Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors?

Yunlian Jiang

Eddy Z. Zhang

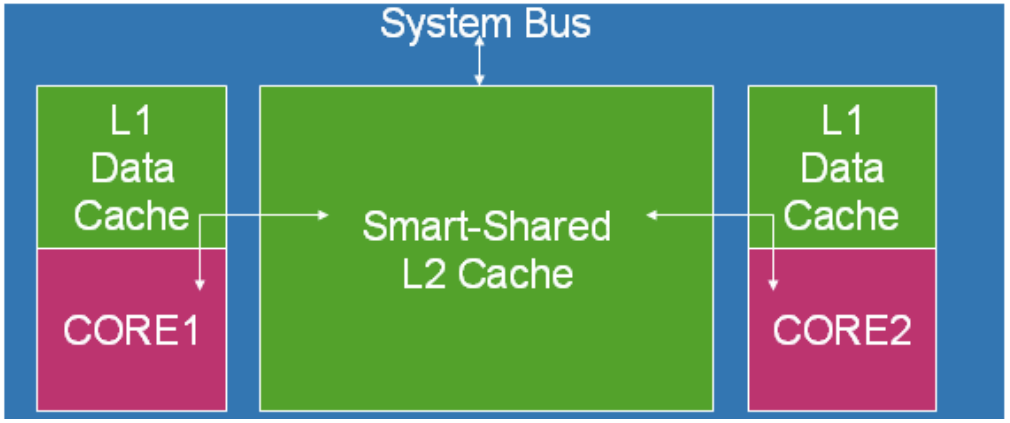
Kai Tian

Xipeng Shen (presenter)

Department of Computer Science
The College of William and Mary, VA, USA

Cache Sharing

- A common feature on modern CMP

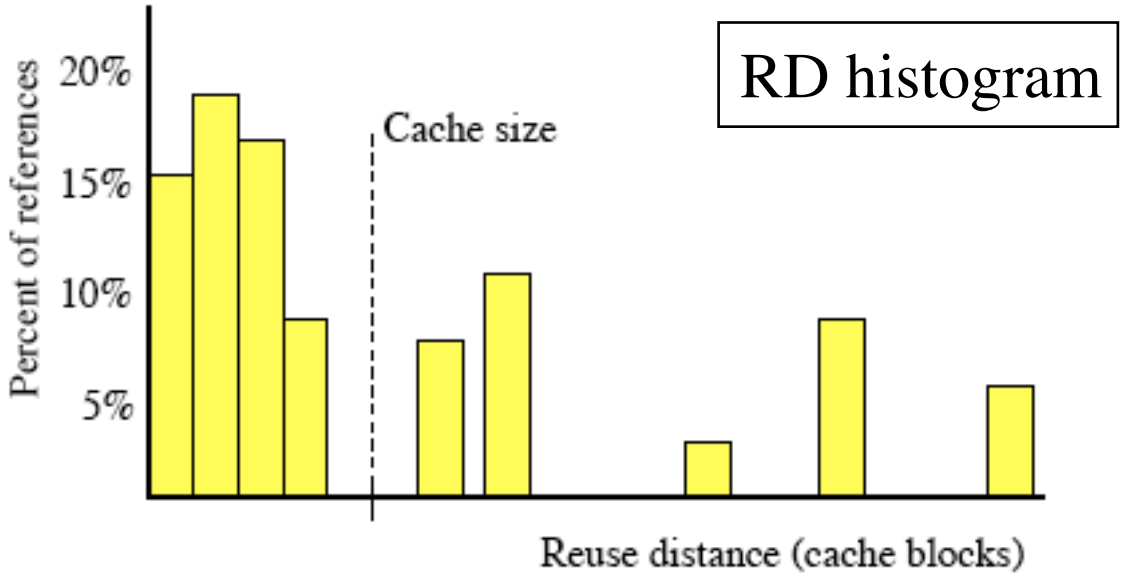


Data Locality

- Extensively studied for uni-core processors
- Two classes of metrics
  - At hardware level
    - E.g., cache miss rate
  - At program level
    - E.g., reuse distance

Reuse Distance (RD)

