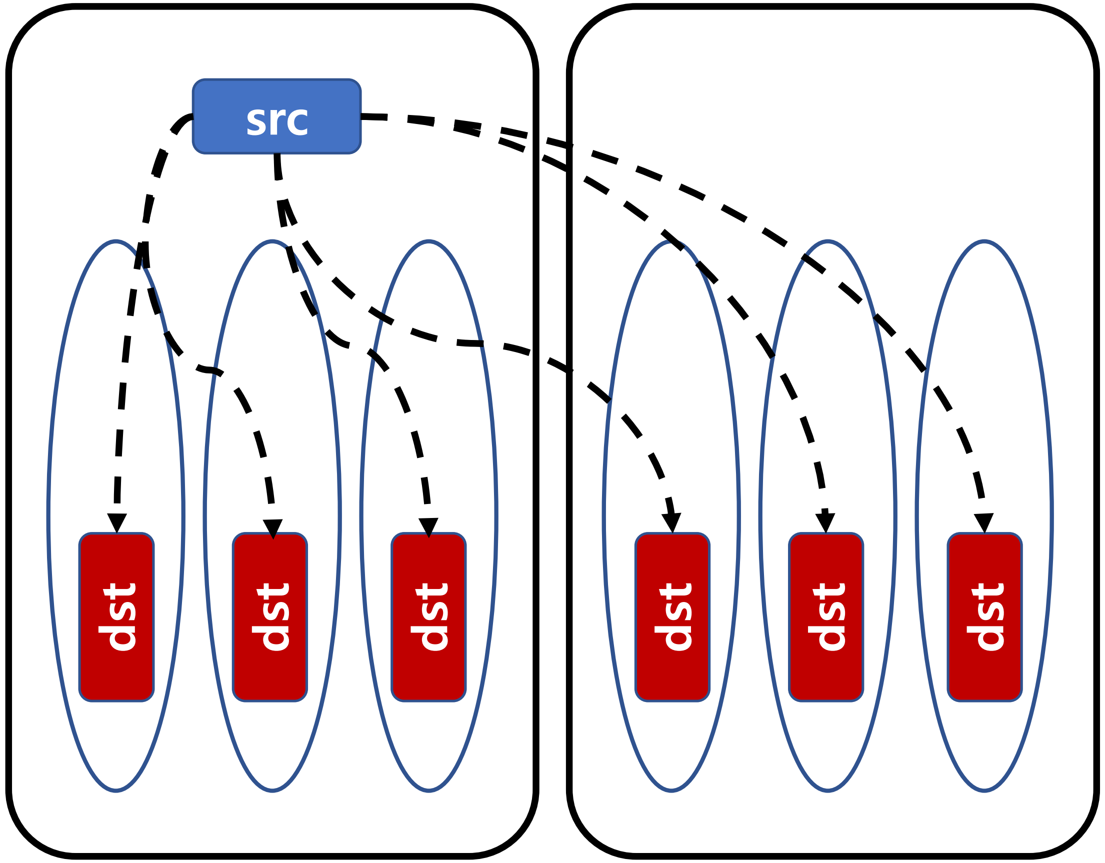
1. Processor Affinity
   - Thread migration to another NUMA node
     - Need to fetch the data from the previous node (overheads)
   - Pinning a thread to a specific core, or limiting the migration to intra-node cores

2. Data Placement with Explicit Memory Allocation Directives
   - Example) libnuma library in Linux
   - Users explicitly designate a NUMA node for memory allocation
     - numa_alloc_onnode()

Example #1: memcpy() microbench

Overview (recap)

- In-house multithread memcpy() microbenchmark
- Per-thread src -> dst memcpy
- One source in primary node is shared by all worker threads
- All src/dst memory are allocated with numa_alloc_onnode()
- Each thread is pinned to its corresponding vCPU



Modification

- Provides both NUMA-aware & non-NUMA-aware versions
  - memcpy/memcpypre: non-NUMA-aware
  - memcpy.numa/memcpypre.numa: NUMA-aware

- Difference between memcpy & memcpypre
  - memcpy: handles page faults during memcpy()
  - memcpypre: finishes all page fault handling before memcpy()

- Refer to the update source!

