TCP: Thread Contention Predictor for Parallel Programs

Aparna Mandke, Bharadwaj Amrutur,
Y. N. Srikant, Chiranjeeb Bhattacharyya

Indian Institute of Science, Bangalore

Overview

- Motivation
- Thread Contention Indexes
- Application to determine a suitable cache configuration of last level cache on CMP
- Application to determine false data sharing
- Future Work

Motivation

- Chip Multicore widely used on desktops and server platforms
- Enable to exploit thread level parallelism present in an application
- Multithreaded applications have become common on chip multicores

Motivation

- Various threads in multithreaded applications share data among themselves, causing
  - significant coherence traffic
  - Also cause contention in NoC
- Cache design depends on contention caused by workload/data sharing properties of a workload

Indices characterizing data sharing properties

- Contention depends on amount of data shared by various threads
  - Depends on the number of read/write accesses
  - Amount of private and shared data present in an application
- Introduce 3 indices to capture these properties of data
  - Can be defined per cache line address
  - Per data object

Data sharing characterizing indices

- Sharing index – denotes average number of threads sharing an address/cache line address/object
- Contention index – quantizes interleaving of accesses done by different threads
- Popularity index – quantizes how often any address is accessed by threads

Sharing Index

$$H(P) = -\sum_{1 \leq i \leq T} P_i . \log(P_i)$$

$$SI = 2^{H(P)}$$

- Sharing index – denotes average number of threads sharing an address/cache line address/object

  entropy formulation is used to calculate SI

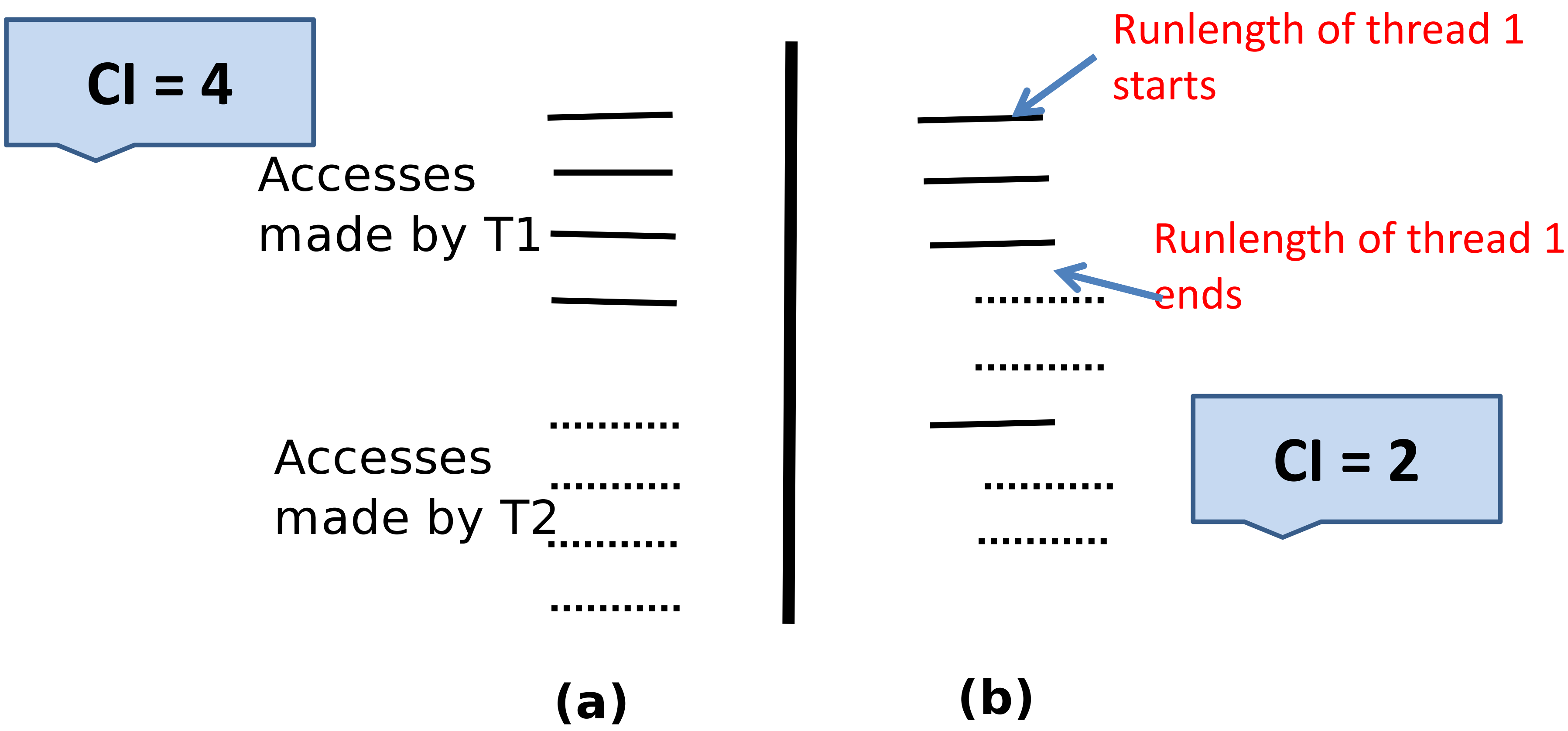
Sharing Index - Example

$$H(P) = -\sum_{1 \leq i \leq T} P_i . \log(P_i)$$

$$SI = 2^{H(P)}$$

- If address shared by 1 thread, its P_i = 1 and rest will be 0; SI = 1
- If equally shared by all threads, P_i = 1/T, SI = T

