Merchandiser: Data Placement on Heterogeneous Memory for Task-Parallel HPC Applications with Load-Balance Awareness

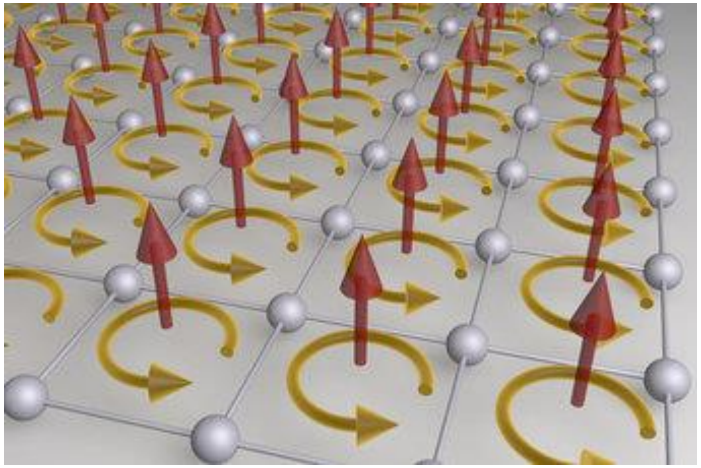
Dong Li

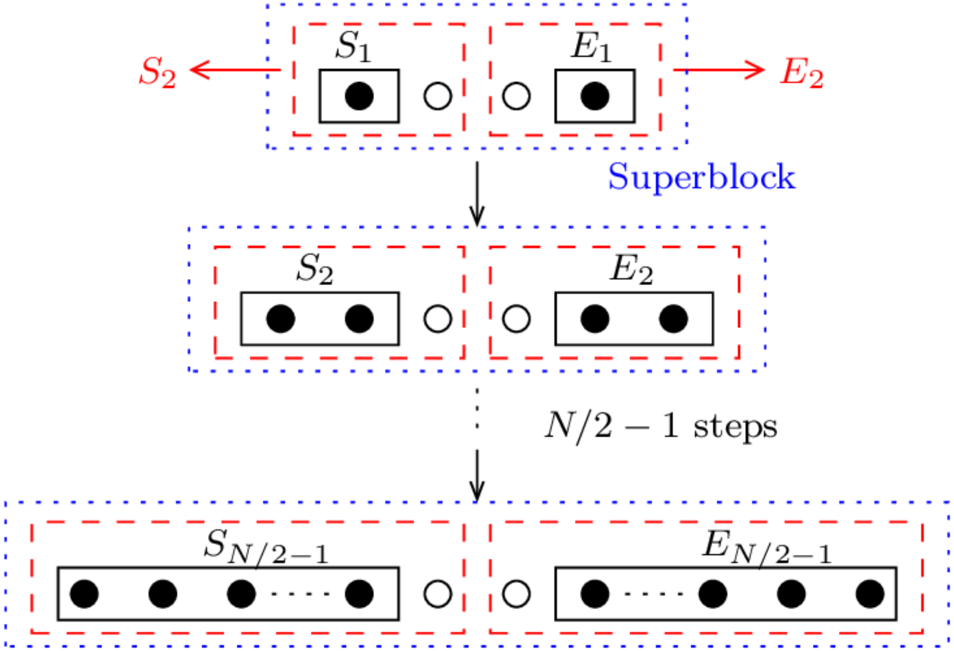
MARCH 13-14, 2023

Electrical Engineering and Computer Science
University of California, Merced

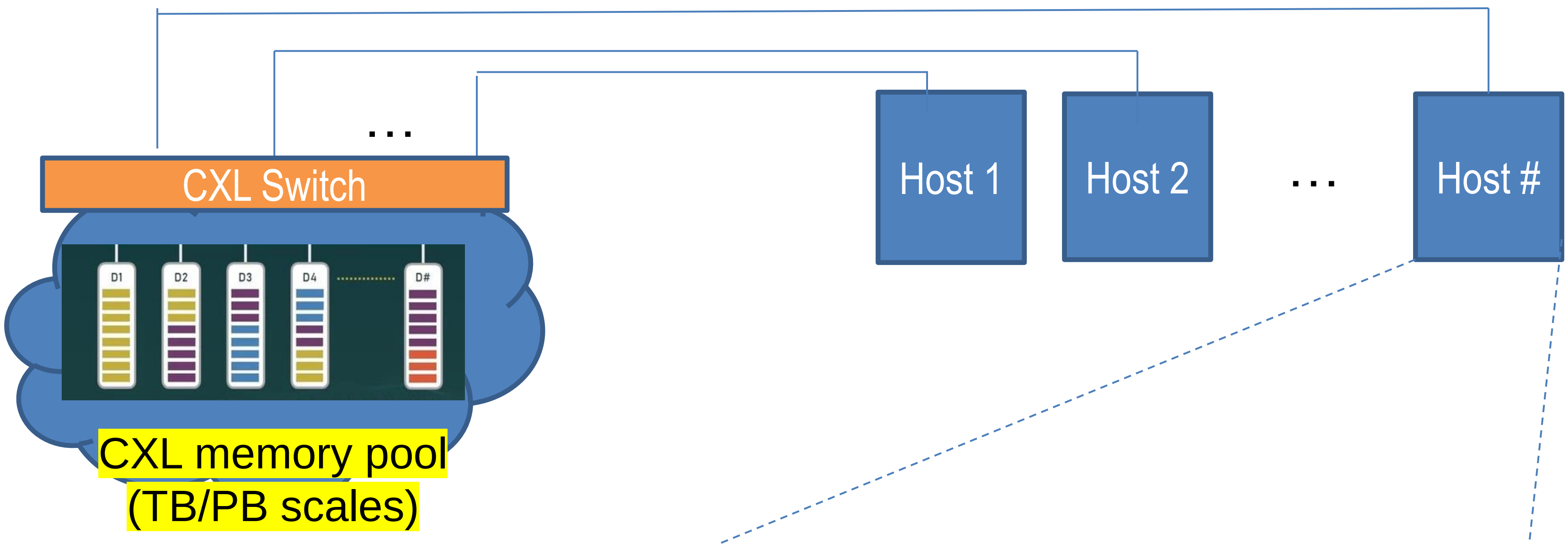
Some HPC applications need large memory capacity