Example #2: PARSEC-blackscholes

Running blackscholes

1. Download source & input data

```
$ wget http://parsec.cs.princeton.edu/download/3.0/parsec-3.0.tar.gz
```

2. Compile blackscholes

```
$ tar zxvf parsec-3.0.tar.gz
$ cd parsec-3.0/bin
$ ./parsecmgmt –a build –p blackscholes –c gcc-hooks
```

3. Run blackscholes

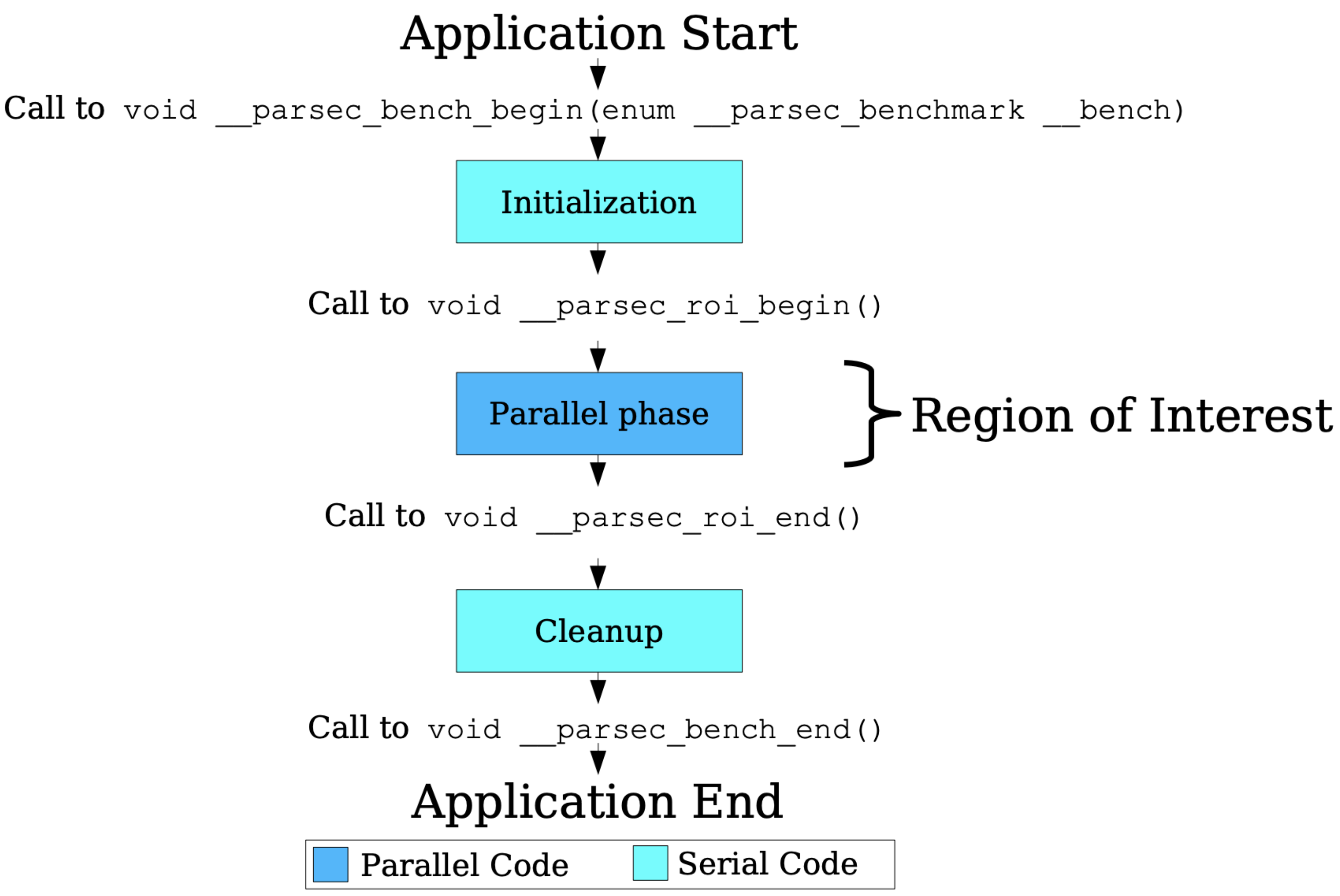
```
$ ./parsecmgmt –a run –p blackscholes –c gcc-hooks –i native –n <# of CPU cores>
```