Contention Index



- Contention index – quantizes interleaving of accesses done by different threads
- CI is calculated as average runlength. Smaller runlength higher contention

Popularity Index

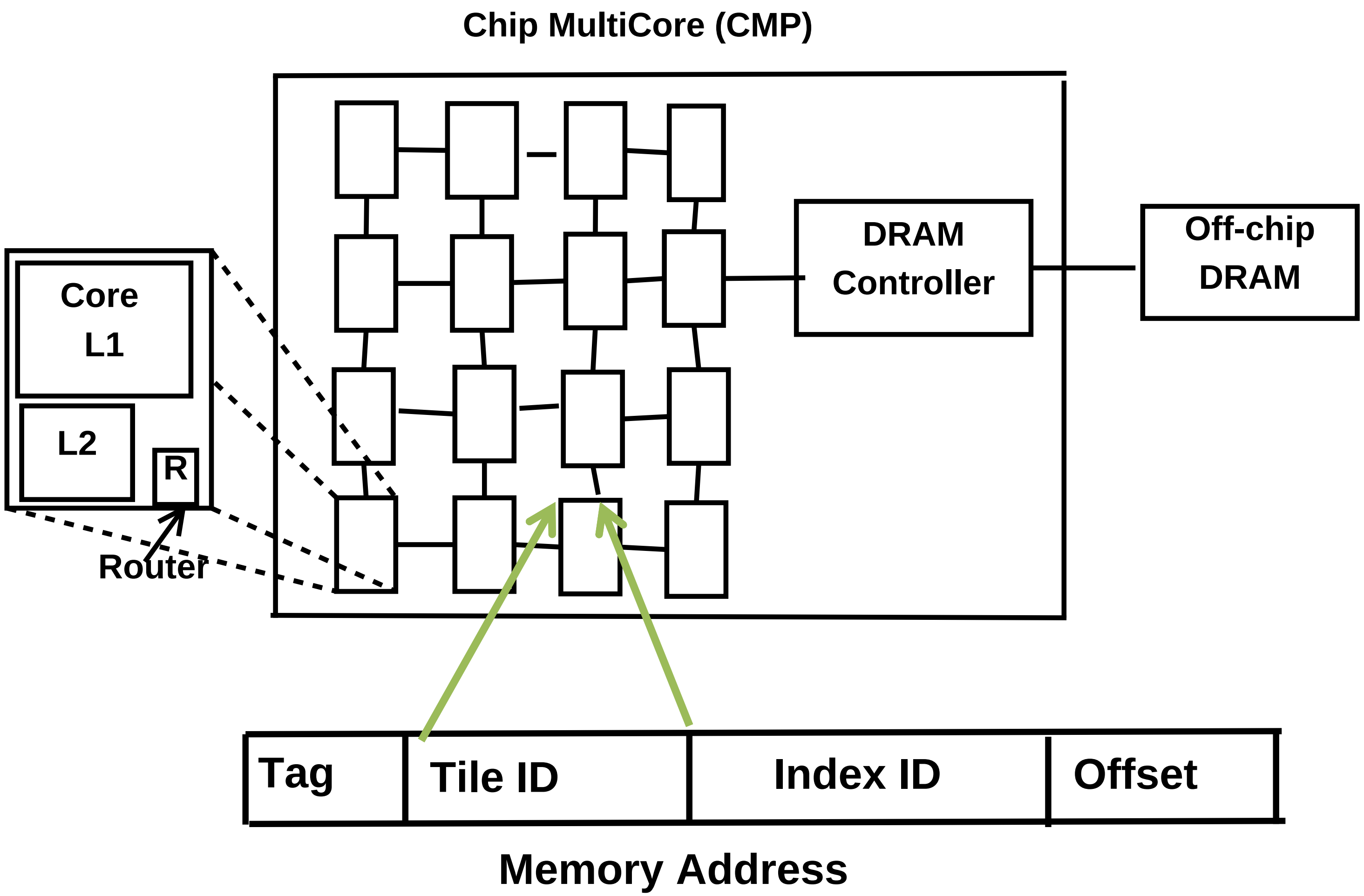
$$PI = \frac{N * SI}{CI}$$

Popularity Index – Address is more popular if accessed by multiple threads (SI), made many accesses (N) and if accesses are done in interleaved manner (smaller CI)