- Example:
  - Density matrix renormalization group (DMRG), a numerical algorithm in quantum many-body systems, can consume 1.271 TB memory in a single machine





Big memory systems tend to be heterogeneous



CXL Switch

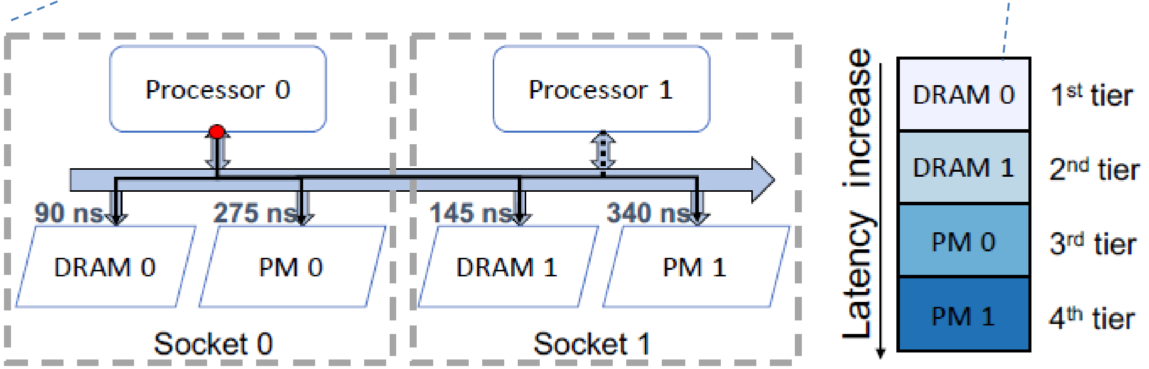
Host 1

Host 2

Host #

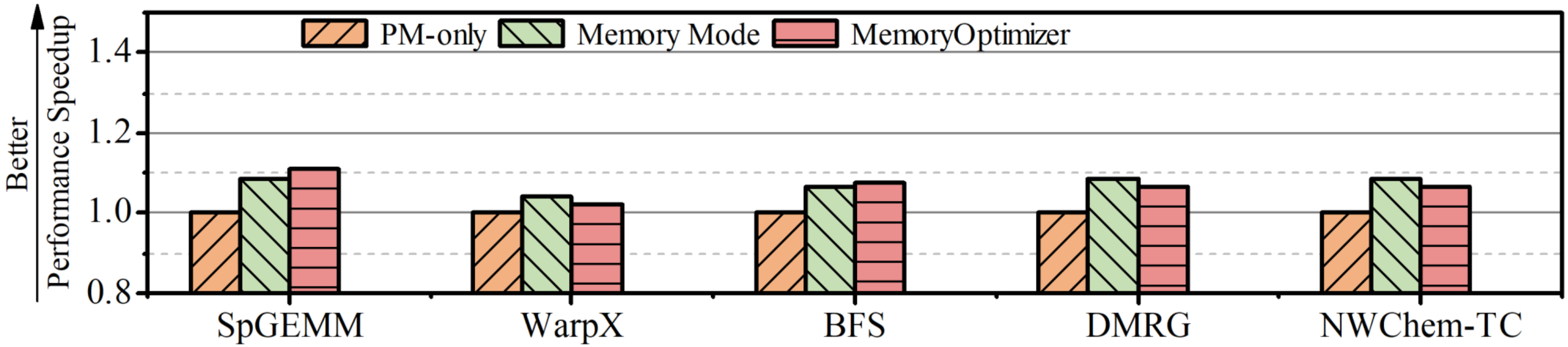
CXL memory pool
(TB/PB scales)

Traditional wisdom: identify frequently accessed pages and migrate them to fast memory as much as possible