- Def: # of **distinct** data between two adjacent ref. to a data element
  - E.g. b c a a c b rd=2



Reuse Distance (RD)

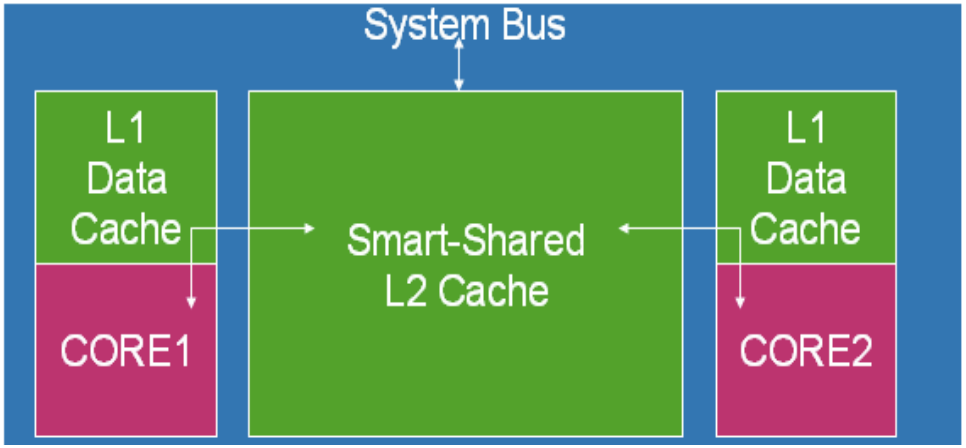
- Def: # of distinct data between two adjacent ref. to a data element
  - E.g. b c a a c b rd=2
- Appealing properties
  - Hardware-independence
  - Accurate, point to point
  - Cross-input predictable
  - Bounded value---data size

Many Uses of Reuse Distance

- Cross-arch performance prediction [Marin +:SIGMETRICS04,Zhong+:PACT03]
- Model reference affinity [Zhong+:PLDI04]
- Guide memory disambiguation [Fang+:PACT05]
- Detect locality phases [Shen+:ASPLOS04]
- Software refactoring [Beyls+:HPCC06]
- Model cache sharing [Chandra+:HPCA05]
- Study data reuses [Ding+:SC04,Huang+:ASPLOS05]
- Insert cache hints [Beyls+:JSA05]
- Manage superpages [Cascaval+:PACT05]

Complexity Caused by Cache Sharing

- Data locality is not solely determined by a process itself
  - Accesses by its co-runners need to be considered



Questions to Answer

- Is reuse distance applicable for locality characterization on CMP?
- What are the new challenges?
- Are these challenges addressable?

Outline

- Complexities in extending reuse distance model to CMP
  - Loss of hardware-independence
  - A chicken-egg dilemma for performance prediction
- Addressing the issues for some multithreading app.
  - A probabilistic model to derive reuse distance in co-runs
  - Evaluation

Terms

- Concurrent reuse distance (CRD)
  - # of distinct data accessed by all co-runners between two adjacent ref. to a data element.
- Standalone reuse distance (SRD)
  - # of distinct data accessed by the current process between two adjacent ref. to a data element.
- Example

SRD = 3; CRD =3+2=5

P1: a b b c d a
P2: p q p q

Distinctive Property of CRD

Dependance on relative running speeds of co-runners.

- Example

Mem. references by P_1 a b c b a

SRD = 2
CRD = 2 + x

r = speed(P_2)/speed(P_1)
The larger r is, the greater x tends to be.

Two Implications

- First, CRD is hard to measure in real programs.
  - Instrumentation changes relative speeds

sharers	1 & 4	2 & 4	3 & 4	1 & 5	2 & 5	3 & 5	1 & 6	2 & 6	3 & 6
r	0.40	0.59	0.25	0.44	0.55	0.43	0.37	0.48	0.30
r'	0.77	0.77	0.87	0.93	0.89	0.93	0.99	1.11	0.88
changes (%)	92.5	30.5	248	111	61.8	116	168	131	193

* programs: 1-ammp; 2-art; 3-mcf; 4-bzip2; 5-gzip; 6-mesa.

relative speed — original: $r = IPC_i/IPC_j$

after instrumentation: $r' = IPC'_i/IPC'_j$

changes of relative speed: $|r-r'|/r$

Two Implications (cont.)

