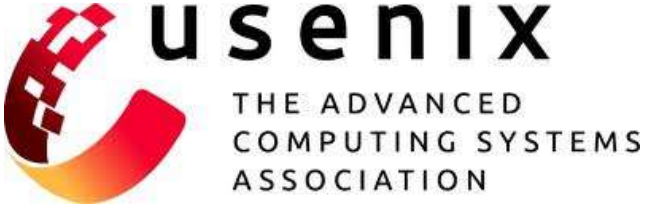
MLB: A Memory-aware Load Balancing Method for Mitigating Memory Contention



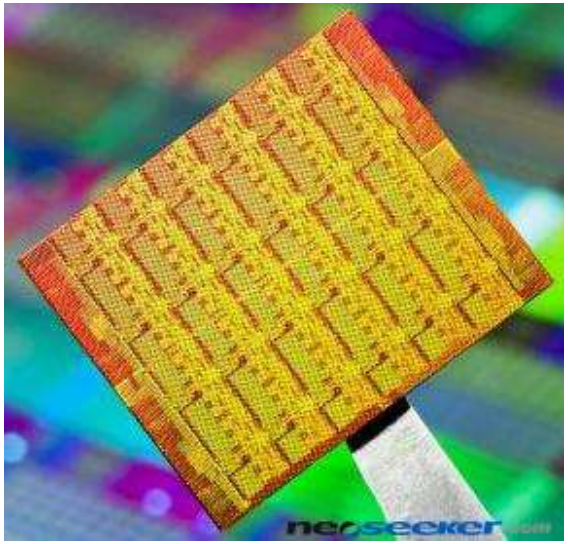
Dongyou Seo, Hyeonsang Eom and Heon Y. Yeom
Department of Computer Science and Engineering, Seoul National University, Korea





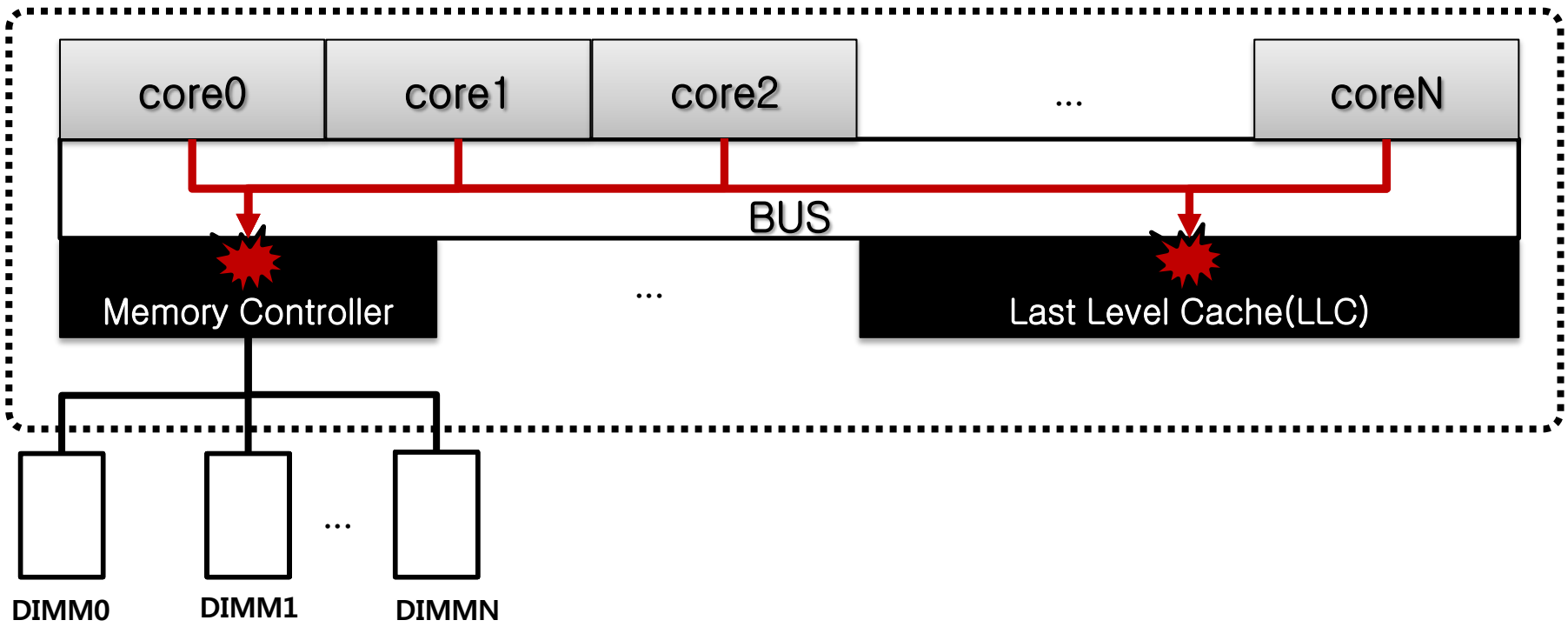
2014 Conference on Timely Results in Operating Systems (***TRIOS'14***), OCT 5, 2014
Broomfiled, Co, USA

Multicore CPU



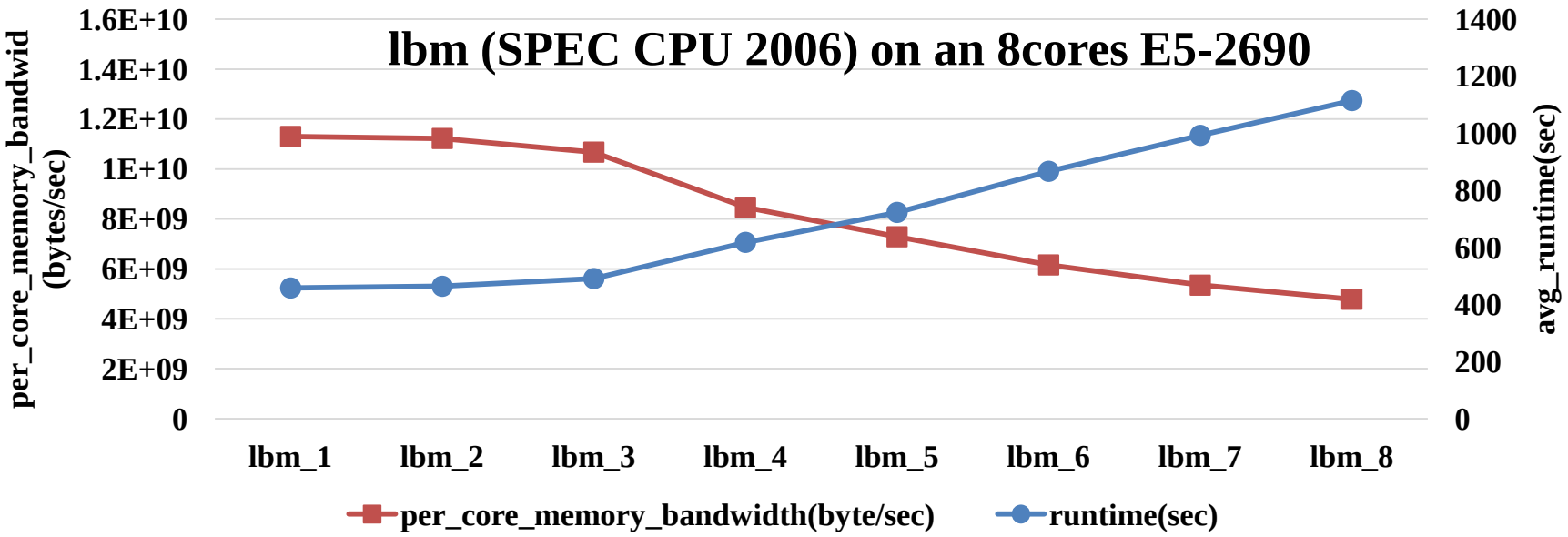
- Multicore has becomes the general CPU architecture
- Intel has recently introduced a Xeon E7-8890 v2 on which 15 physical cores are integrated
- 1,000 core processor will be possible, be said Intel http://www.neoseeker.com/news/15313-1000-core-processor-possible-says-intel/