DRAM 0 1st tier
DRAM 1 2nd tier
PM 0 3rd tier
PM 1 4th tier

Traditional wisdom does not effectively guide page migration



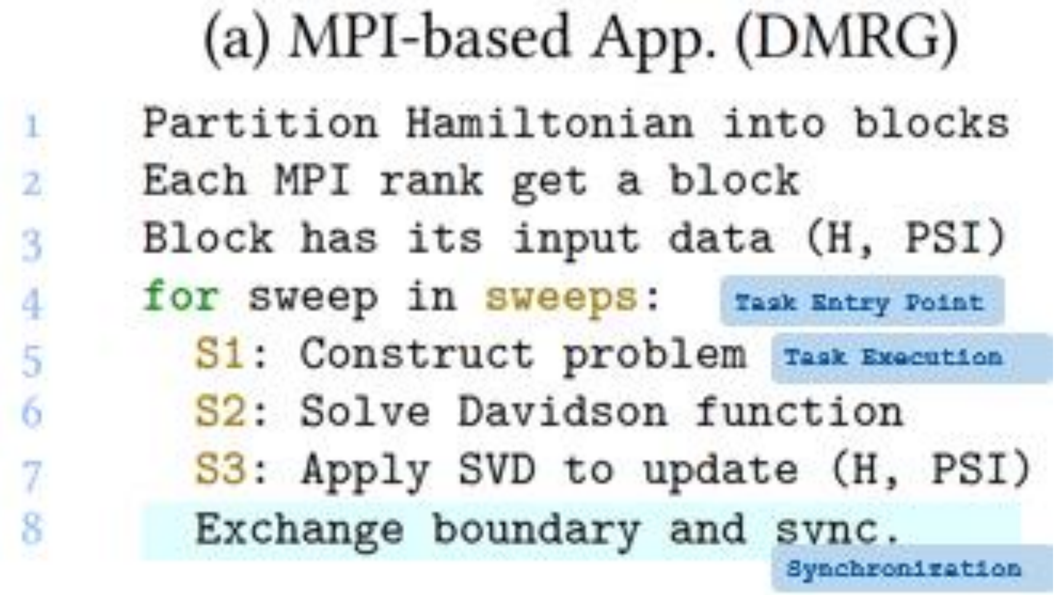
Performance of Memory Mode, MemoryOptimizer, and PM-only, normalized to the performance of PM-only execution

- 192 GB DRAM as fast memory and 1.5 TB Intel Optane Persistent Memory as slow memory
- Memory mode (a hardware-based solution) and MemoryOptimizer (a software solution from Intel) improves performance by less than 10%

What’s going on?

Let’s look at these applications

(a) MPI-based App. (DMRG)



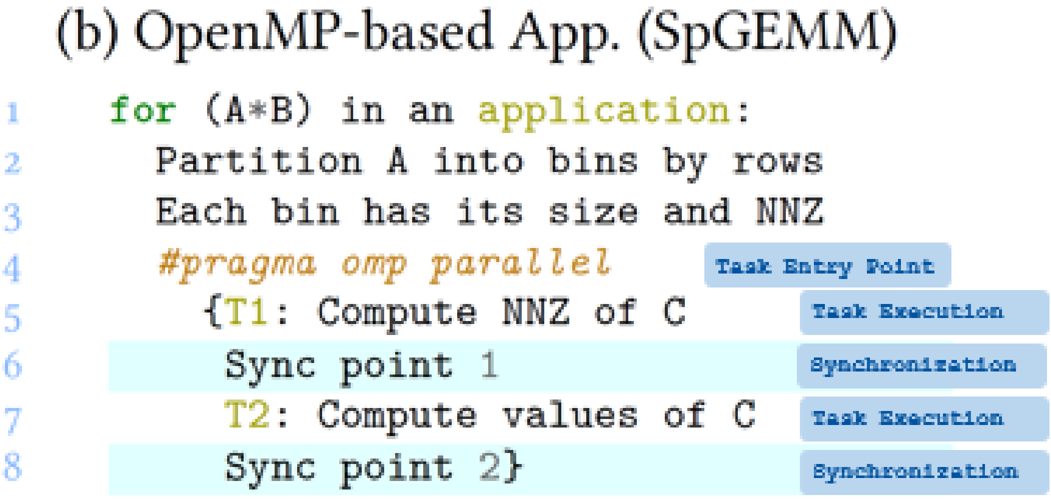
```
Partition Hamiltonian into blocks
Each MPI rank get a block
Block has its input data (H, PSI)
for sweep in sweeps:
  S1: Construct problem
  S2: Solve Davidson function
  S3: Apply SVD to update (H, PSI)
  Exchange boundary and sync.
```

Task

- An iteration of the loop is regarded as a task instance
- The task is repeatedly executed
- Different task instances use different inputs (i.e., PSI)
- There is a global sync among MPI processes

What’s going on?