A Model to estimate suitable cache configuration among SNUCA and DNUCA

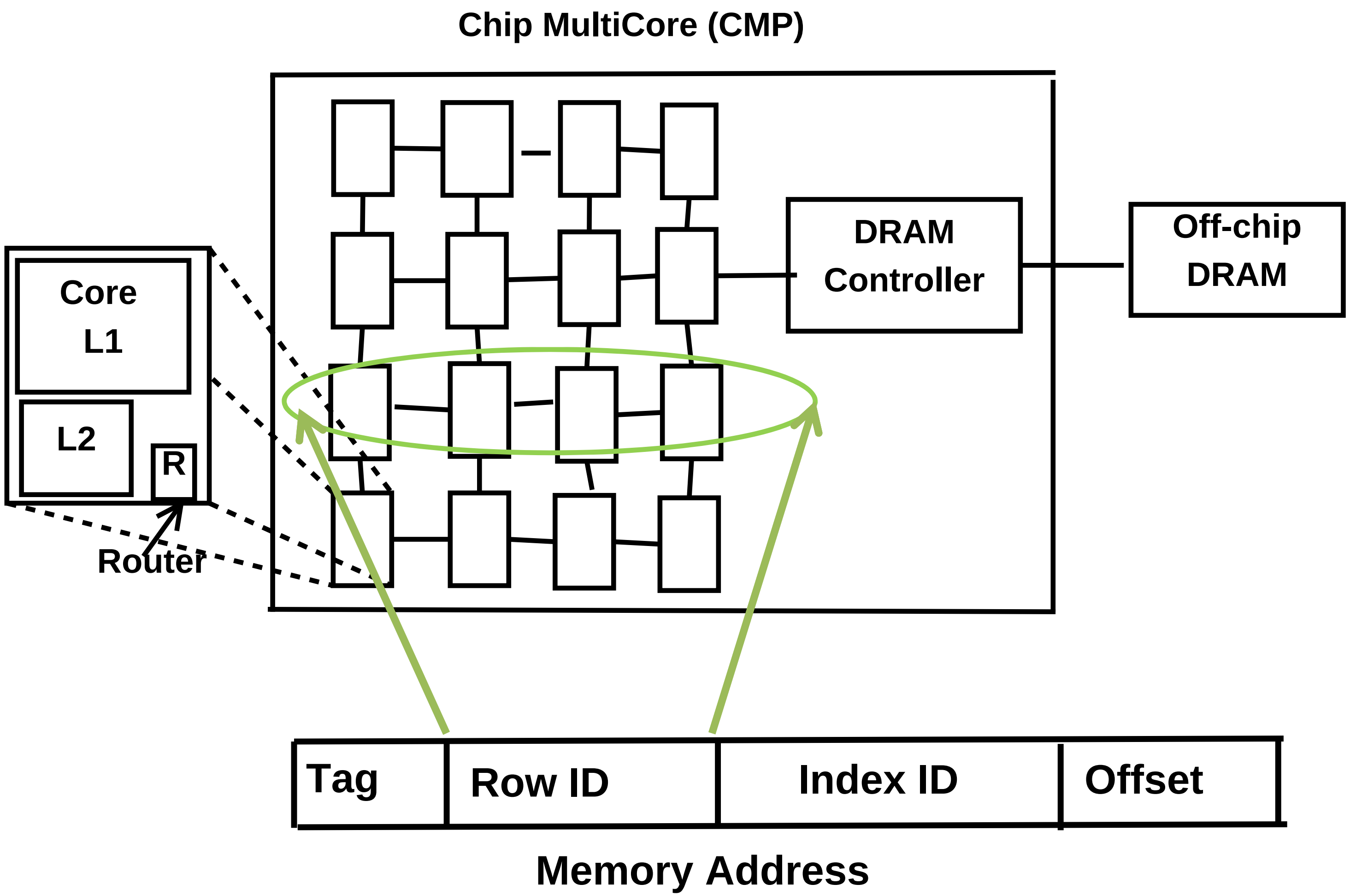
- Due to increasing number of cores, large caches have become inevitable
- SNUCA and DNUCA are two main cache access policies for accessing large caches
- While designing a new platform, a choice has to be made between SNUCA and DNUCA
- architectural simulation is time consuming

SNUCA



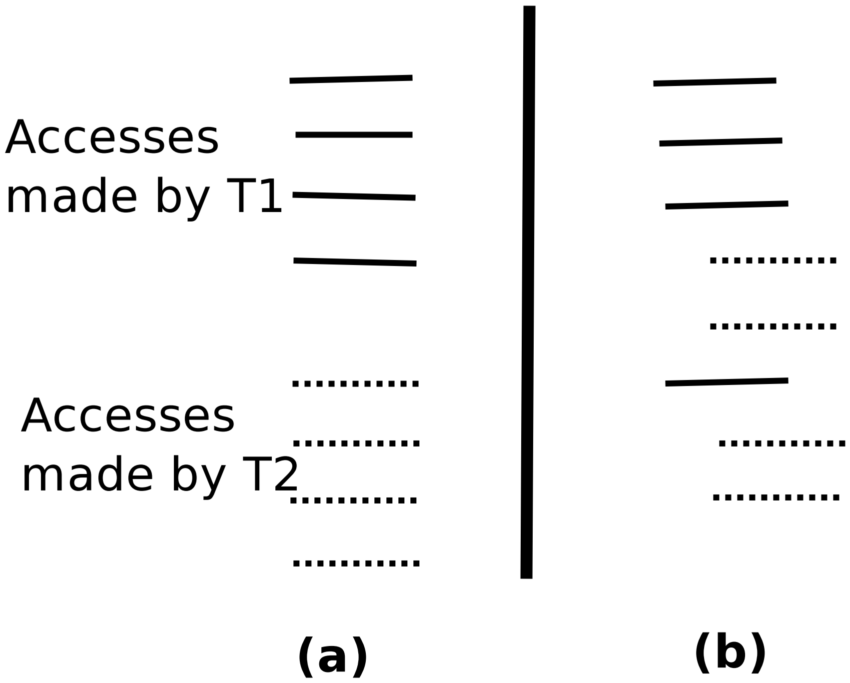
- In SNUCA, on L1 miss, all threads read data from same L2 slice – preferable if data is shared by many thread (CI small, SI large)