Trend in the CPU architecture



- Symmetric Multi-Processing (SMP) may be the basic architecture to support parallel computing with shared resources such as the memory controller and last level cache
- Shared resources may be efficiently used for the fastest on-chip communication

Performance bottleneck on multicore CPU



- When there are too many requests for the shared resources, the contention for memory subsystems occurs
- The more lbms are executed at the same time, the longer becomes the runtime of each lbm and the lower gets the per-core memory bandwidth

Related work

- Various solutions have been developed to mitigate contention for memory subsystems
  - **Zhuravlev et al. Addressing Shared Resource Contention in Multicore Processors via Scheduling. In ASPLOS, 2010.**
  - Xie et al. Dynamic Classification of Program Memory Behaviors in CMPs. In Proc. of CMP-MSI, held in conjunction with ISCA, 2009.
  - Jiang et al. Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors. In PACT, 2008.
  - Kim et al. Virtual Machine Consolidation Based on Interference Modeling. In the journal of Supercomputing, April 2013.
  - And so on…
- However, most of the methodologies have been developed and used on the assumption that a task is the sole owner of a CPU core and thus they are not applicable in the multitasking case except for VBSC [Merkel et al. EuroSys'10]
- We will show the results of comparing our method with VBSC

Cluster imbalance

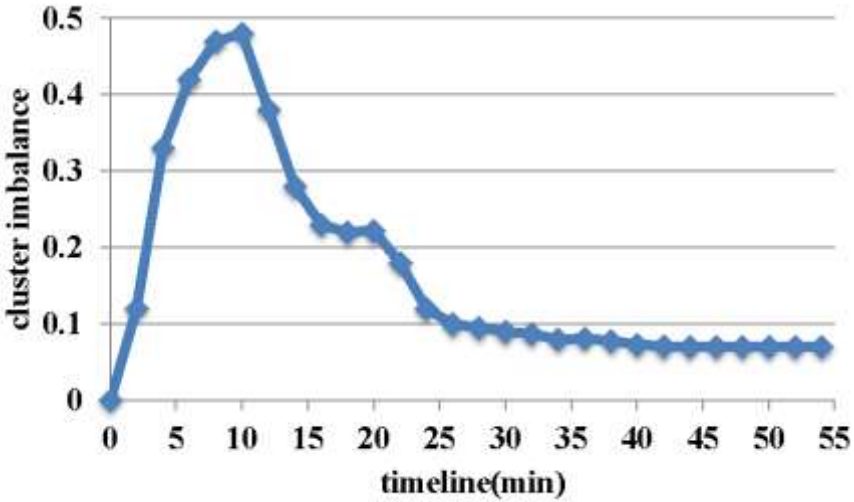
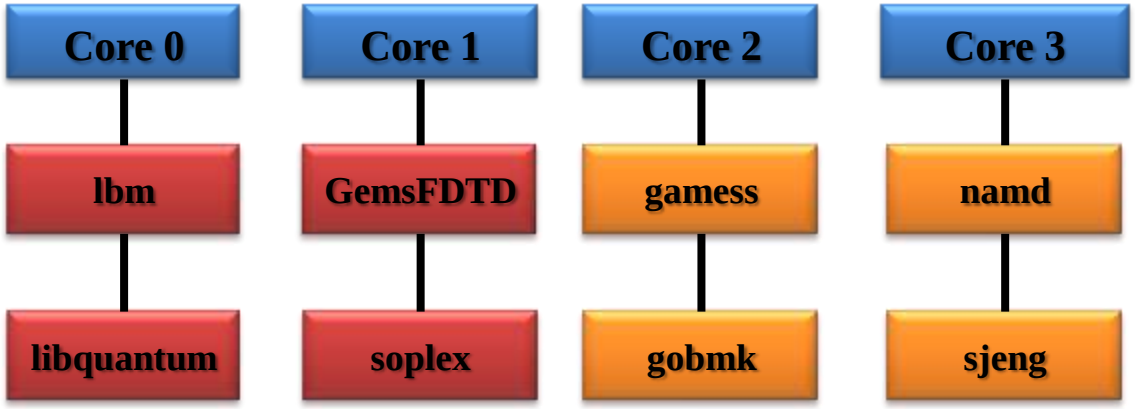


Figure. VMware cluster imbalance

- Cluster imbalance = $\sigma(Demand_{host_node}/Capacity_{host_node})$
- The server environment is characterized by a large dynamic range of resource demands with high variations over short time intervals [Reiss et al. CCGrid'13] [Gupta et al. SOCC'12]
- The dynamicity can make specific host nodes seriously overloaded and hence it prevents a task executing in the nodes from being the solo owner of a CPU core

Memory-aware Load Balancing(MLB)



- Memory-aware Load Balancing (MLB) is a solution dealing with the multitasking case by mitigating memory contention
- MLB migrates memory intensive tasks in several cores and compels the only designated cores to generate high memory traffics, not all cores
- MLB can dynamically increase and decrease the number of the cores by complying with our memory contention model

Contributions

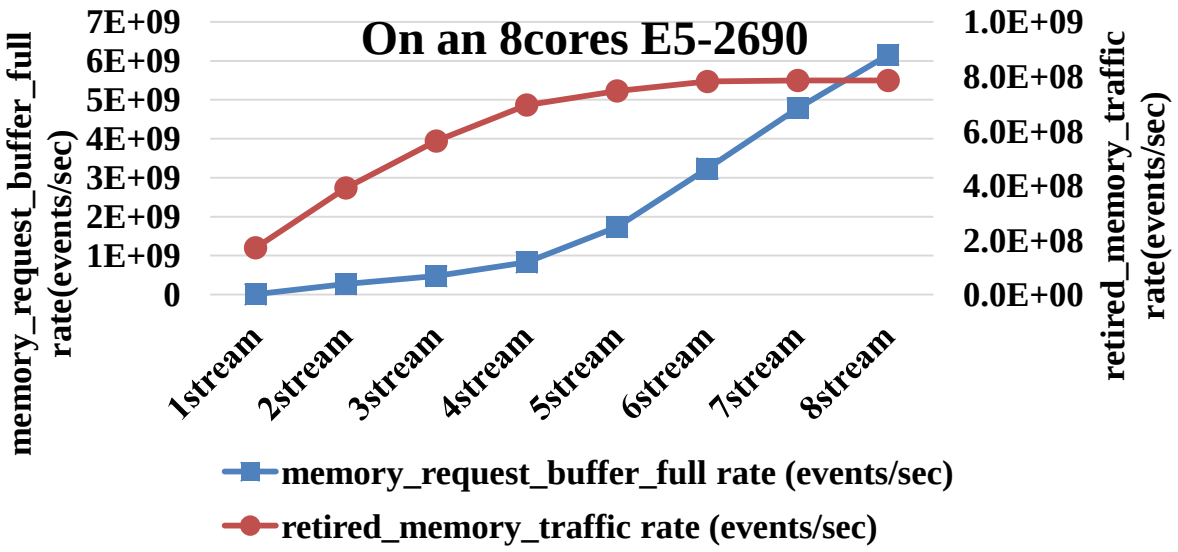
- Presenting simple models indicating the level of memory contention and predicting the maximum number of the memory intensive tasks spreading on multiple cores for dynamic MLB operation
- Evaluating MLB on both server-level and desktop-level CPUs
- Comparing MLB with the state of the art method considering the multitasking case, Vector Balancing & Sorted Co-scheduling (VBSC)

Contents

- Memory Contention Modeling
- Design and Implementation of MLB
- Evaluation
- Comparison with VBSC
- Extension of MLB for multi-threaded applications
- Conclusion