- Second, CRD loses hardware-independence.
  - Relative speeds change across architectures.

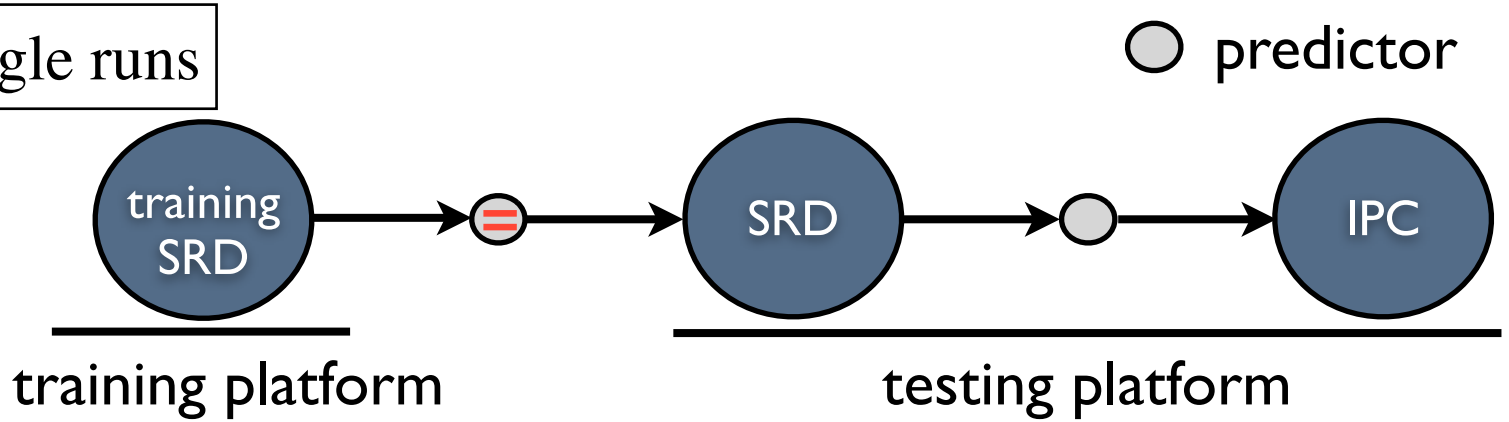
sharers	1 & 4	2 & 4	3 & 4	1 & 5	2 & 5	3 & 5	1 & 6	2 & 6	3 & 6
r	0.40	0.59	0.25	0.44	0.55	0.43	0.37	0.48	0.30
r'	0.40	0.48	0.3	0.57	0.72	0.54	0.37	0.48	0.31
changes (%)	0	18.6	20.0	29.5	30.9	25.6	0	0	3.3

* r: on Xeon E5310; r': on Xeon 7120M.
* programs: 1-ammp; 2-art; 3-mcf; 4-bzip2; 5-gzip; 6-mesa.