DNUCA



- In DNUCA, on L1 miss, data is read/migrates into nearest L2 slice – preferable for private data (CI large, SI small)

SNUCA Cost



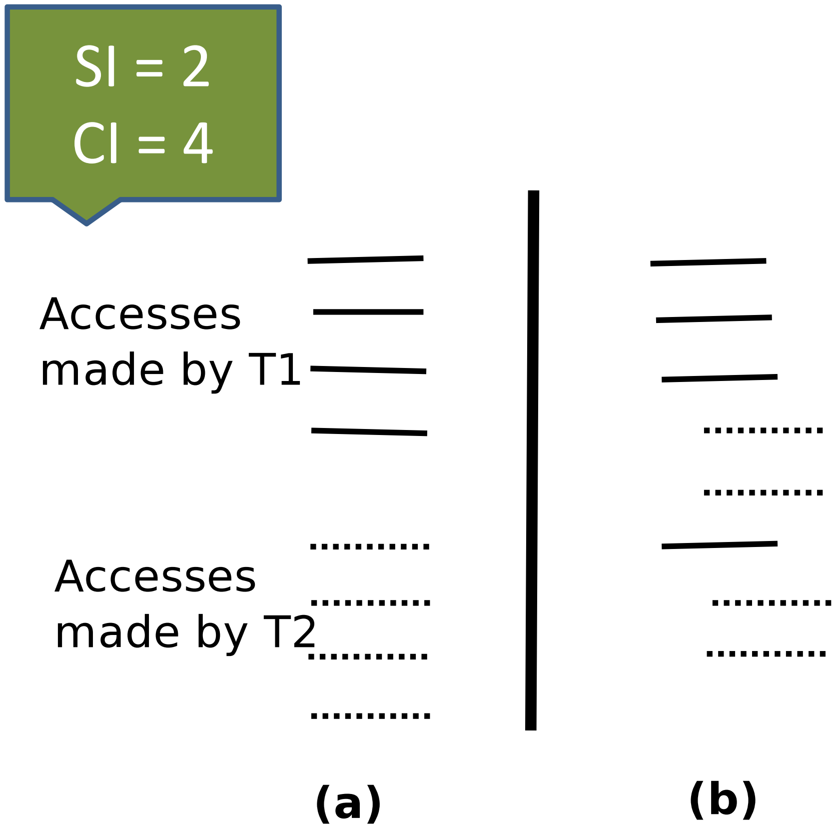
Assumptions:

- 2 threads execute on cores 0 and 1
- Distance betn t1 and four L2 slices is 1,2,3,4
- Distance from t1 to home location is 3 (D1_Home = 3)
- Distance from t2 to home location is 4 (D2_Home = 4)

SNUCAcost = 4*3 + 4*4

= 28

DNUCA Cost – Scenario (a)



First accesses of every runlength are searched in all L2 slices

PeerSearchCost = 2*(1+2+3+4) = 20

For rest, data is assumed in the nearest L2 slice

if SI == 1 || CI > 2

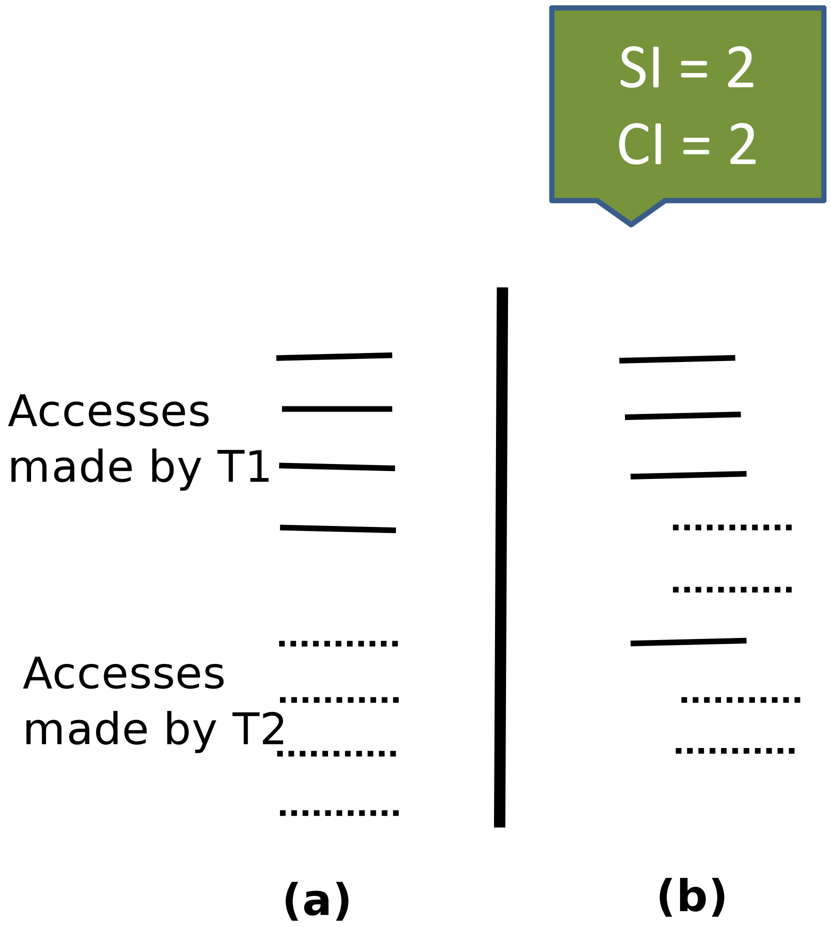
Here, SI = 2, CI = 4

nearSearchCost = 3*1*2 = 6

Total DNUCAcost = 26

DNUCA is preferable

DNUCA Cost – Scenario (b)