Let’s look at these applications



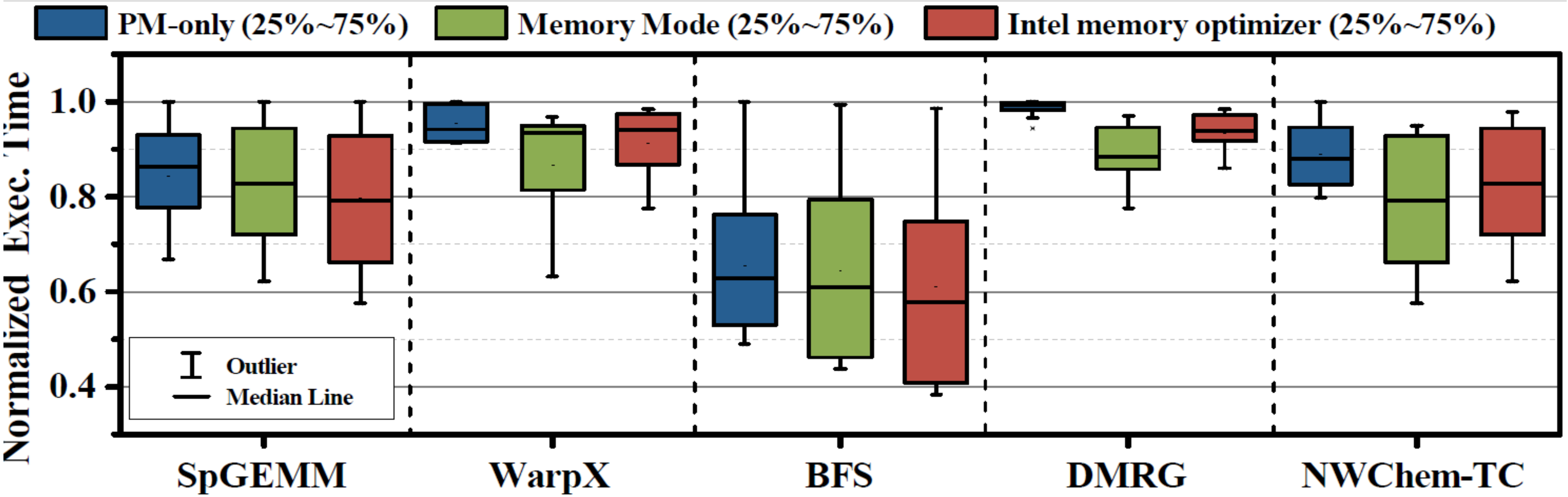
(b) OpenMP-based App. (SpGEMM)

```
for (A*B) in an application:
  Partition A into bins by rows
  Each bin has its size and NNZ
  #pragma omp parallel
   {T1: Compute NNZ of C
     Sync point 1
     T2: Compute values of C
     Sync point 2}
```

Tasks

- A thread works on a task instance
- The task is repeatedly executed
- Different task instances use different inputs (i.e., A and B)
- There is an implicit synchronization among threads

Traditional wisdom does not effectively guide page migration

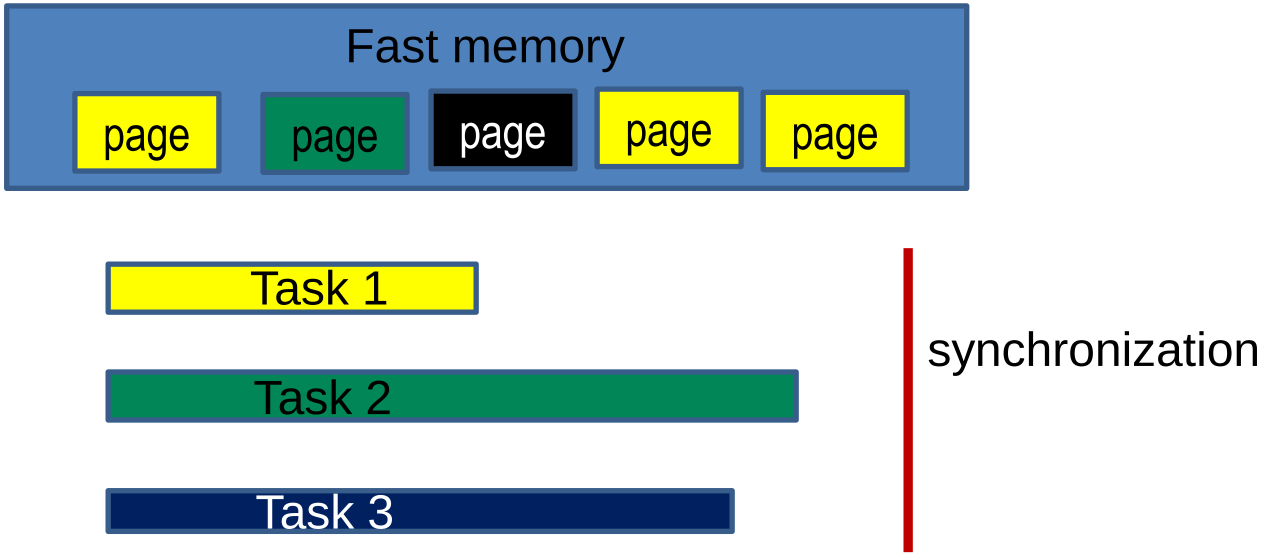


Task execution time and their variance. In the figure, wider box and longer whiskers indicate larger performance variance and worse load balance among tasks. Performance is normalized by the performance of PM-only

- Performance variance across tasks becomes much larger
  - Compared with PM-only, the memory mode and MemoryOptimizer increase the average coefficient of variation by 57.2% and 55.4%

Reasons why traditional wisdom cannot work

- Profiling-guided optimization (PGO) approaches periodically sample memory pages and track memory accesses to them