Memory Contention Modeling

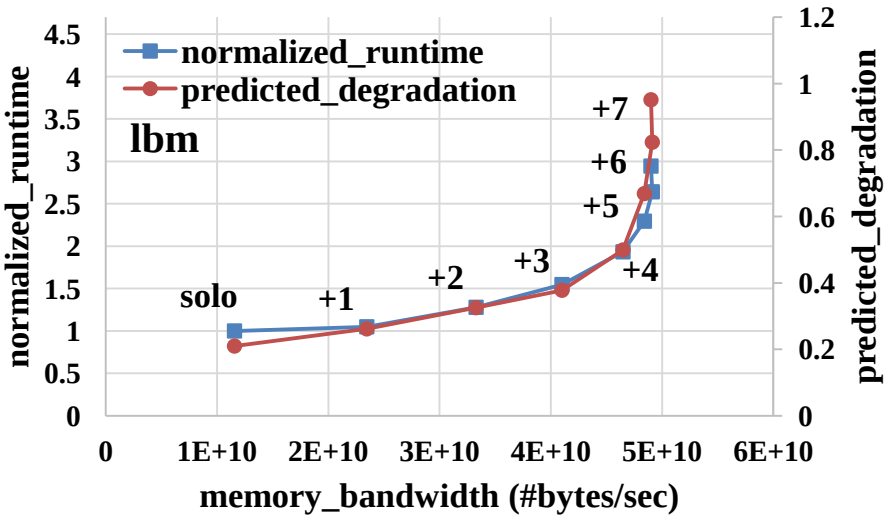
Memory Contention Modeling

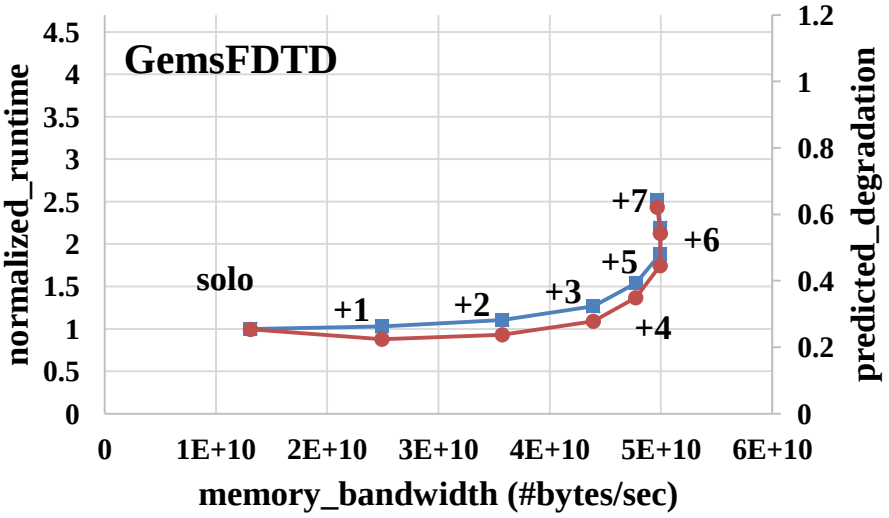


- MLB can dynamically recognize the current contention status of memory subsystems with a simple contention model and migrates memory intensive tasks in the designated cores with respect to this model
- Our model uses the correlation between retired memory traffic rate and memory request buffer full rate
  - **Memory contention level** = $memory\ request\ buffer\ full\ rate \,/\, retired\ memory\ traffic\ rate$
- The contention model indicates the efficiency in memory access

Memory Contention Modeling (Cont'd)

- Correlation between memory bandwidth, memory contention level and performance (On an E5-2690 with 8cores)

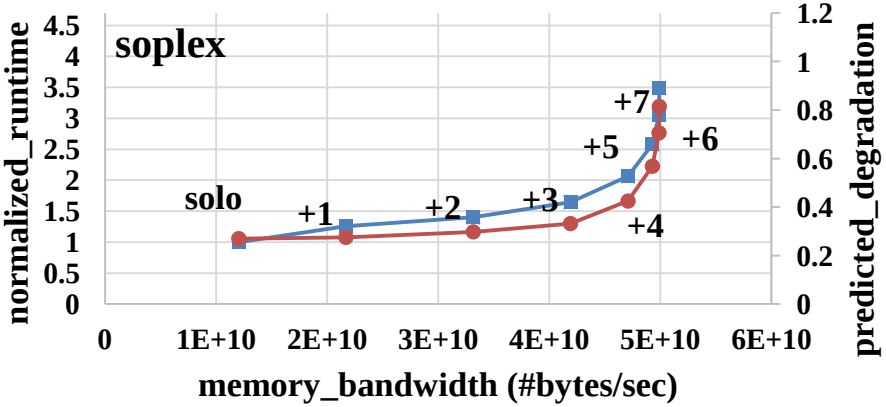


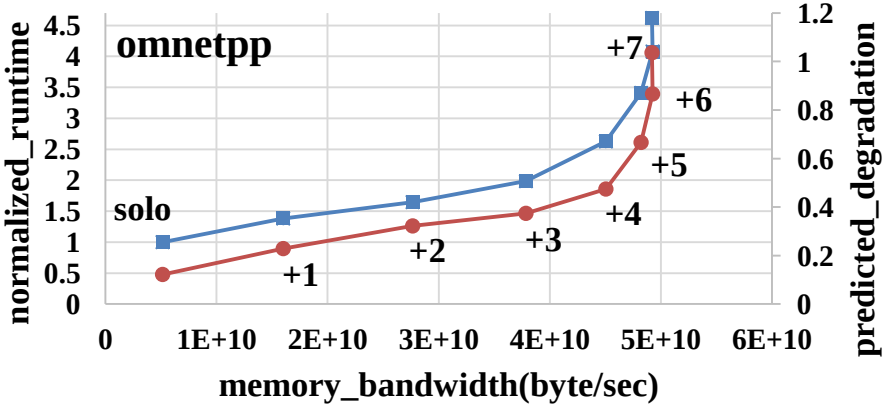


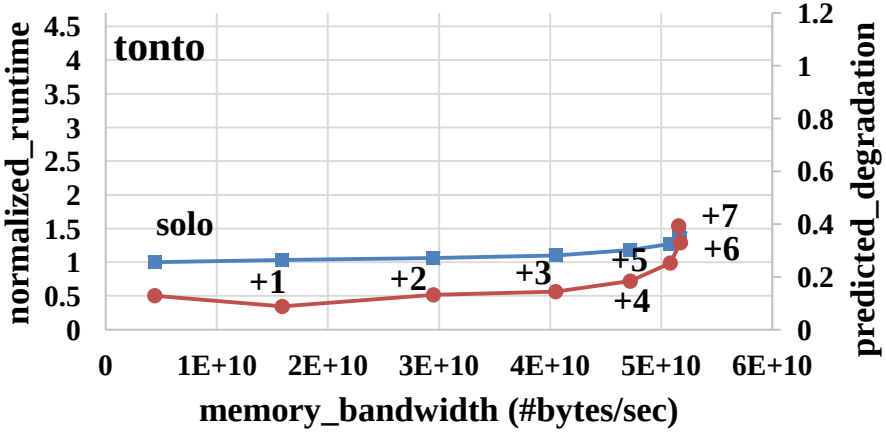
- Target applications + n stressors
- Runtime baseline is solo execution runtime without any interference
- Sensitivity = $(1 - \frac{LLC_{miss}}{LLC_{ref}}) \times \frac{Cycle_{stall}}{Cycle_{retired}}$ [Kim et al. Journal of SC'13]
- Predicted Degradation = $Memory\ contention\ level_{system-wide} \times Sensitivity_{target}$

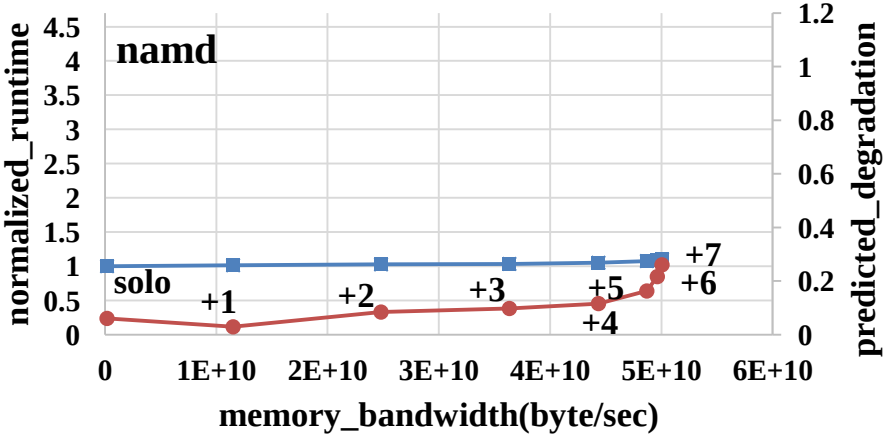
Memory Contention Modeling (Cont'd)