First accesses of every runlength are searched in all L2 slices

PeerSearchCost = 4*(1+2+3+4) = 40

For rest, avg of all L2 slice distances is taken

if SI != 1 || CI < 3

Here, SI = 2, CI = 2

avgSearchCost = 2.5*(2+1+1) = 10

Total DNUCAcost = 50

PeerSearch major cost component

SNUCA is preferable

Method

- Execute application once with SNUCA cache access policy
- During simulation, gather per thread runlength information for every cache line address
- At the end of simulation, determine SI, CI for every address and also total number of access done by each thread to each cache line address
- Calculate SNUCA cost and DNUCA cost

Notations used in analysis

Table 8.1: Table gives the meaning of various terms used in CPP model

Parameter	Description
t_i	thread executing on core i
T	total # of threads present in an application
A_{ij}	total # of accesses made by thread i to cache line address (CLA) j
N	total # of CLAs
K	runlengths of size $0, 1, ..K$ tracked during one-time simulation on SNUCA. Runlengths of size equal to and greater than K are counted by $(K-1)^{th}$ array entry.
r_{ijk}	# of times thread i exhibits runlength of size k for an address j
D_{ip}	distance between L1 in tile i and L2 slice in tile p where data can be cached in DNUCA
P	total # of peer L2 slices in a bankset in DNUCA
$D_{i\_Nearest}$	distance between L1 in tile i and its nearest L2 slice where address can be cached in DNUCA
$D_{i\_Average}$	average distance between L1 in tile i and all L2 slices in a bankset where address can be cached in DNUCA
D_{i_Home}	distance between L1 in tile i and "home" L2 slice, where data is cached in SNUCA
$SNUCA_{cost}$	Time spent in transit for SNUCA
$DNUCA_{cost}$	Time spent in transit for DNUCA

SNUCA Cost

$$SNUCA_{cost} = \sum_{0 \leq i < T} \sum_{0 \leq j < N} A_{ij}.D_{i\_Home}$$

- For a cache line addresses, its SNUCA cost depends on distance of a thread from home location of an address and # of accesses made by that thread
- Sum of SNUCAcost for all cache line addresses gives total SNUCAcost

DNUCA Cost – PeerSearch Component

$$PeerSearchCt_{ij} = \sum_{0 \leq p < P} \sum_{0 \leq k < K} r_{ijk} * D_{ip}$$

- For first accesses of each runlength, all peer L2 slices are searched

DNUCA Cost – NearSearch Component

$$NearSearchCt_{ij} = (A_{ij} - \sum_{0 \leq k < K} r_{ijk}) * D_{i\_Nearest}$$

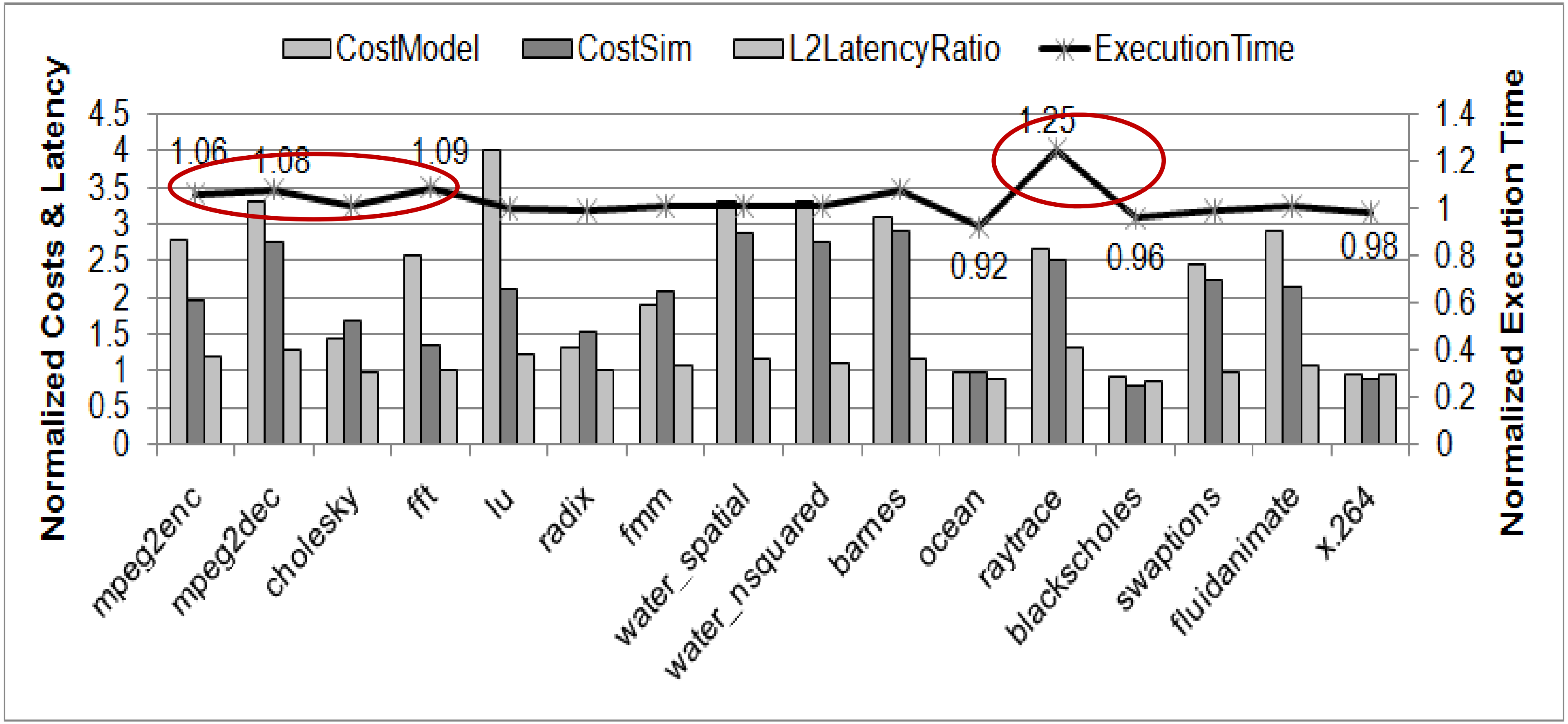
- All rest of accesses in a runlength, are assumed to be done to the nearest L2 slice if SI == 1 or CI > 2