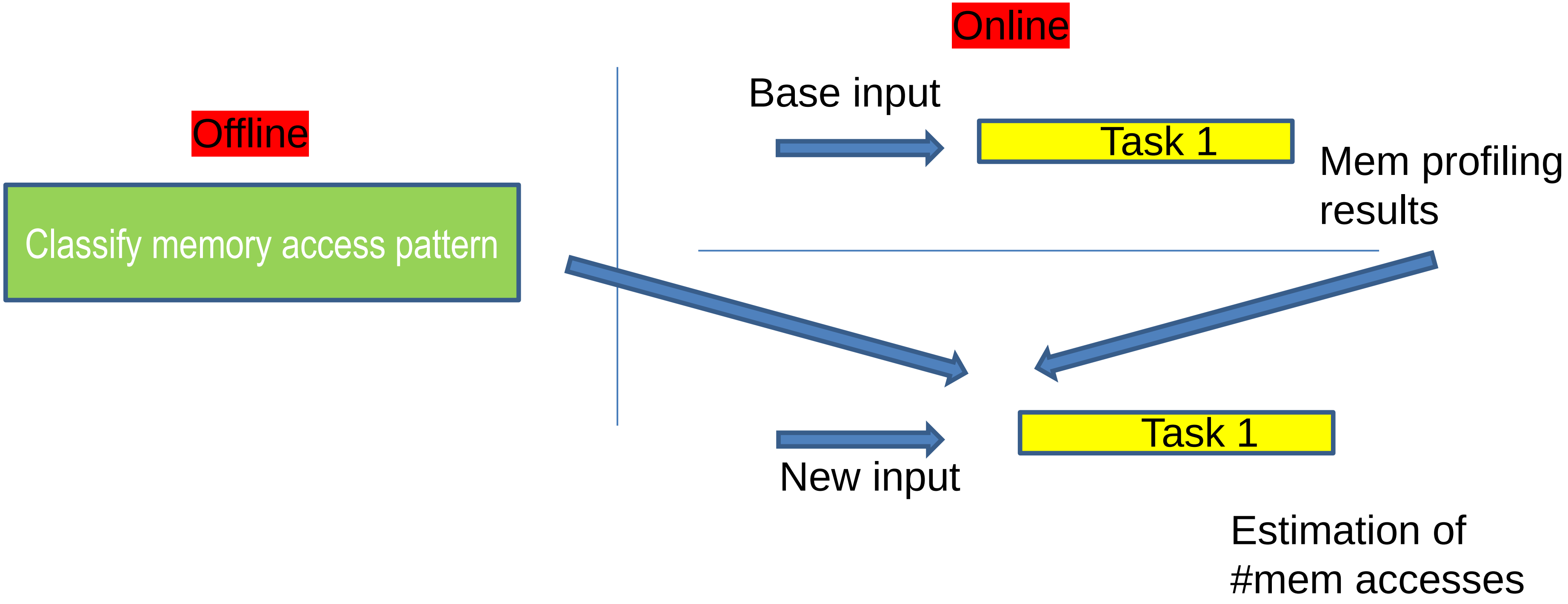
- Lack a view of “finishing all tasks fast” for high performance

Merchandiser: a load balance-aware data placement system for HM

- Input-aware memory access quantification
  - Estimating memory accesses to data objects for an input problem
- Performance modeling
  - Modeling application performance under various data placement on HM
- Load balance-aware runtime system

Input-aware memory access quantification

Basic idea



Online

Offline

Base input

Task 1

Mem profiling results

Classify memory access pattern

Task 1

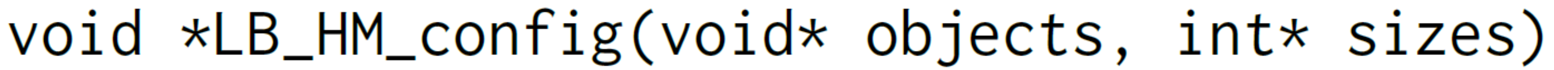
New input

Estimation of #mem accesses

Input-aware memory access quantification

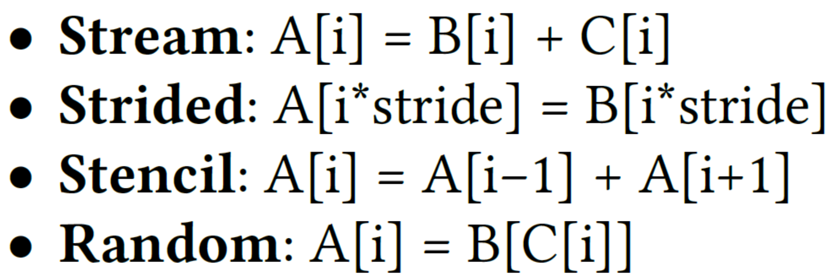
Classification of memory access patterns

- Specify data objects for management



```
void *LB_HM_config(void* objects, int* sizes)
```

- Object-level memory access pattern analysis



- **Stream**: A[i] = B[i] + C[i]
- **Strided**: A[i*stride] = B[i*stride]
- **Stencil**: A[i] = A[i−1] + A[i+1]
- **Random**: A[i] = B[C[i]]

Input-aware memory access quantification

Estimation of memory access count

$$esti\_mem\_acc = \frac{S_{new}}{S_{base} \times \alpha} \times prof\_mem\_acc$$

- Measure at the data object level during the first execution of the task (using the base input)
- α (a parameter) models the caching effects
  - α depends on memory access patterns
  - α is measured offline using microbenchmarks or analytical modeling
- For random access pattern and input-dependent stencil, refine α at runtime

Performance modeling

Goal: Modeling application performance under various data placement on HM

Basic idea

- Bound the performance prediction by the best (DRAM only) and the worst (PM only)
- Build upon $esti\_mem\_acc$ to scale the two performance bounds based on workload characterization

Simplifies our efforts to model memory access patterns but significantly improves usability

Performance modeling

$$T_{new\_hybrid} =$$

$$T_{new\_pm\_only} \times (1 - \boxed{r_{dram_{acc}}}) \times f(PMCs, r_{dram_{acc}})$$

$$+ T_{new\_dram\_only} \times r_{dram_{acc}}$$

$$\Big\downarrow$$

$$r_{dram_{acc}} = \frac{dram\_acc}{esti\_mem\_acc}$$

Performance modeling

$$T_{new\_hybrid} = T_{new\_pm\_only} \times \left(1 - r_{dram_{acc}}\right) \times f\left(PMCs, r_{dram_{acc}}\right) + T_{new\_dram\_only} \times r_{dram_{acc}}$$