- Consequence
  - Cross-arch. perf. pred. becomes hard for co-runs

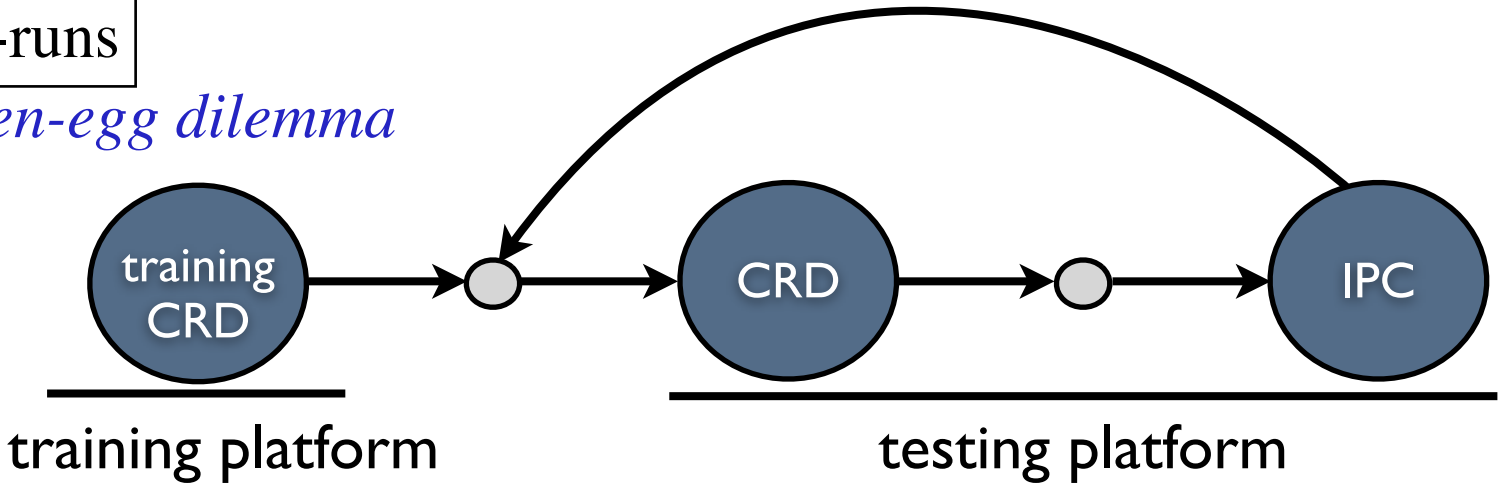
Cross-Arch. Performance Prediction

for single runs

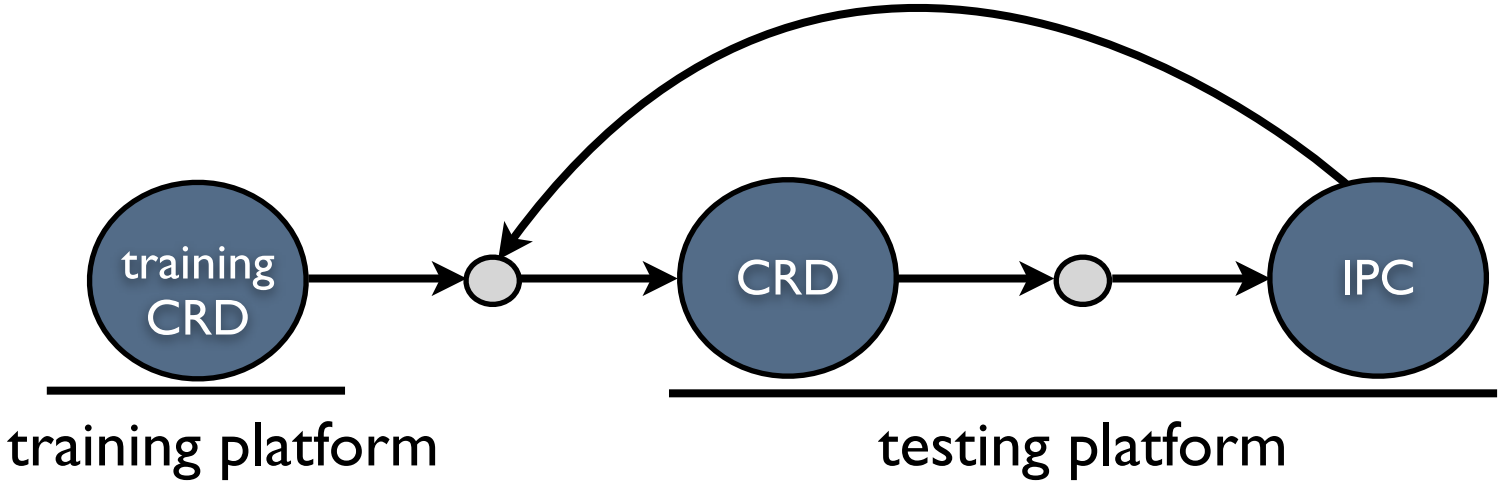


for co-runs

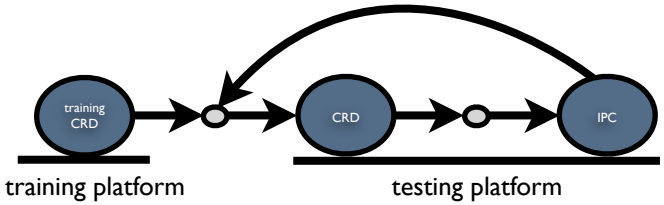
chicken-egg dilemma

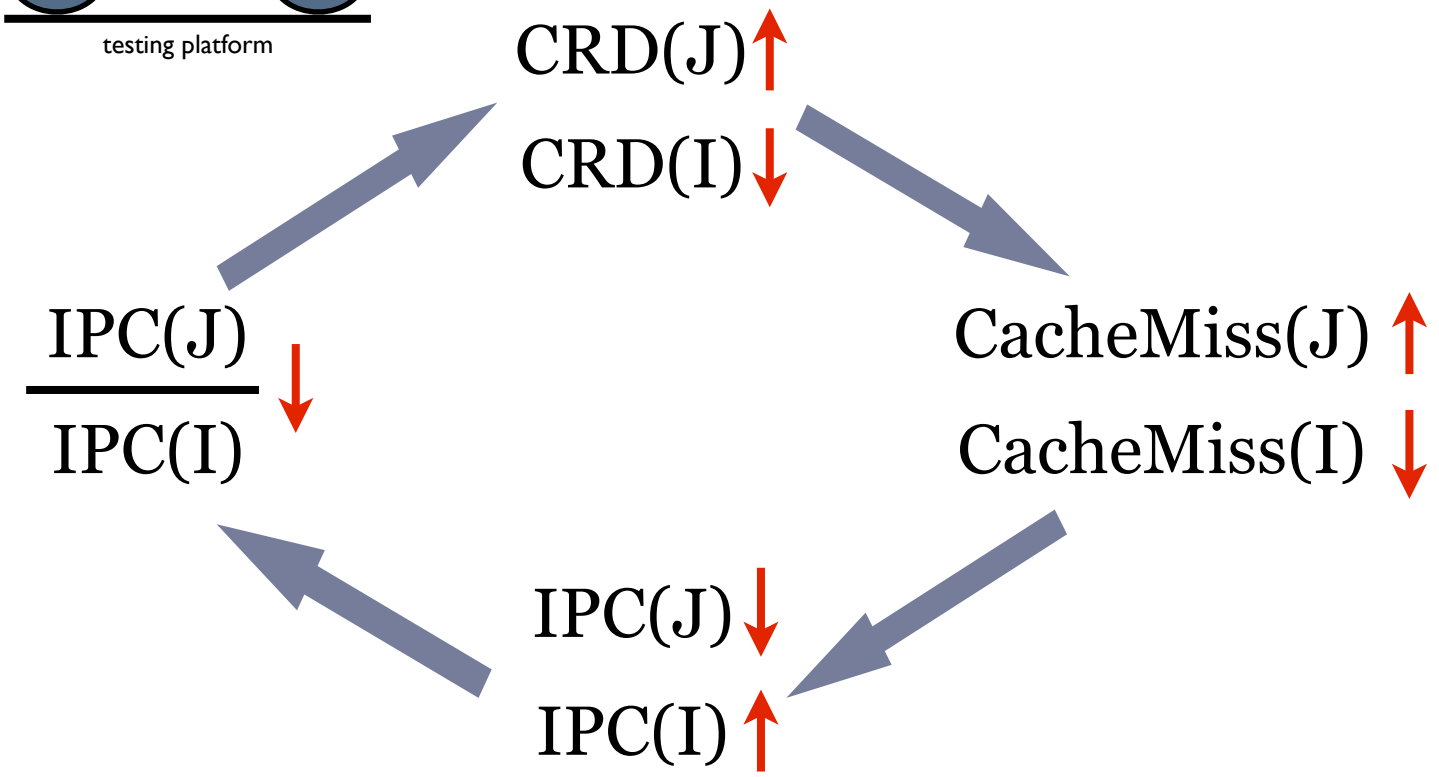


Iterative Approach Not Applicable



Iterative Approach Not Applicable





Outline

- Complexities in extending reuse distance model to CMP
  - Loss of hardware-independence
  - A chicken-egg dilemma for performance prediction
- Addressing the issues for some multithreading app.
  - A probabilistic model to derive reuse distance in co-runs
  - Evaluation

Favorable Observations

- From a systematic study [Zhang+:PPoPP'10] on PARSEC non-pipelining multithreading benchmarks
  - All parallel threads of an app. conduct similar computations
  - Uniform relations among threads.