- Correlation between memory bandwidth, memory contention level and performance





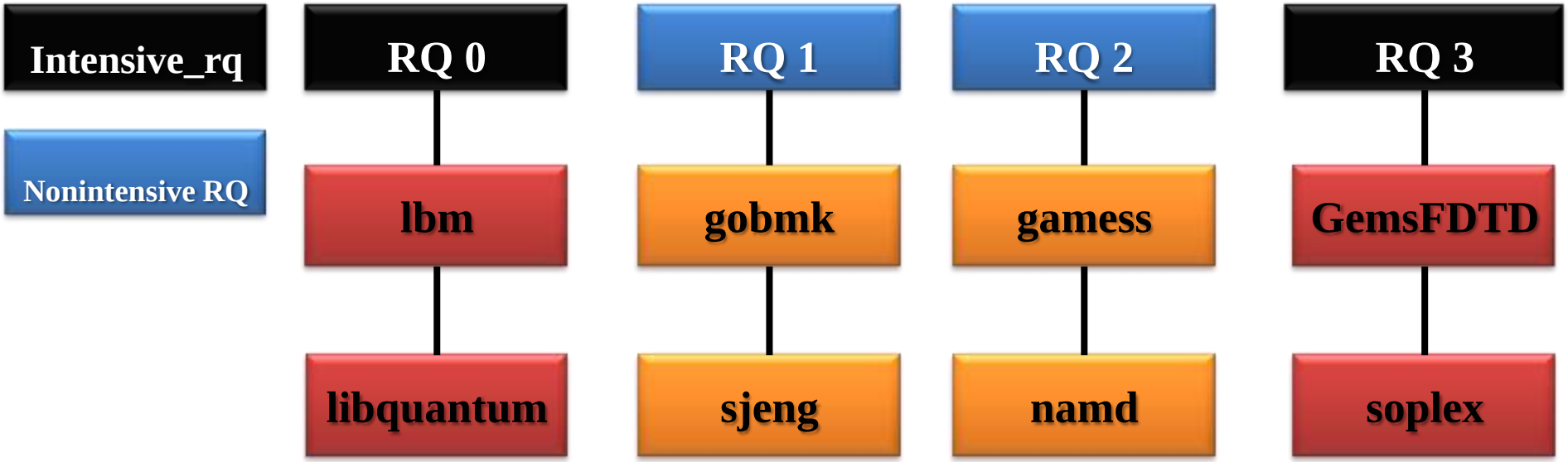




Design and Implementation of MLB

Design and Implementation of MLB

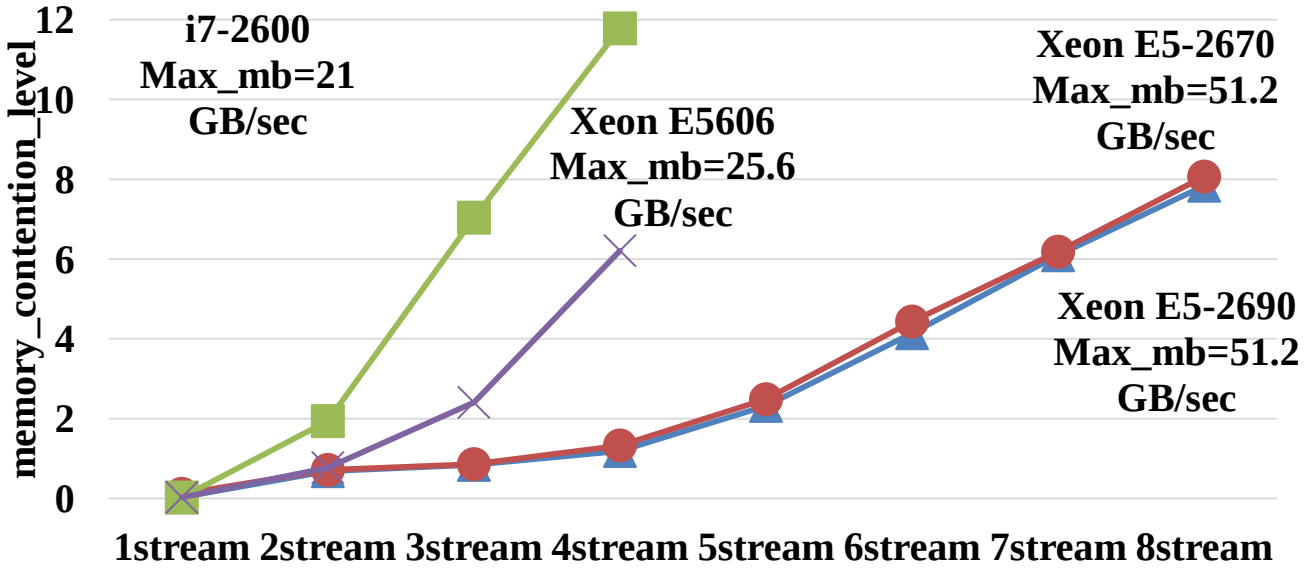
- Intensive runqueue and non intensive runqueue lists



- MLB mechanism is implemented in Linux kernel 3.2.0
- Memory intensive tasks have the higher intensity than the average of all tasks
- Memory intensive tasks should be executed on the cores whose runqueues are intensive ones
- Non memory intensive tasks should be executed on the cores whose runqueues are non intensive one

Design and Implementation of MLB (Cont'd)

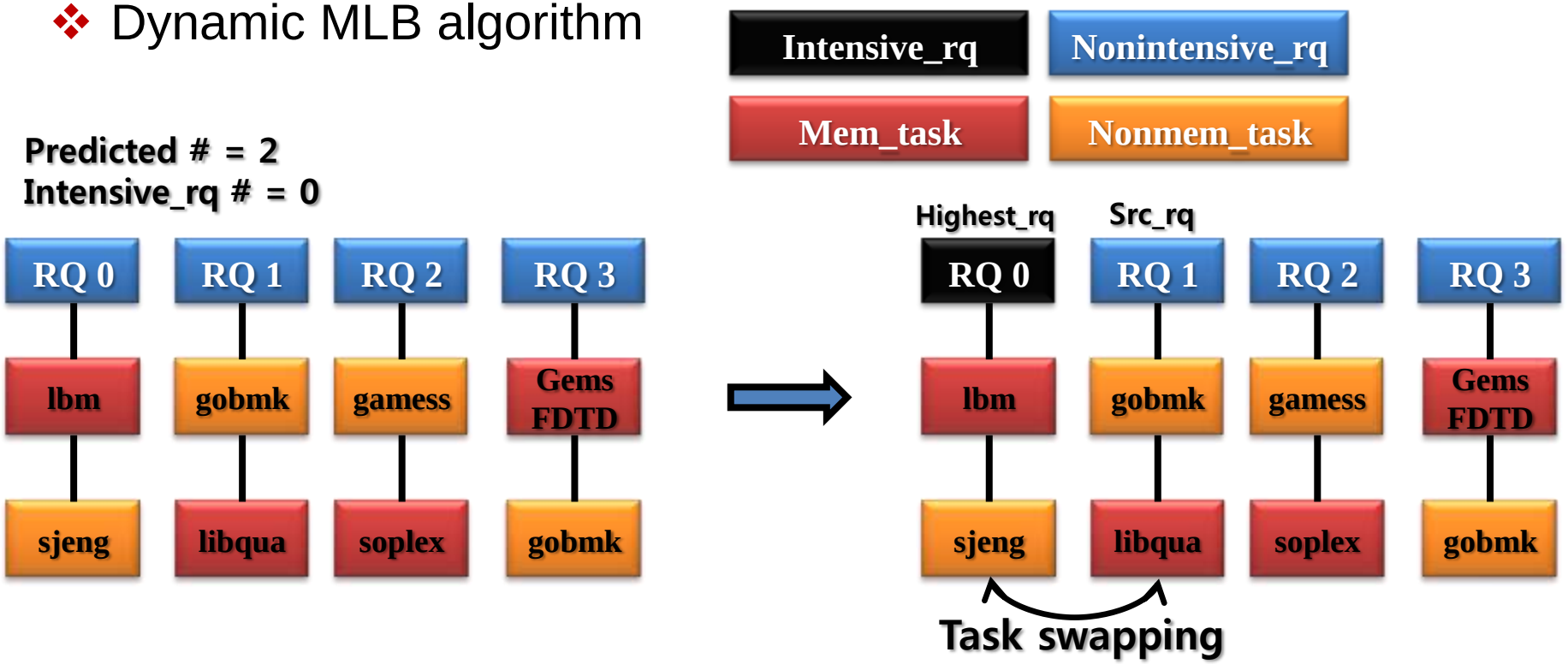
- Predicted number



- Predicted number is the anticipated number of memory intensive tasks which are to be scheduled on each core at the same time
- Predicted Number = $\sqrt{\frac{Memory\ Contention\ Level - Baseline}{Inclination}}$
  - Inclination $\propto {}^{1}/_{Max\ Memory\ Bandwidth}$
  - Baseline = Memory Contention Level Baseline