DNUCA Cost – AvgDistanceSearch Component

$$AvgDistanceSearchCt_{ij} = (A_{ij} - \sum_{0 \leq k < K} r_{ijk}) * D_{i\_Average}$$

- For all remaining accesses in a runlength, average distance if used if SI != 1 or CI <= 2
- Experimentally observed that, applications with lot of data sharing – PeerSearch cost dominates

Model Predictions – nThr = 16

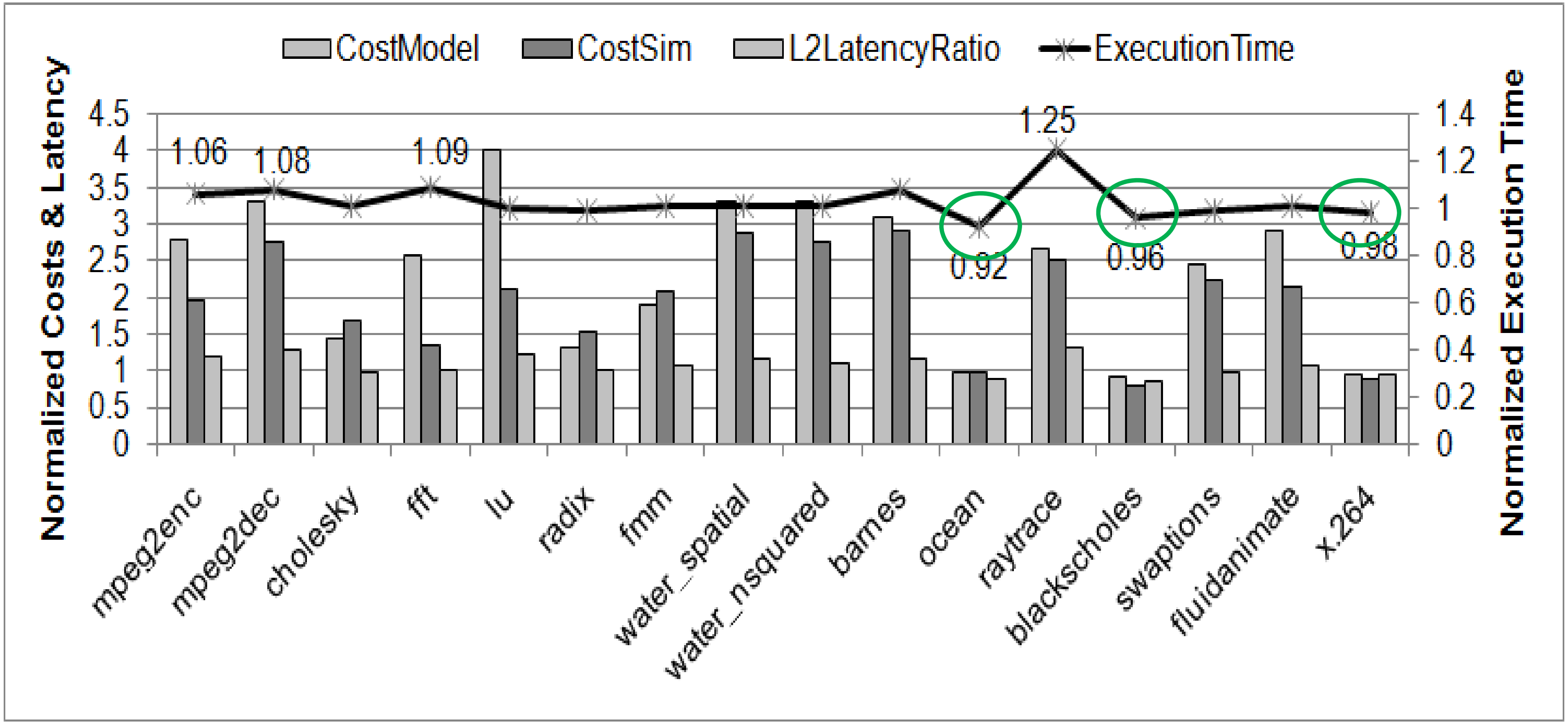


- Mpegenc, mpegdec, raytrace, fft execution time degrades with DNUCA.
  - Either due to majority of accesses with SI > 1 or with CI < 3