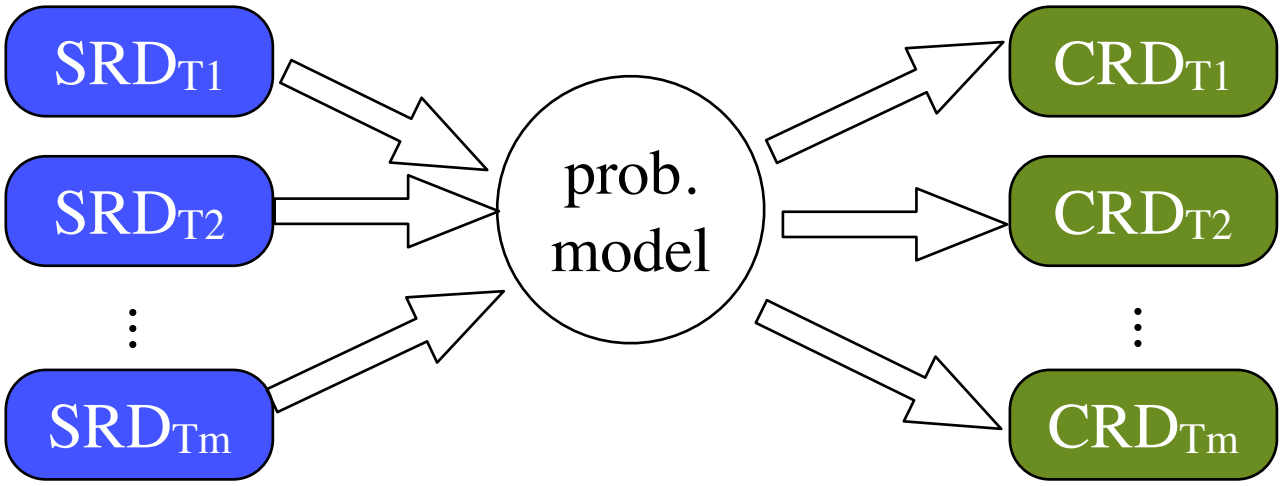
They hold across arch, inputs, # of threads, thread-core assignments, program phases.

Implication

- Relative speeds among threads tend to remain the same across arch. and inputs.

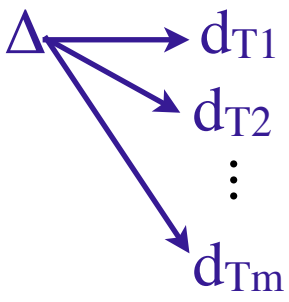
input	machine	IPC(thread 0)/IPC(thread 1) for programs						
		blackscholes	bodytrack	canneal	facesim	fluidanimate	streamcluster	swaptions
simlarge	7120M	1.00	0.96	1.00	1.00	1.00	1.00	1.00
	E5310	1.00	1.00	1.00	1.00	0.99	1.00	1.00
native	7120M	1.00	0.92	1.00	1.00	0.99	1.00	1.00
	E5310	1.00	0.99	1.00	1.01	0.99	1.00	1.00
changes by arch. (%)		0	5.9	0	0	0.5	0	0
changes by input (%)		0	2.6	0	0.5	0.5	0	0

An Efficient Way to Estimate CRD



Two Steps

(1) **Δ ⟶ d** (# of distinct data accessed)