Design and Implementation of MLB (Cont'd)

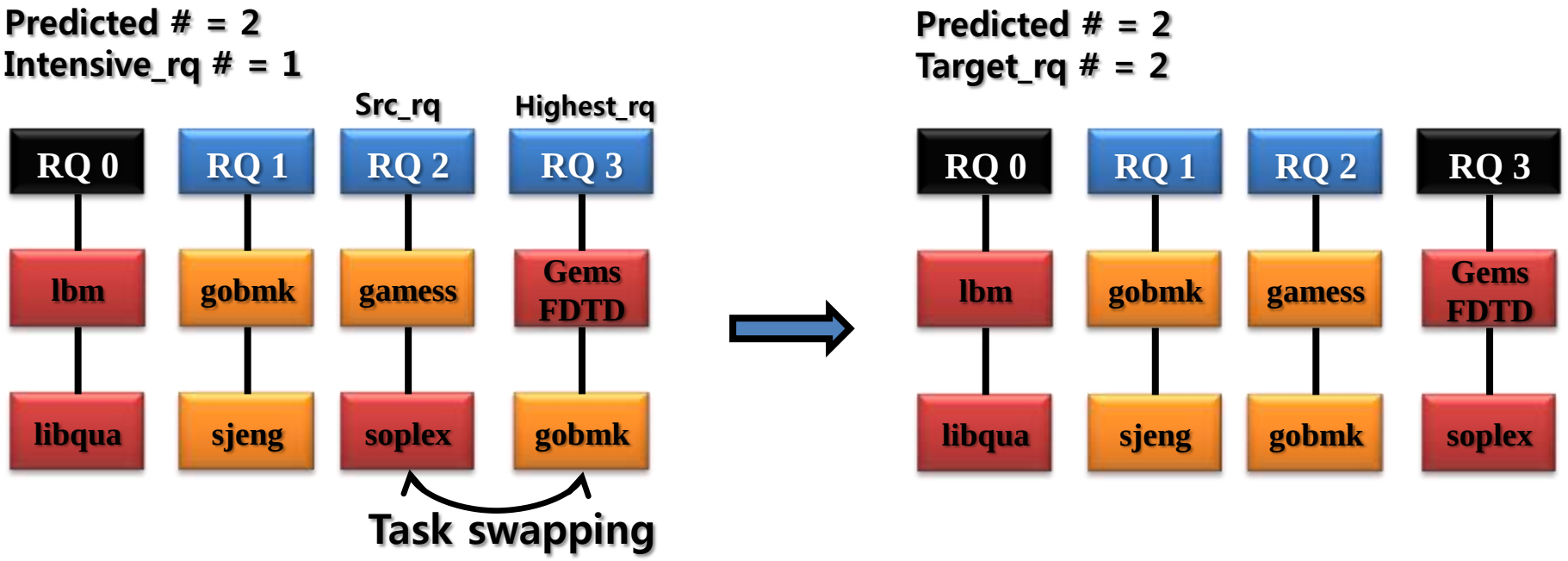
- Dynamic MLB algorithm



- Src_rq is a non intensive runqueue which includes the highest memory intensive task in non intensive runqueue list
- Dest_rq is an intensive runqueue which has the highest intensity sum for all tasks in a non intensive runqueue selected to minimize the number of task swapping

Design and Implementation of MLB (Cont'd)

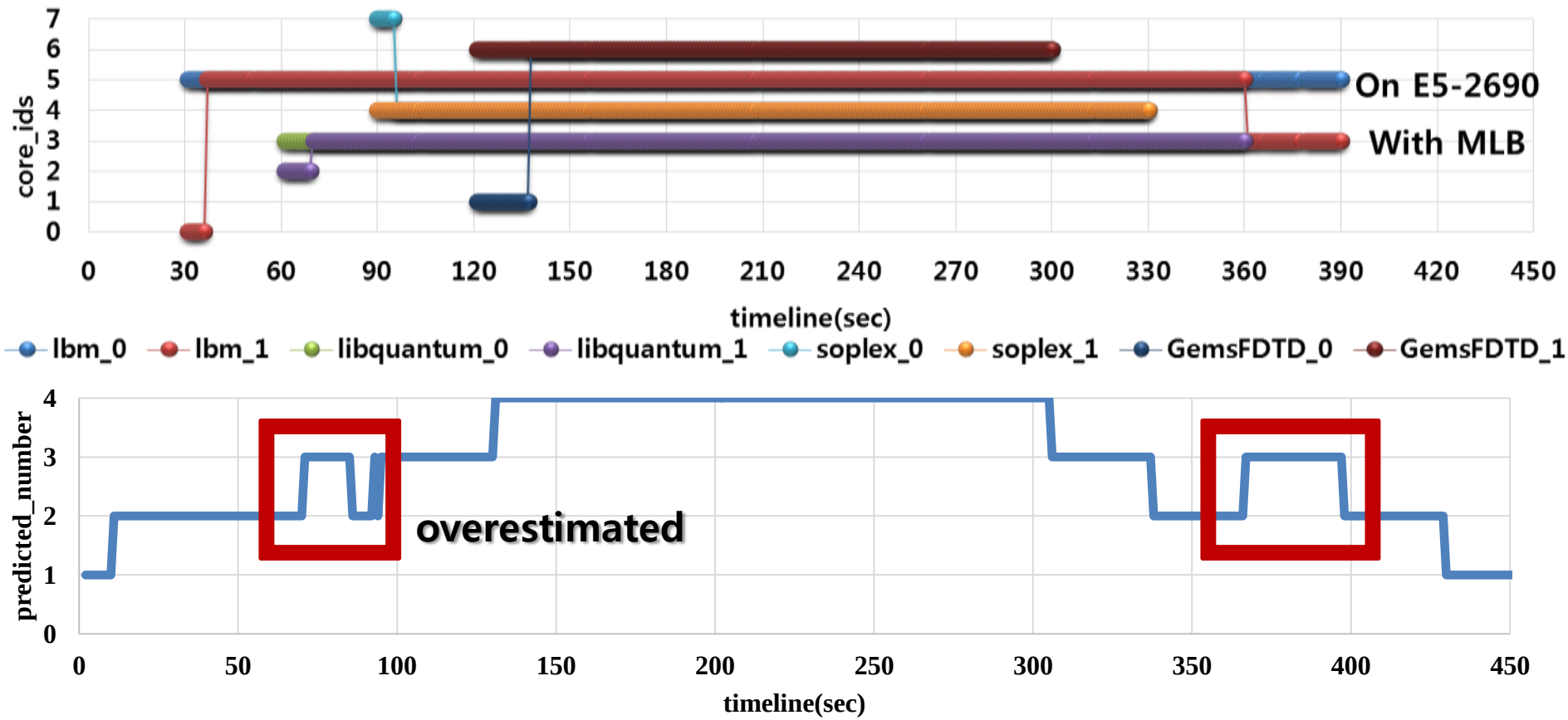
- Dynamic MLB algorithm



- Until only non memory intensive tasks reside in the source rqs, MLB continues to conduct task swapping
- If needed, a non target rq is inserted to the target rq list