Running blackscholes: output

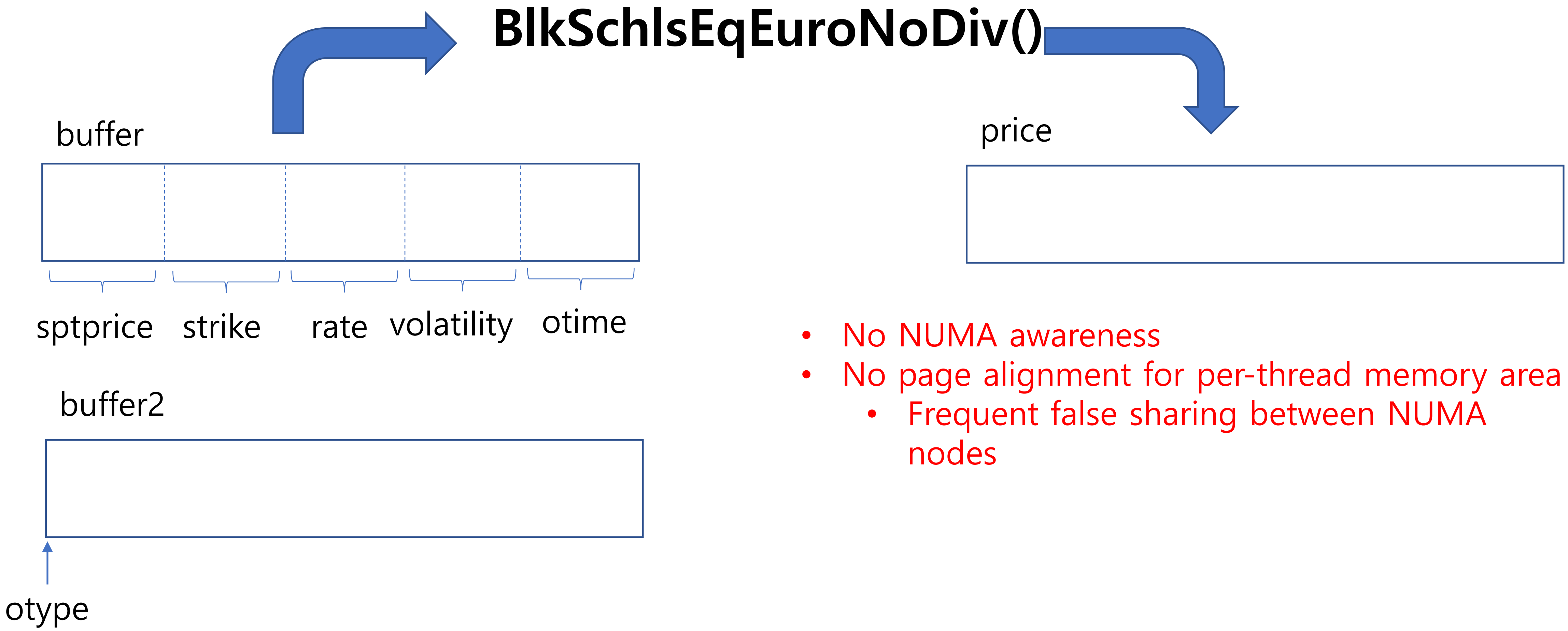
```
[PARSEC] Benchmarks to run:  parsec.blackscholes

[PARSEC] [========== Running benchmark parsec.blackscholes [1] ==========]
[PARSEC] Deleting old run directory.
[PARSEC] Setting up run directory.
[PARSEC] Unpacking benchmark input 'native'.
in_10M.txt
[PARSEC] Running 'time /home/baiksong/giantvm/github/ememos/Tutorial/Examples/parsec-3.0/bin/../pkgs/apps/blackscholes/inst/amd64-linux.gcc-hooks/bin/blackscholes 20 in_10M.txt prices.txt':
[PARSEC] [---------- Beginning of output ----------]
PARSEC Benchmark Suite Version 3.0-beta-20150206
[HOOKS] PARSEC Hooks Version 1.2
Num of Options: 10000000
Num of Runs: 100
Size of data: 400000000
[HOOKS] Entering ROI
[HOOKS] Leaving ROI
ROI: 3.108148
[HOOKS] Total time spent in ROI: 3.108s
[HOOKS] Terminating

real    0m18.575s
user    1m15.025s
sys     0m0.904s
[PARSEC] [----------    End of output    ----------]
[PARSEC]
[PARSEC] BIBLIOGRAPHY
[PARSEC]
[PARSEC] [1] Bienia. Benchmarking Modern Multiprocessors. Ph.D. Thesis, 2011.
[PARSEC]
[PARSEC] Done.
```