$CRD_{T1} = d_{T1} + d_{T1} + \ldots + d_{Tm}$

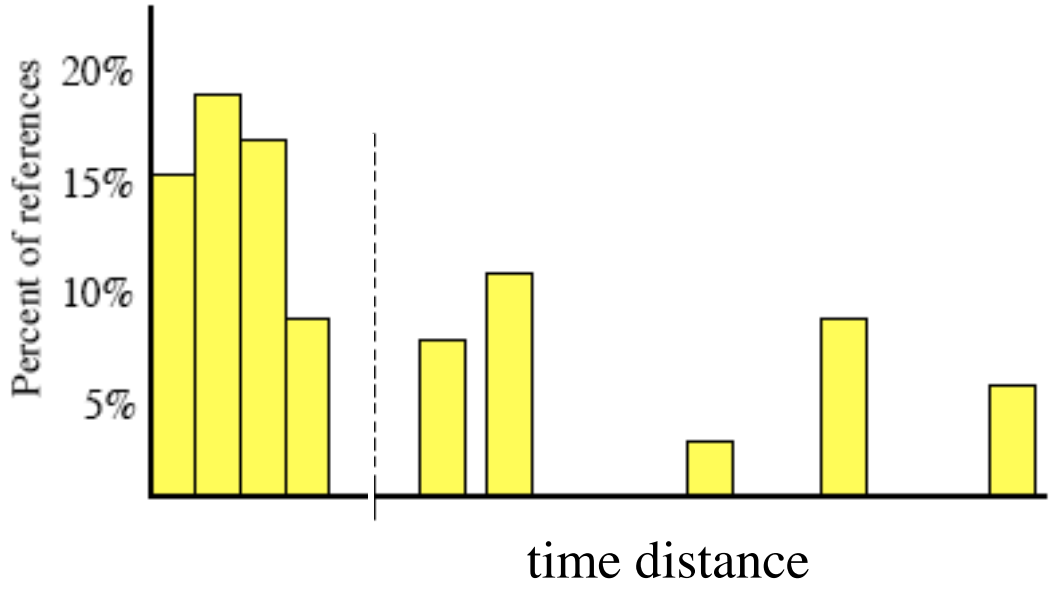
assuming no data sharing

(2) Handle effects of data sharing

Time Distance (TD)

- Def : the # of elements between reuses
  - E.g. b c a a c b td=4 (rd=2)
- TD Histogram (TDH)

Shows the probability for an access to have a certain TD.

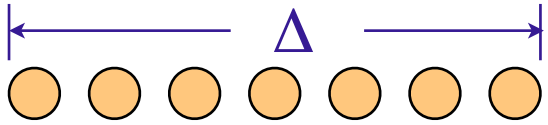


$$\Delta \xrightarrow{\text{TDH}} d$$

- $P_i(\Delta)$: Probability for an object O_i to be referenced in a Δ-long interval.

$$P_i(\Delta) = P_i(\Delta\text{-}1) + q_i(\Delta)$$
$$P_i(\Delta\text{-}1) = P_i(\Delta\text{-}2) + q_i(\Delta\text{-}1)$$
$$\vdots$$
$$P_i(1) = P_i(0) + q_i(1)$$

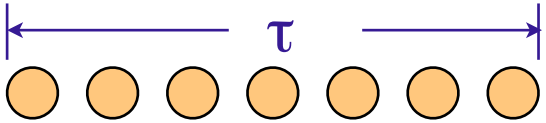
$$P_i(\Delta) = \sum_{\tau=1}^{\Delta} q_i(\tau)$$



$q_i(\Delta)$: O_i is accessed at time point Δ, but not at the Δ-1 points ahead.

$$\Delta \xrightarrow{\text{TDH}} d$$

- $q_i(\tau)$: O_i is accessed at time point τ, but not at the τ-1 points ahead. It is equivalent to



1) The object accessed at τ is O_i &

2) The time distance of that reference is greater than τ.

$$q_i(\tau) = \frac{n_i}{T} \sum_{\delta=\tau+1}^{T} H_i(\delta)$$

TDH

$$P_i(\Delta) = \frac{n_i}{T} \sum_{\tau=1}^{\Delta} \sum_{\delta=\tau+1}^{T} H_i(\delta)$$

$$\Delta \xrightarrow{\text{TDH}} \mathrm{d}$$

- P(k, Δ): prob. for a Δ-long interval to contain k distinct data.

$$P(k,\Delta) = \sum_{S:|S|=k;S\subseteq A} \left(\left(\prod_{i\in S} P_i(\Delta)\right) \left(\prod_{j\in A-S} (1-P_j(\Delta)) \right) \right)$$

- d: # of distinct data referenced in a Δ-long interval.

$$\mathrm{d} = \sum_{k=0}^{min(\Delta-1,N)} k \cdot P(k,\Delta)$$

See paper for details.