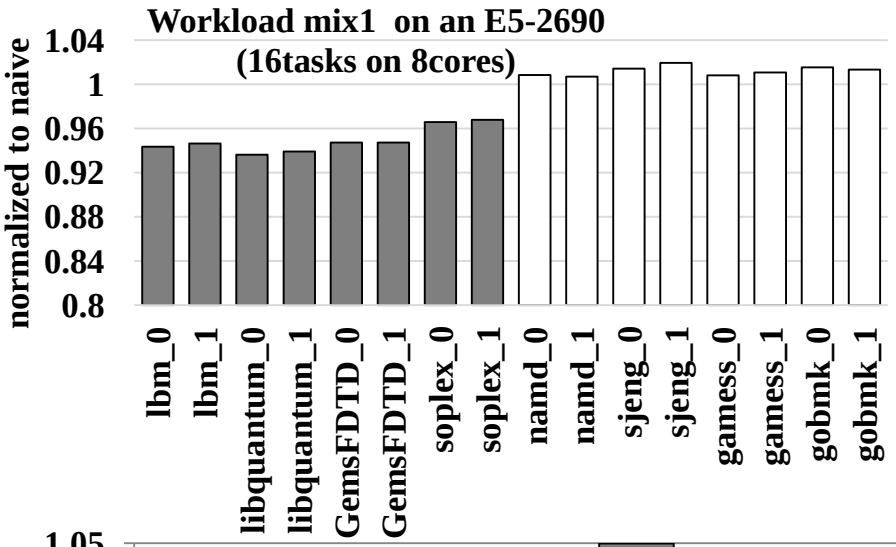
Evaluation

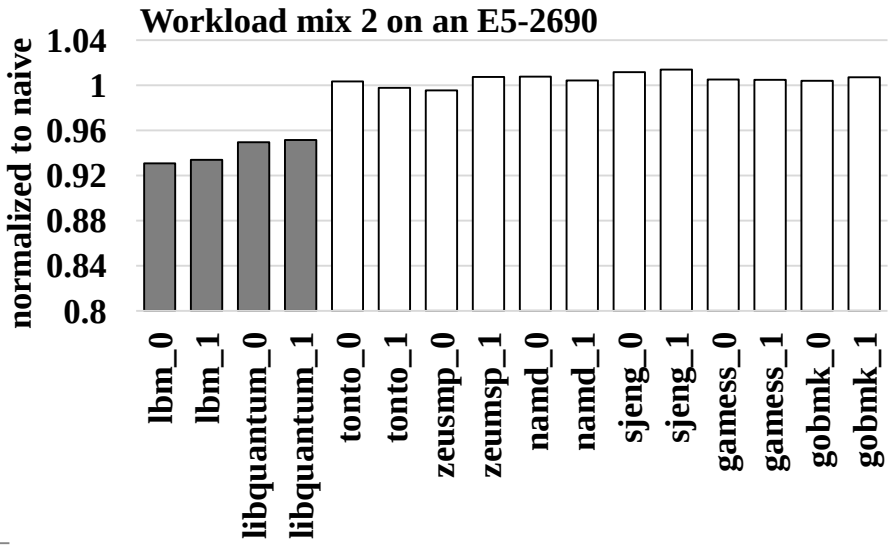
Evaluation (Model applicability)

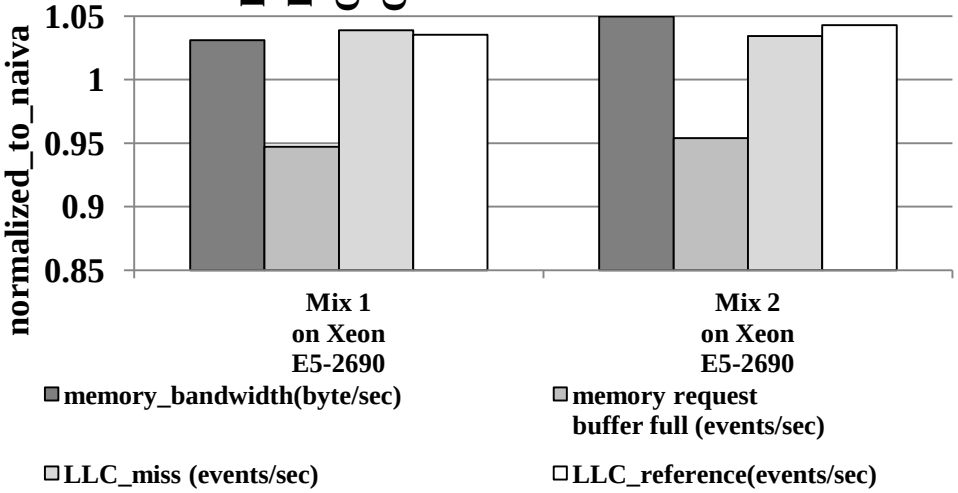


- Some sections are overestimated, but there are unnecessary task migrations due to prevention mechanism
  - Src_rq->highest_task->intensity <= intensive_rq_list->lowest_task->intensity

Evaluation (System performance)

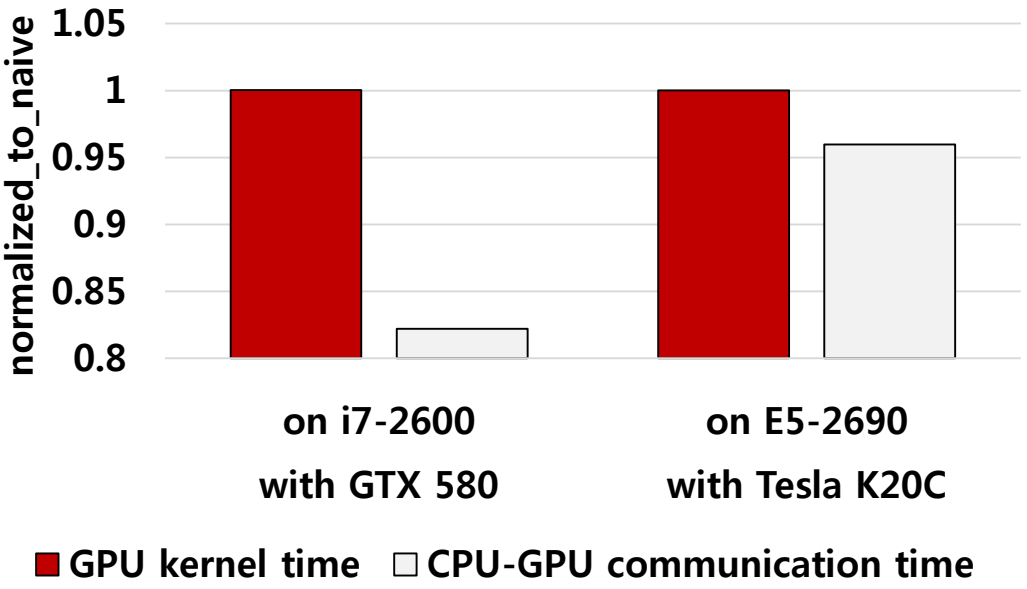






# of workload mixes	Workloads (SPEC CPU 2006)
Workload-mix1 (E5-2690)	2lbms, 2libquantums, 2GemsFDTDs, 2soplexs, 2namds, 2sjengs, 2gamesses, 2gobmks
Workload-mix2 (E5-2690)	2lbms, 2libquantums, 2tontos, 2zeusmps, 2namds, 2sjengs, 2gamesses, 2gobmks

Evaluation (Additional benefit)