Correlation function

- f(.) captures workload characterization
- PMCs: performance monitor counters

Performance modeling

Construction of the correlation function

$$f\left(PMCs, r_{dram_{acc}}\right)$$

- Input
  - – Some performance events measured using the base input
    - Performance events are selected based on their importance to performance prediction
    - LLC_MPKI, IPC, PRF_Miss, MEM_WCY, L2_LD_Miss, BR_MSP, VEC_INS, and L3_LD_Miss
- A statistical model
  - – Gradient boosted regressor (GBR)

Load balance-aware runtime system

Runtime system

- Extend the existing page migration mechanism
- Check the DRAM page constraint for each task before page migration

Merchandiser

Offline

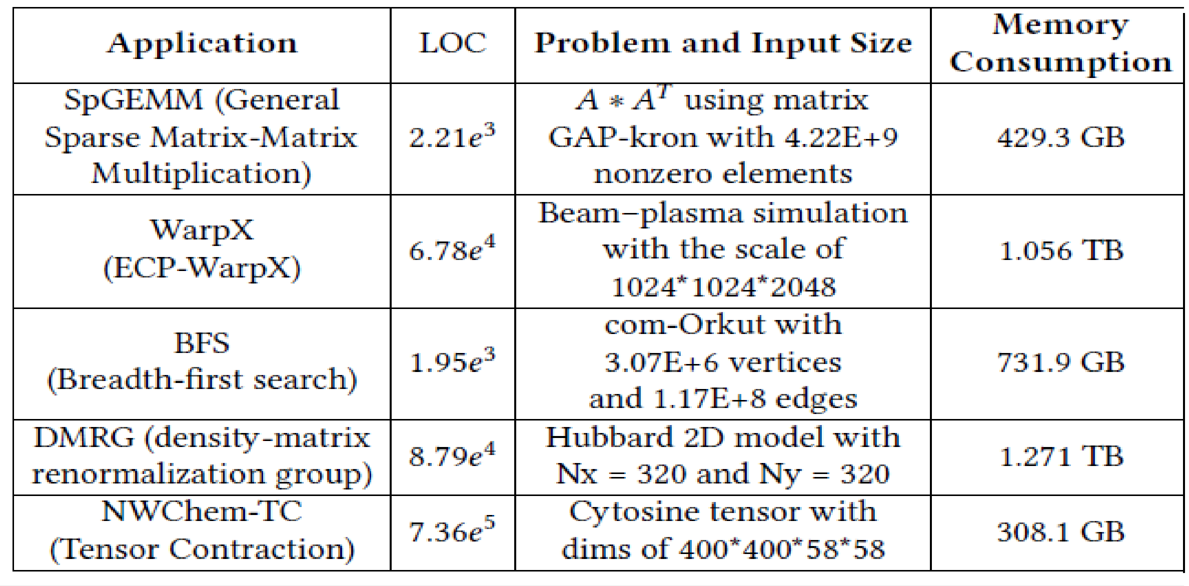
- Construction of f(.)
    - Happens only once
- Identify input-independent basic blocks
    - Happens only once per application
- Get memory access patterns
    - Happens only once per application

Online

- Collection of task information using the base input
- Online performance prediction with a new input

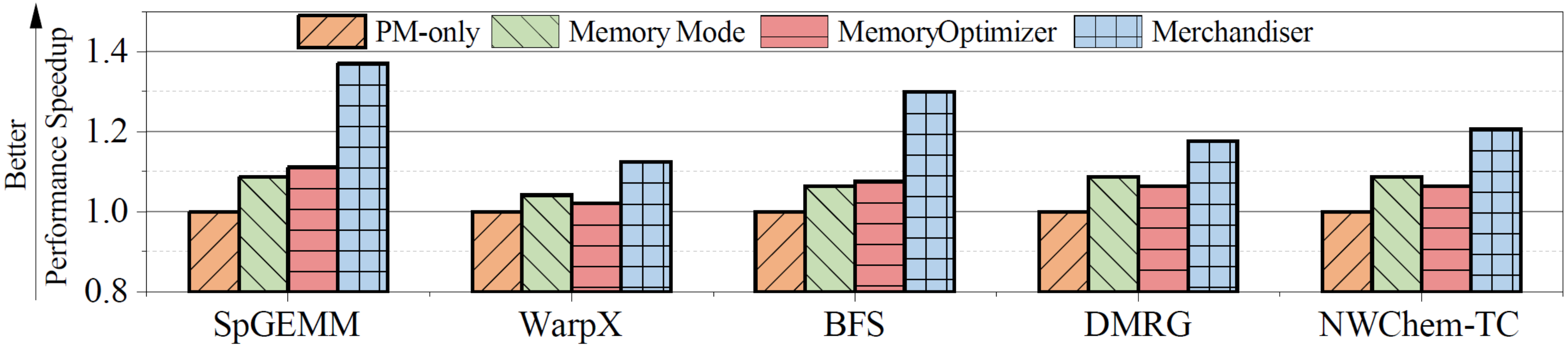
Performance evaluation

- Hardware
  - Dual-socket Intel Xeon Gold 6252N 24-core processors running Linux 5.17.0
  - 192 GB DRAM for fast memory in HM
  - 1.5 TB Intel Optane Persistent Memory for slow memory in HM
  - Single rank per node + OpenMP thread pinning
- Applications