- LU has 99% access with SI = 1, however 100% accesses have CI < 3. It has limited reuse of data cached in L2. So SNUCA is preferable

SI, CI Distribution

	CLA Distribution				Costs determined by CPP			Costs determined by Simulator	
App	SI = 1	SI > 1	CI < 3	CI >= 3	Peer	Near	Avg	Peer	Near
mpgenc	0.47	0.53	0.63	0.39	0.92	0.06	0.02	0.85	0.15
Raytrace	0.64	0.36	0.89	0.11	0.88	0.07	0.05	0.89	0.11

Model Predictions – nThr = 16

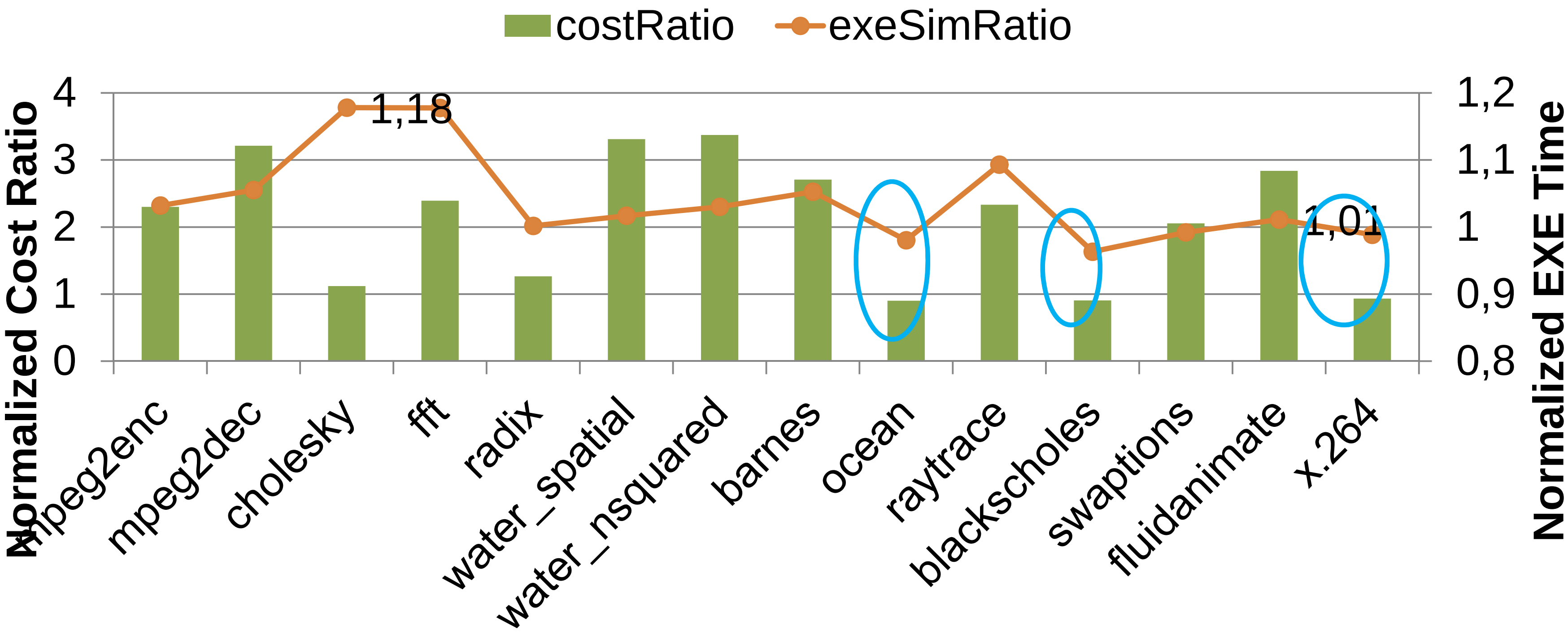


- For ocean, blackscholes and x.264 DNUCA preferable
- X.264 poor thread scalability, 97% accesses have SI = 1
- Ocean and blackscholes have 95% accesses with SI = 1