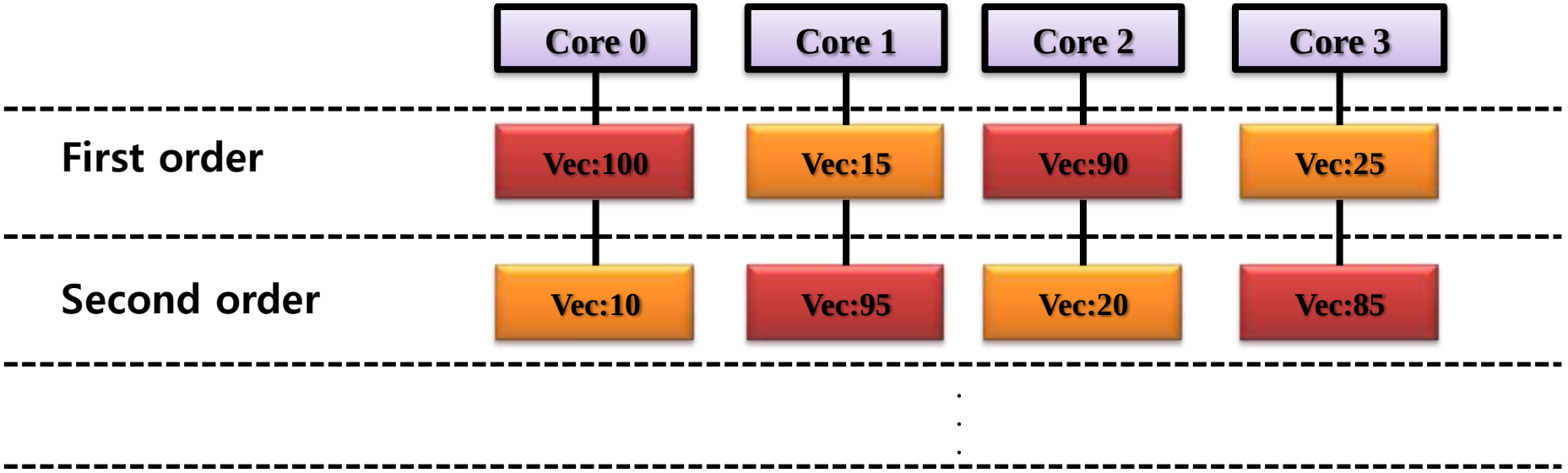


- (Workload mix 1 – a soplex) + a streamcluster (Rodinia benchmark)
- MLB can indirectly lead to improvements in the performance of CPU-GPU communication via the mitigation of memory contention when memory contention is intensive (17.8% on i7-2600 with GTX 580 and 4.02% on E5-2690 with Tesla K20C)

Comparison with VBSC

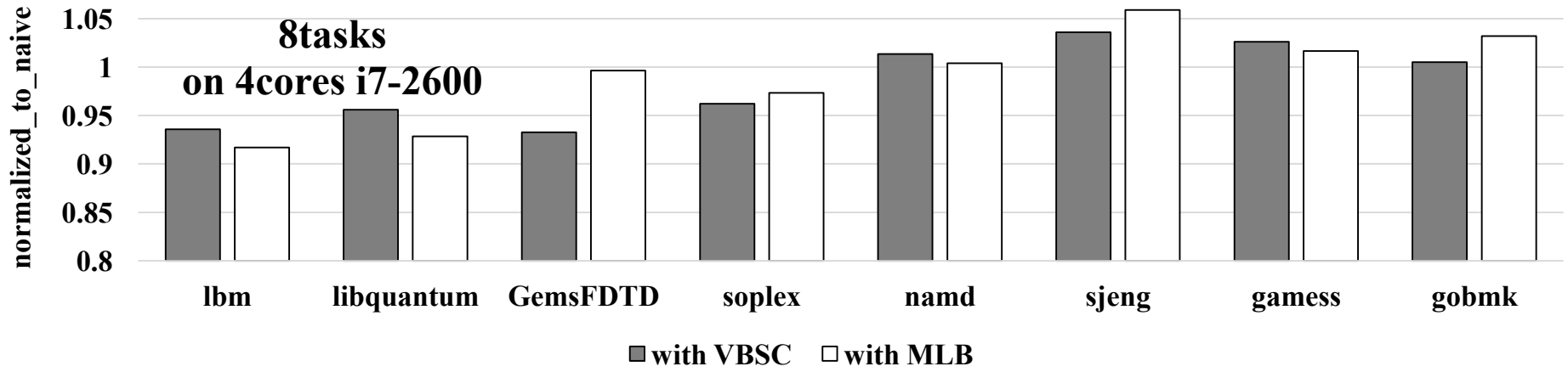
Comparison with VBSC

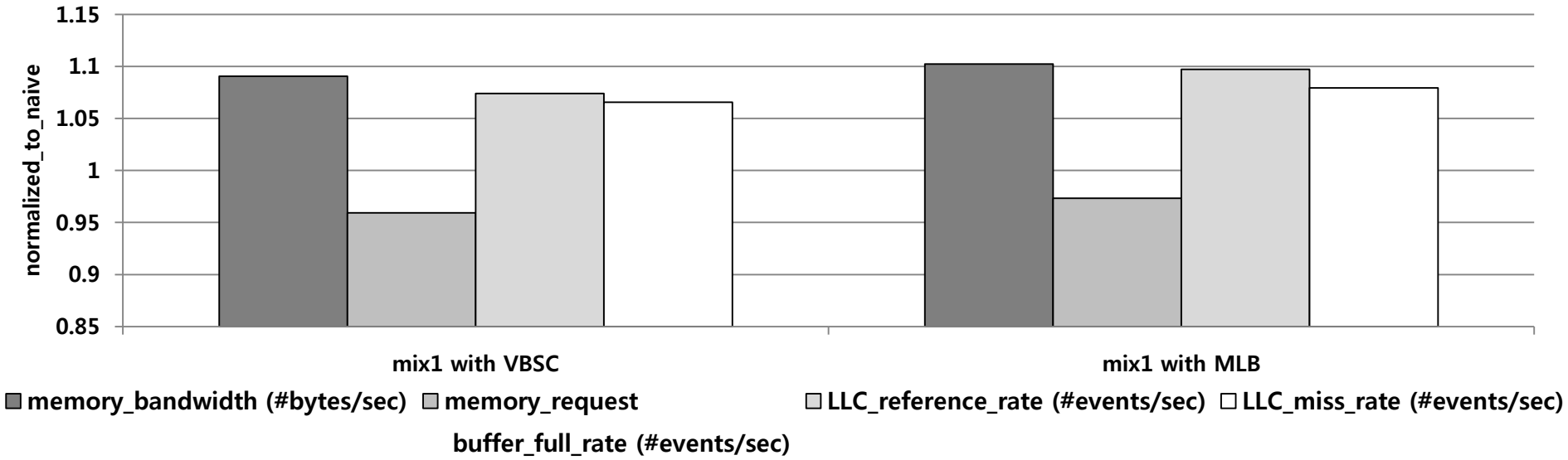
- The state of the art method considering multitasking case is Vector Balancing & Sorted Co-scheduling (VBSC) [Merkel et al. EuroSys'10]



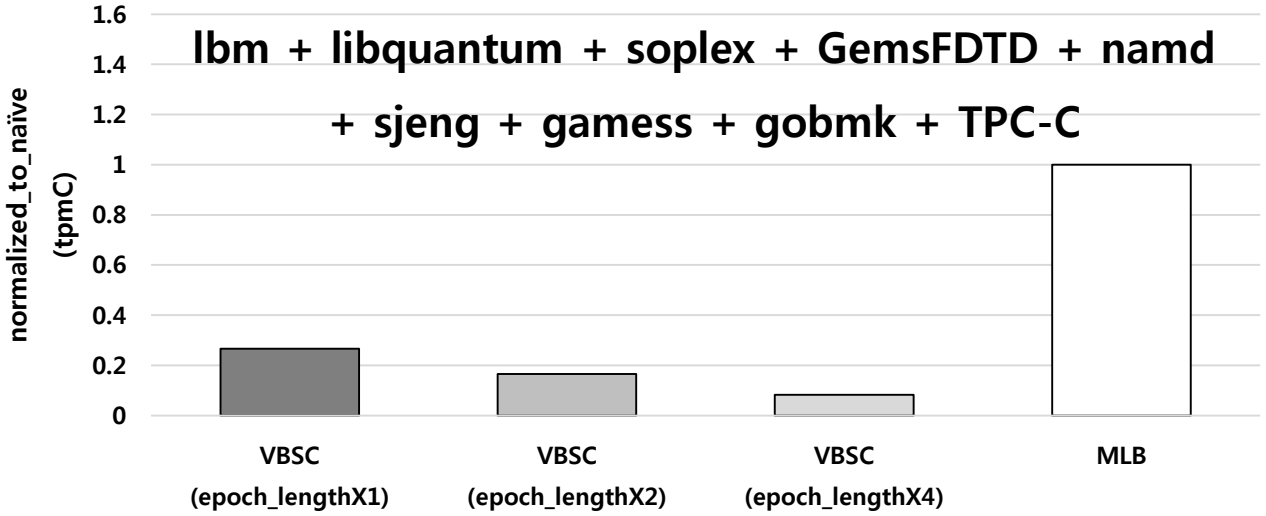
- varsum $= \sum_{i=o}^{(\#cores\ -1)} \sigma(Per\ task\ vector)$ (Vector balancing)
- On even cores, tasks are sorted in vector-ascending order (Sorted Co-scheduling)
- On odd cores, tasks are sorted in vector-descending order (Sorted Co-scheduling)
- We ported the VBSC mechanism in the current CFS scheduler and compared VBSC with MLB