Application	LOC	Problem and Input Size	Memory Consumption
SpGEMM (General Sparse Matrix-Matrix Multiplication)	$2.21e^3$	$A * A^T$ using matrix GAP-kron with 4.22E+9 nonzero elements	429.3 GB
WarpX (ECP-WarpX)	$6.78e^4$	Beam−plasma simulation with the scale of 1024*1024*2048	1.056 TB
BFS (Breadth-first search)	$1.95e^3$	com-Orkut with 3.07E+6 vertices and 1.17E+8 edges	731.9 GB
DMRG (density-matrix renormalization group)	$8.79e^4$	Hubbard 2D model with Nx = 320 and Ny = 320	1.271 TB
NWChem-TC (Tensor Contraction)	$7.36e^5$	Cytosine tensor with dims of 400*400*58*58	308.1 GB

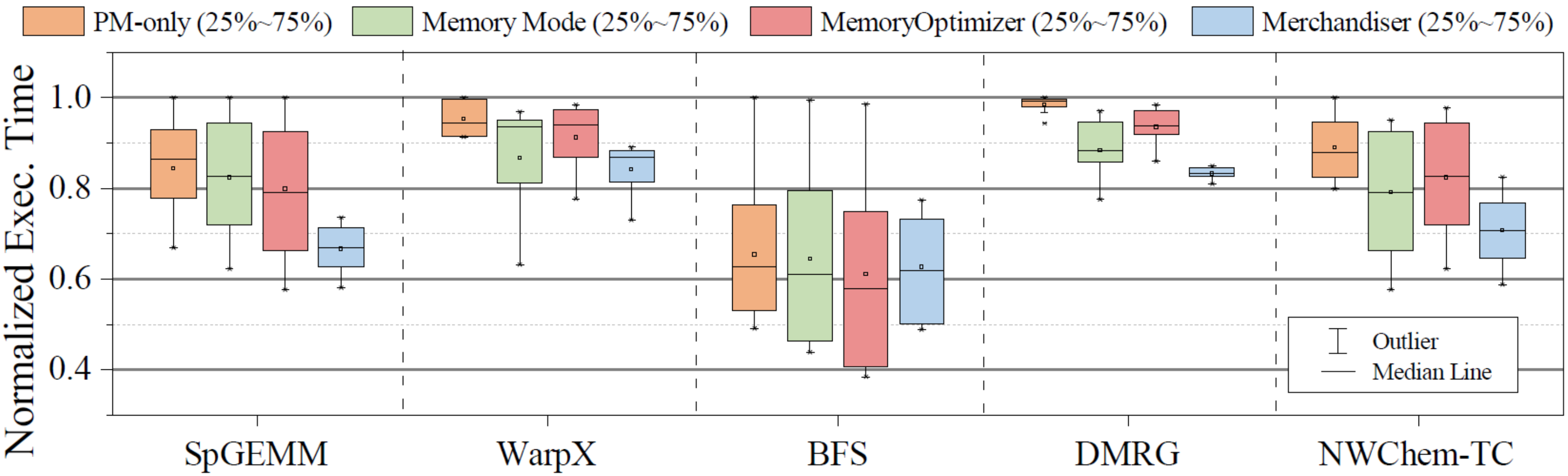
Performance evaluation – overall performance



Performance of Memory Mode, MemoryOptimizer, and Merchandiser, normalized to the performance of PM-only execution

- Merchandiser introduces 23.6%, 17.1%, and 15.4% performance improvement over PM-only, Memory Mode, and MemoryOptimizer respectively