Handling Data Sharing

- Two effects from data sharing on CRD
- Example

a b X X b X c d X a

X: references by T_2

  - Scenario 1: Xs $\notin$ {a, b, c, d}.
    - a b p q b p c d q a CRD=3+2=5
  - Scenario 2: a $\in$ Xs.
    - a b p a b p c d q a break into 2 reuse intervals
  - Scenario 3: {b,c,d} $\cap$ Xs $\neq \phi$.
    - a b p c b p c d c a CRD=3+1=4

should not be counted.

Treating the Effects

See paper for details.

- Probability for a reuse interval to break

$$(|S|/N_1) * (n_2/N_2)$$

- Probability for |C|=c is

$$\frac{1}{\binom{N_1}{n_1} * \binom{N_2}{n_2}} \sum_{d=c}^{|S|} \binom{|S|}{d} \binom{N_1 - |S|}{n_1 - d} \binom{d}{c} \binom{N_2 - d}{n_2 - c}$$

S:	set of all shared data.
N1, N2:	data size of T1 and T2.
n1, n2:	# of distinct data accessed by T1 and T2 in an interval of length V.
C:	intersection of data sets referenced by T1 and T2 in the interval.

- Estimation accuracy of CRD histograms on synthetic traces

distr.	s=0		s=10%		s=20%		average
	n_1=200	n_1=200	n_1=200	n_1=200	n_1=200	n_1=200	
	n_2=100	n_2=200	n_2=100	n_2=200	n_2=100	n_2=200	
random	94.9	93.3	91.3	90.0	89.7	79.8	89.8
expon.	93.2	92.3	91.1	92.2	93.4	90.1	92.1
normal	95.9	94.6	94.4	80.8	93.4	91.6	91.8
random+ expon.	94.0	93.3	88.5	87.2	84.0	79.0	87.7
random+ normal	93.9	93.5	87.4	90.9	91.6	89.1	91.1
expon.+ normal	93.6	94.2	92.5	79.9	92.2	89.9	90.4
2random+ expon.+ normal	88.2	88.5	83.3	82.0	82.5	81.6	84.4
random+ 2expon.+ normal	89.0	84.8	70.1	72.8	85.3	83.5	80.9
random+ expon.+ 2normal	85.0	85.9	84.1	80.0	81.2	81.2	82.9
average	92.0	91.2	87.0	84.0	88.1	85.1	87.9

s: sharing ratio
n1, n2: data sizes

On Traces of Real Programs

- Using simulator to record traces.
  - SIMICS with GEMS
  - Simulate UltraSPARC with 1MB shared L2 cache.
- Three PARSEC programs
  - vips (image processing)
    - negligible shared data, 33,000 locks
    - accuracy 76%
  - swaptions (portfolio pricing)
    - 27% shared data, 23 locks
    - accuracy 74%
  - streamcluster (online clustering)
    - 3% shared data, 129,600 barriers
    - accuracy 72%

Related Work

- All-window profiling [Ding and Chilimbi]
- Predict cache misses of co-runs from circular stack distance histograms [Chandra et al., Chen & Aamodt]
- Statistical shared cache model [Berg et al.]

Conclusions

- Is reuse distance applicable for locality characterization on CMP?

  Difficult in general.

- What are the new challenges?

  Reliance on relative speeds; loss of hardware-indep; falling into a chicken-egg dilemma.

- Are these challenges addressable?

  Yes for a class of multithreading applications.
  A probabilistic model facilitates the derivation of CRD.

Thanks!

Questions?