Comparison with VBSC (CPU-bound application)



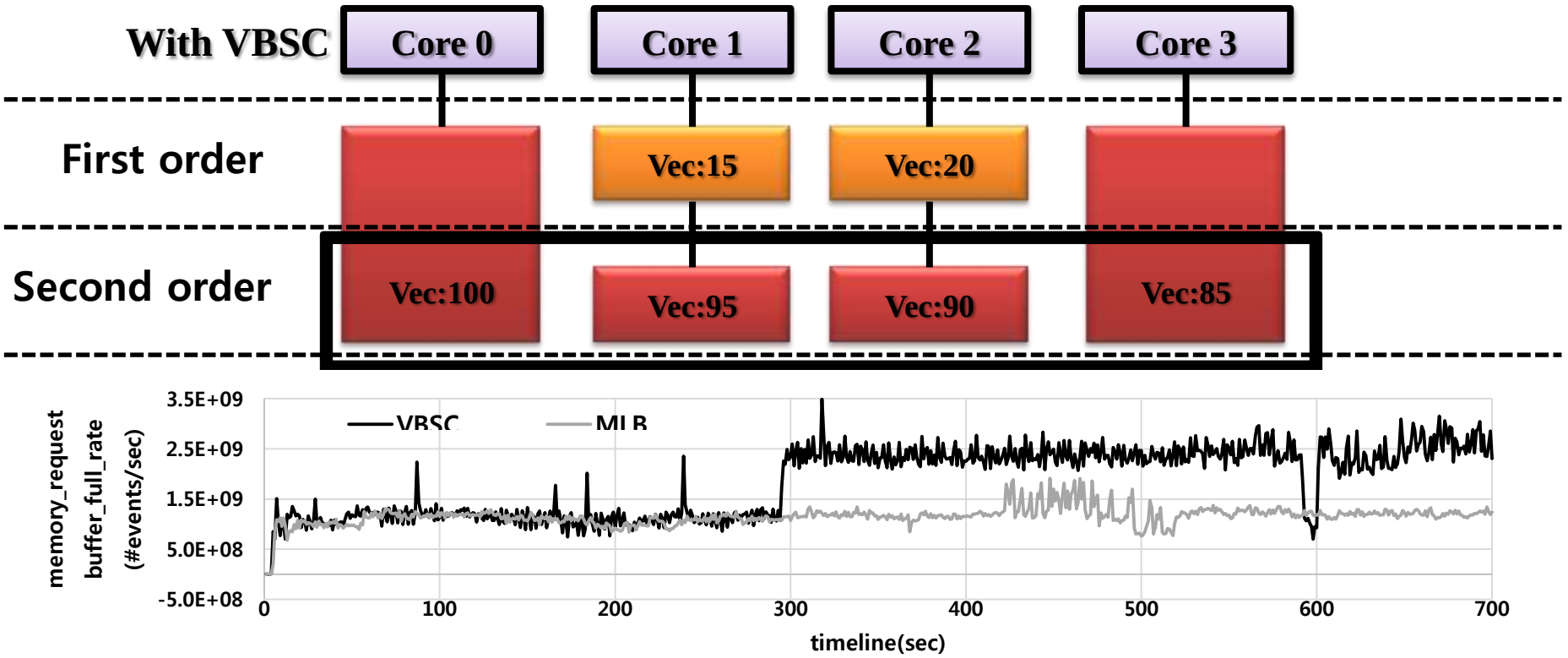


Comparison with VBSC (I/O-bound application)



- VBSC cannot avoid modifying the timeslice mechanism for sorting tasks in a runqueue, and hence this scheduling scheme can be in conflict with I/O-bound tasks
- An I/O boud application, TPC-C benchmark, is seriously degraded by about 300% with VBSC
- MLB can allow the I/O boud task to be performed without causing performance degradation compared with the naïve Linux scheduler

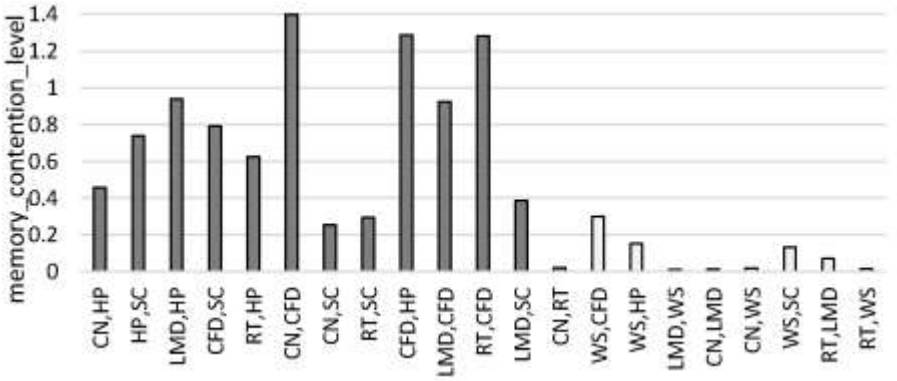
Comparison with VBSC (Worst case)

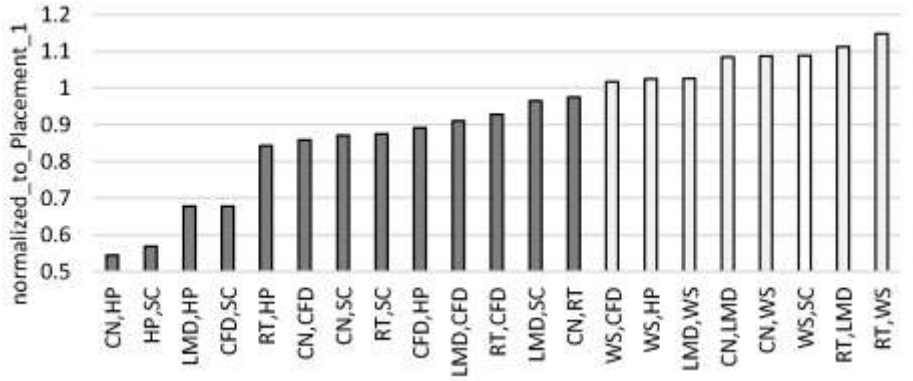


- Task termination can make the worst case with VBSC
- All memory intensive tasks can be scheduled at the same time after at 300secs
- But, there is no growth of memory request buffer full rate with MLB

Extension of MLB for multi-threaded application

Extension of MLB for multi-threaded applications





- MLB can reduce the level of the parallelism, but the placement by MLB can decrease the memory contention and increase the cache hit ratio [Dey et al. ISPASS'11]
- A pair for seven applications (total 21 pairs) and each application creates 8 threads on 8 cores of Xeon E5-2690 (**16threads on 8cores E5-2690**)
- canneal, ratrace (PARSEC), streamcluster, hotspot, lavaMD, cfd (rodinia), water spatial (SPLASH2x)
- The approach in which the highly intensive tasks are migrated to specific cores as in MLB can be extended to multi-threaded applications

Conclusion

- MLB has three stronger points than VBSC
  - MLB mechanism can be more easily implemented in both user and kernel levels than VBSC without the modification of timeslice,
  - MLB can allow I/O boud applications to be executed without causing performance degradation unlike VBSC
  - MLB can handle the worst case which can be shown by the VBSC mechanism
- We will extend MLB on the bigger system which includes those in the NUMA architecture and multiple GPGPUs

Thanks
Any questions