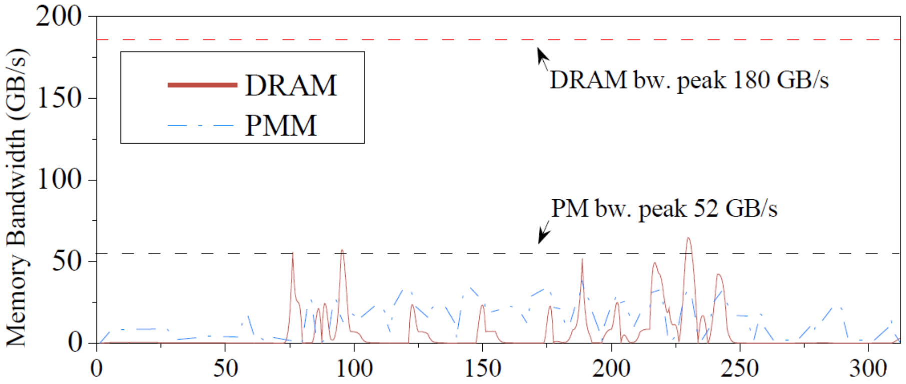
Performance evaluation – load balance



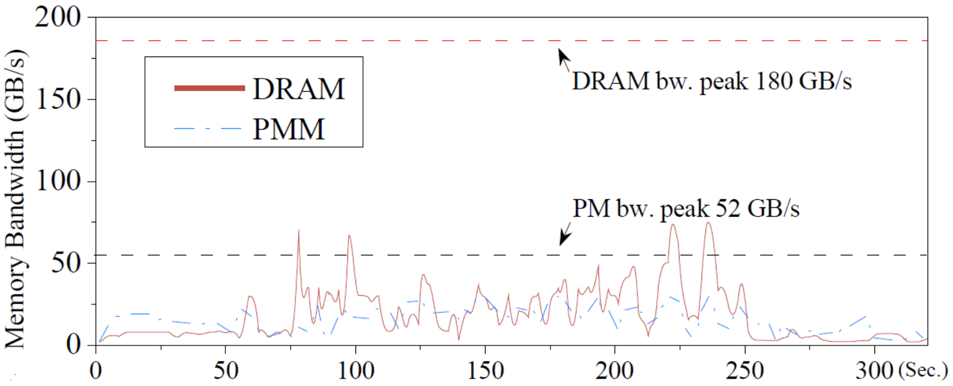
Task execution time and their variance

- Compared with Memory Mode and MemoryOptimizer, Merchandiser reduces the average coefficient of variation by 51.6% and 42.7% on average, respectively.

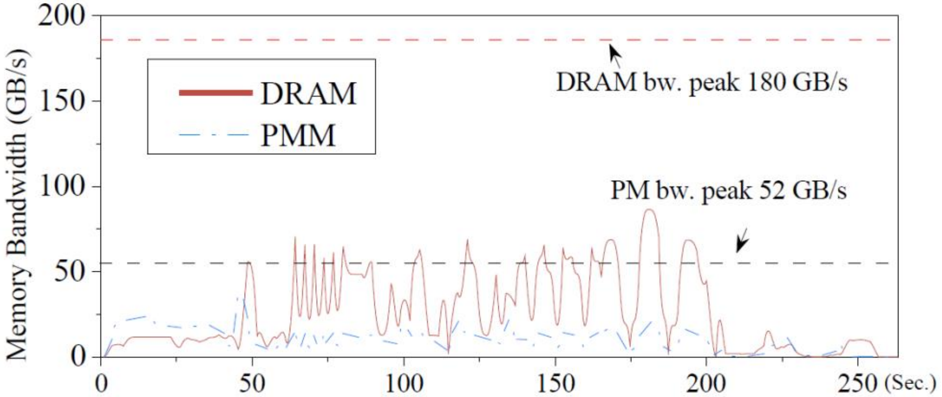
Performance evaluation – DRAM utilization



(a) Memory bandwidth usage of using Memory Mode for WarpX application



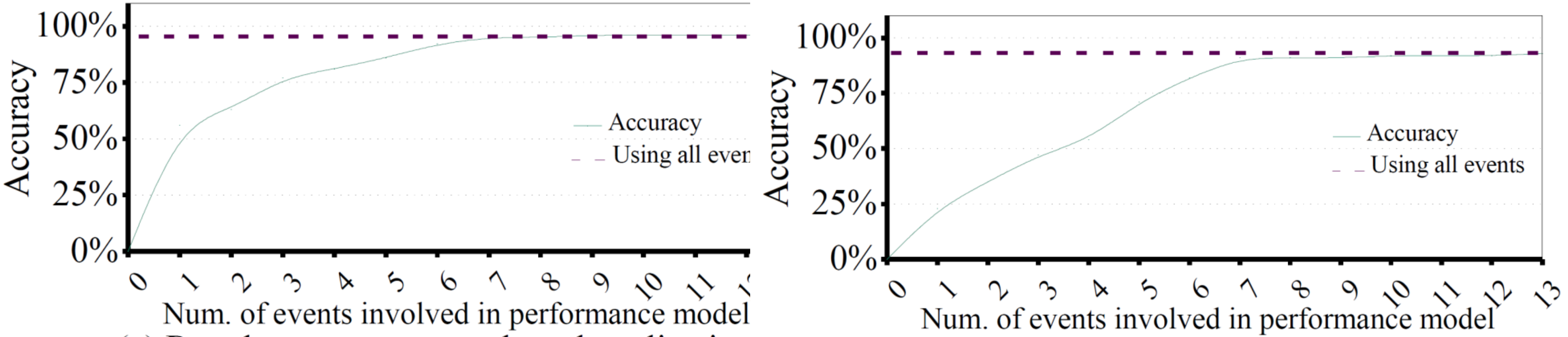
(b) Memory bandwidth usage of using MemoryOptimizer for WarpX application



(c) Memory bandwidth usage of using Merchandiser for WarpX application

- Compared with Memory Mode, Merchandiser increases average DRAM bandwidth usage from 5.98 GB/s to 24.31 GB/s, indicating the usage of fast memory is improved;
- Meanwhile, the average PM bandwidth usage is reduced from 13.74 GB/s to 9.97 GB/s, indicating the effectiveness of page migration in Merchandiser

Performance evaluation – event selection and modeling accuracy



(a) Regular access pattern-based application

(b) Irregular access pattern-based applications

Accuracy of the scaling function using different amounts of performance events as input

- Using the top 8 events, the model accuracy is 93.7% and 93.2% for regular- (i.e., WarpX and DMRG) and irregular- applications (i.e., SpGEMM, BFS, and NWChem-TC) respectively, which is close to the accuracy of using all events (94.8% and 94.1%).

Conclusions

- The traditional wisdom “migrating frequently accessed pages to fast memory leads to better performance” is not necessarily correct

- We introducing task semantics during memory profiling and migration to address the limitation of traditional wisdom

- We introduce a load balance-aware data placement system for HM
  - Performance modeling and runtime system