SI, CI Distribution

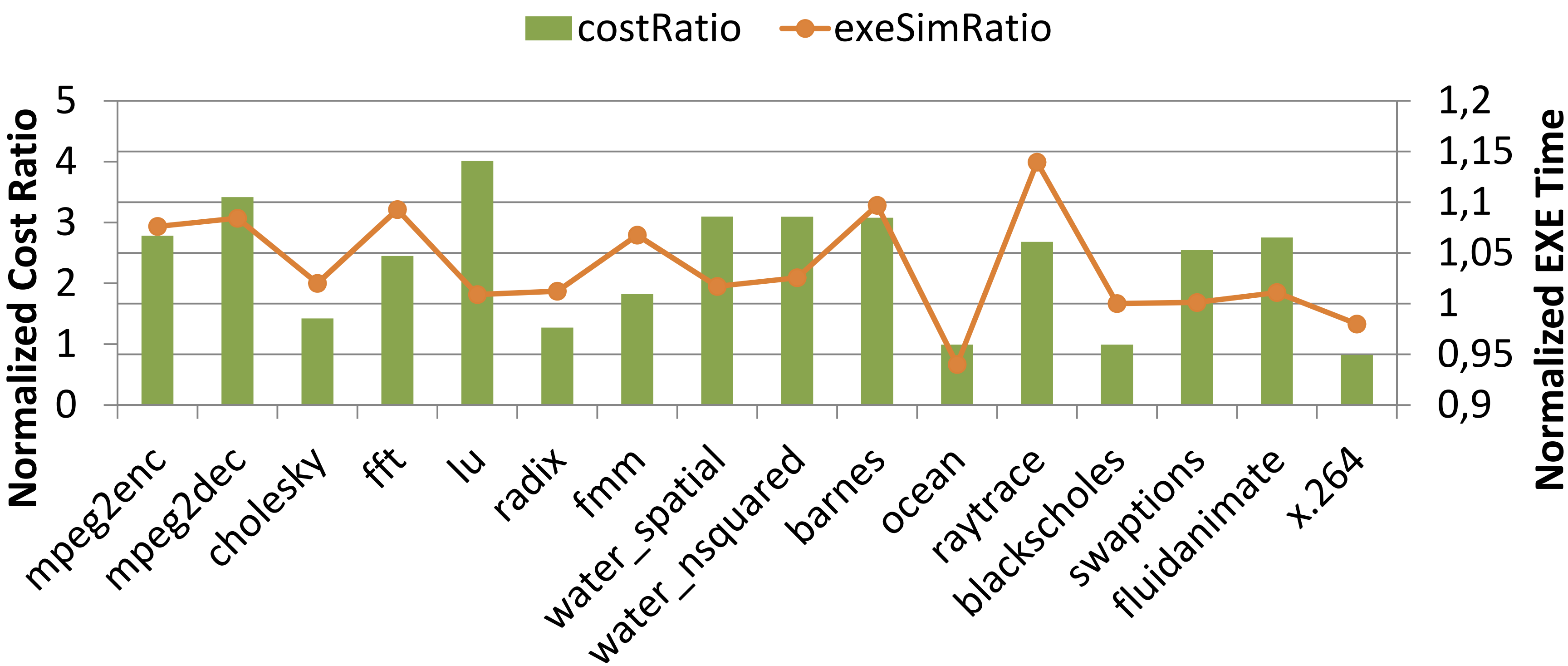
	CLA Distribution				Costs determined by CPP			Costs determined by Simulator	
App	SI = 1	SI > 1	CI < 3	CI >= 3	Peer	Near	Avg	Peer	Near
Blackscholes	0.95	0.05	0.07	0.93	0.11	0.89	0	0.01	0.99
Ocean	0.95	0.05	0.19	0.81	0.39	0.6	0.01	0.47	0.53

Model Predictions – nThr = 8



Model predictions are correct on varying # of threads

nThr = 16, L2 slice = 256KB



L2 slice of 256KB is used. Model predictions are correct for various cache sizes

Predictions remain same even on changing cache size

Application to determine false data sharing

$$PI = \frac{N * SI}{CI}$$

- If two data objects, shared by lot of threads are allocated in the same cache line then cause significant amount of coherence traffic

- Use popularity index of a cache line to filter such cache lines

Application to determine false data sharing

$$PI = \frac{N * SI}{CI}$$

- Cache line addresses with significantly higher value of popularity index were filtered
- Instruction addresses accessing such cache lines are tracked
- False sharing if accesses are made through procedures belonging to different tasks!!

Code Snippet from Raytrace

```
Typedef struct gmem {
    int nprocs;
    int pid;
    int rid;
    |
    barrier_t start;
    lock_t pidlock
    lock_t ridlock;
    lock_t memlock;
    lock_t (wplock)[MAX_PROCS]
    |
} GMEM;
```

```
Typedef struct gmem {
    int nprocs;
    int pid;
    lock_t pidlock;
    char PAD1[40];
    int rid;
    lock_t ridlock;
    char PAD2[40];
    |
    barrier_t start;
    char PAD3[60];
    lock_t memlock;
    char PAD4[60];
    lock_t (wplock)[MAX_PROCS]
    |
} GMEM;
```

EDP and Execution time savings

App	Execution Time	L2 Latency	EDP
Raytrace	10.7	23.7	18.9
Barnes	2	4.2	3.77

- **Different task information can be obtained using program annotations or different tasks can be declared in separate file**
- **Accesses to the same cache line are detected from different tasks, programmer can be alerted**
- **Over padding may increase working set size of an application !!**

Conclusion

- Proposal of sharing index to quantify average no. of threads sharing an object
- Contention index to quantify contention caused by various threads
- Popularity index to quantify significance of an object in a program execution
- Application to determine a suitable last level cache configuration on CMP
- Use to determine possible false data sharing in a cache line

Thank you and Questions