Region of Interest (parallel execution)

PARSEC hooks API



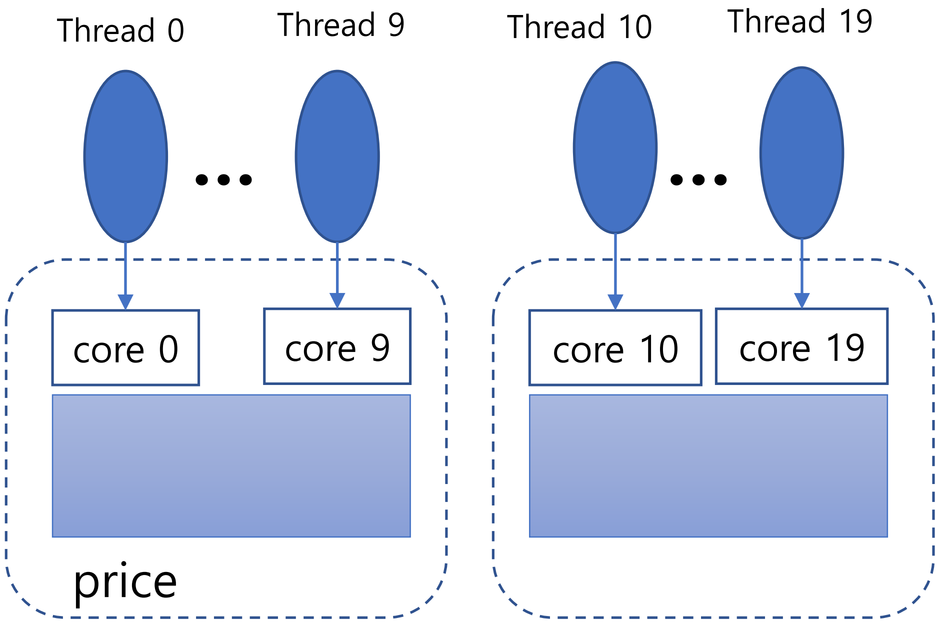
blackscholes: overview



- No NUMA awareness
- No page alignment for per-thread memory area
  - Frequent false sharing between NUMA nodes

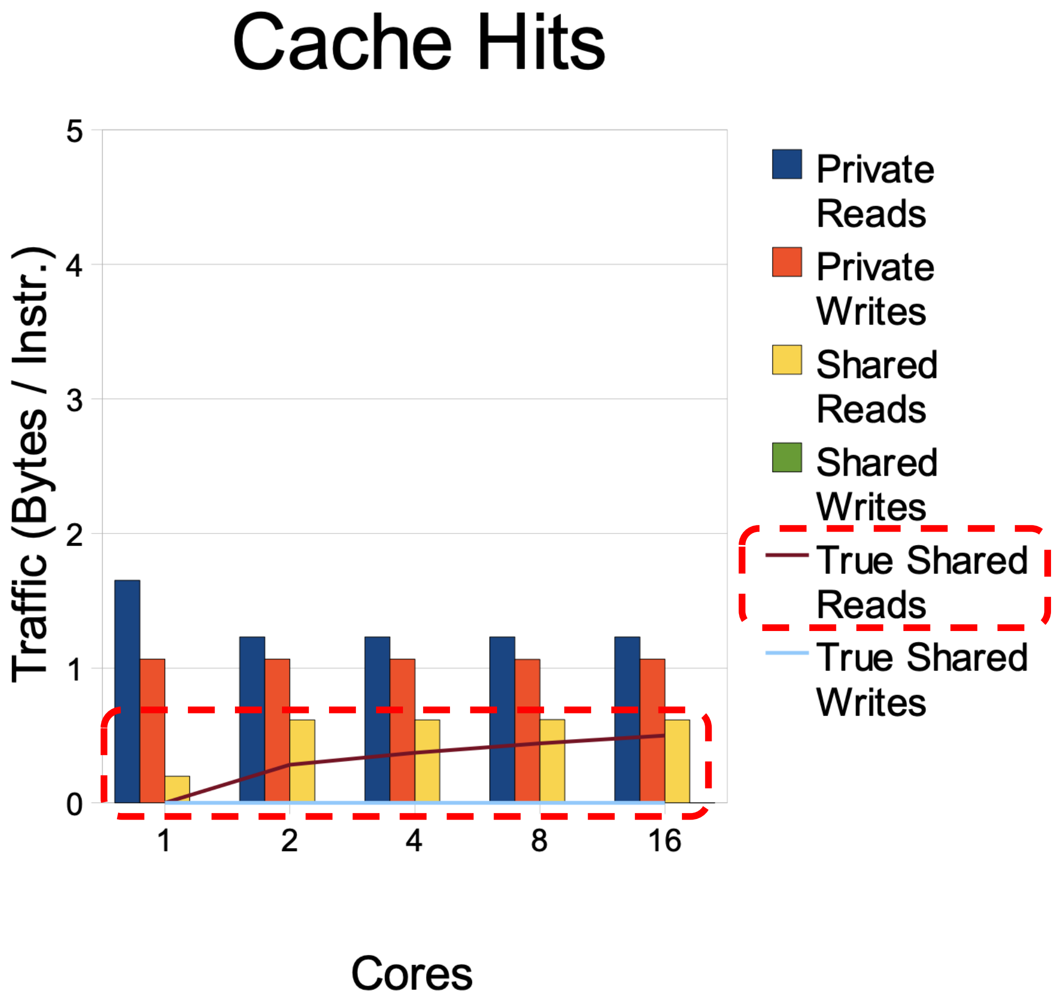
Refactoring blackscholes

- NUMA-aware memory allocation of ‘price’
  - Thread 0~9: NUMA node #0
  - Thread 10~19: NUMA node #1
    - (Assumption) 10 cores per NUMA node
  - Use numa_alloc_onnode()
  - Refer to man page
    - $ man numa
- CPU pinning of threads
  - CPU_SET()
  - sched_setaffinity()



Refactoring blackscholes (cont'd)

- What about the input memory?
  - buffer, buffer2
  - Read-only
    - Little chance of misses after first touch
  - True shared reads
    - NUMA-aware allocation is not that meaningful



ETRI's preliminary results

- Execution time in ROI (20 threads, 2 NUMA nodes, 10 cores per node)
  - blackscholes: 3.15 sec
  - blackscholes.numa: 3.0 sec
  - Approx. 5% of improvement observed
- In baremetal machines, the amount of improvement is not that impressive
  - However, the improvement becomes huge with GiantVM
- Your optimization can beat ETRI's one!

Example #3: SpMV/DMV

NUMA-aware programming

Overview

- Core part of Quantum simulation tool for Advanced Nanoscale Devices (Q-AND)
  - Developed by KISTI(Korea Institute of Science and Technology Information)
- Matrix-vector multiplication
  - Widely used computational kernel existing in many scientific applications
  - $y = Ax$
    - A (input matrix): **sparse(SpMV) or dense(DMV)** (immutable or mutable)
    - x/y (input/output vectors): dense/sparse
- Programming model
  - **OpenMP**

In-depth overview

- MVsample
  - A small test program running both SpMV & DMV repeatedly
- Repeatedly calls spmv()(sparce matrix multiplication) and dmv()(dense matrix multiplication) in separate for loop
  - Each call to spmv() or dmv() is parallelized by OpenMP
    - #pragma omp parallel for
- For each for loop iteration, spmv() / dmv() is called twice **with input / output vectors switched**

```
for(int ii = 0; ii < niter; ii++)
{
        printf("SPMV at iteration %d\n", ii+1);
        spmv(smatrix, VR, VI, WR, WI);
        spmv(smatrix, WR, WI, VR, VI);
}
```

```
for(int ii = 0; ii < niter; ii++)
{
        printf("DMV at iteration %d\n", ii+1);
        dmv(dmatrixR, dmatrixI, YR, YI, WR, WI, DIM);
        dmv(dmatrixR, dmatrixI, WR, WI, YR, YI, DIM);
}
```

NUMA optimization

- NUMA-aware memory allocation for all matrices used in DMV: A, x & y
- Thread pinning to each CPU core
- NUMA-aware allocation for SpMV is not easy
  - Refer to the source code