UNIVERSITY of HOUSTON

Compile-Time Detection of False Sharing via Loop Cost Modeling

Munara Tolubaeva, Yonghong Yan and Barbara Chapman

High Performance Computing and Tools Group (HPCTools)

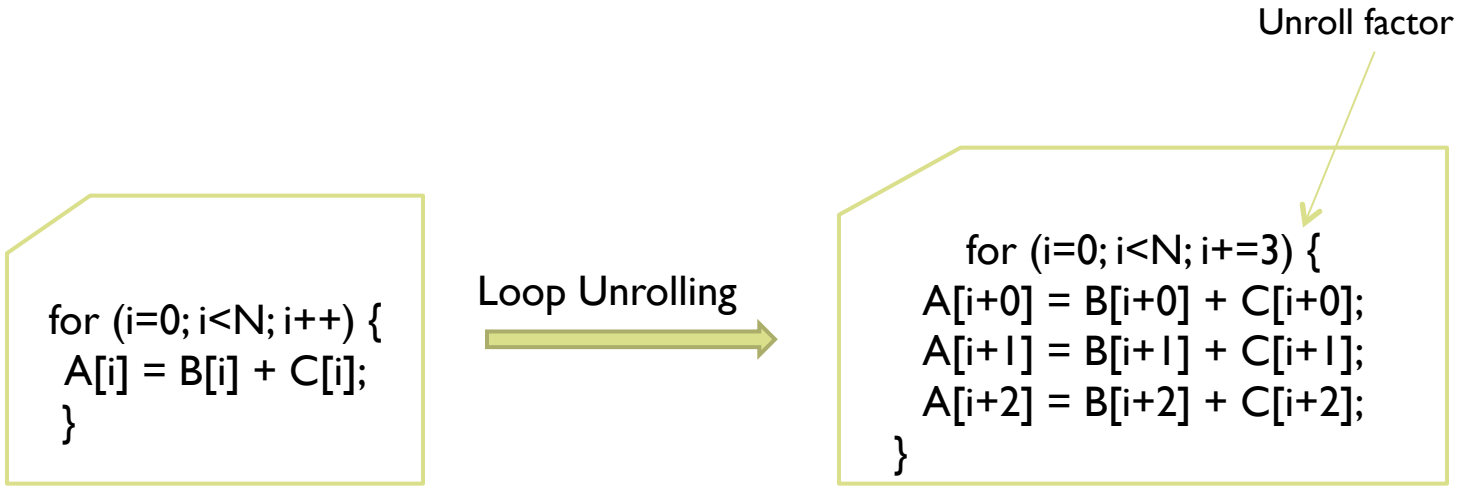
Computer Science Department
University of Houston

OUTLINE

- Introduction and Motivation
- Methodology
- Experiment
- Conclusion

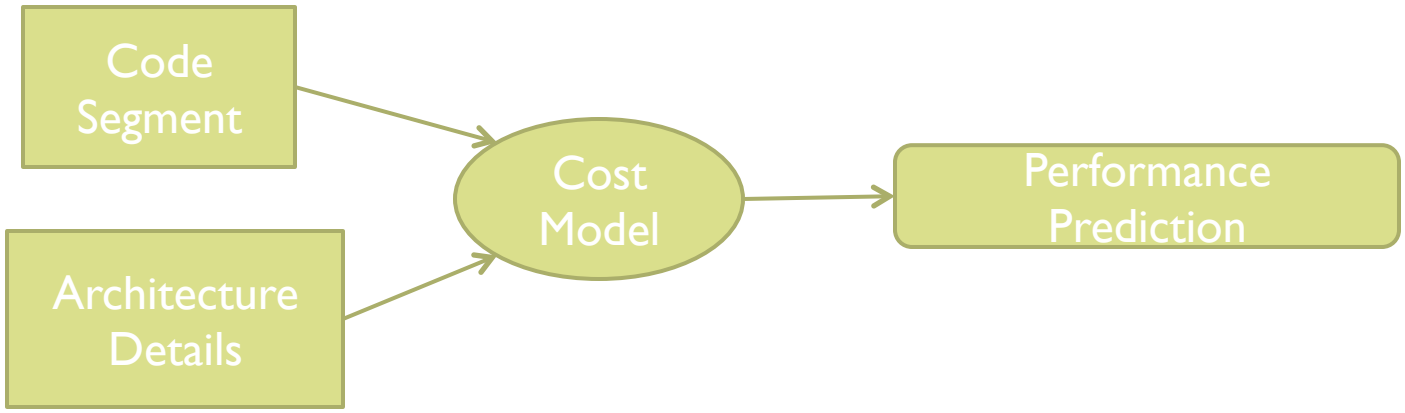
Introduction & Motivation

- Compiler Transformation Example



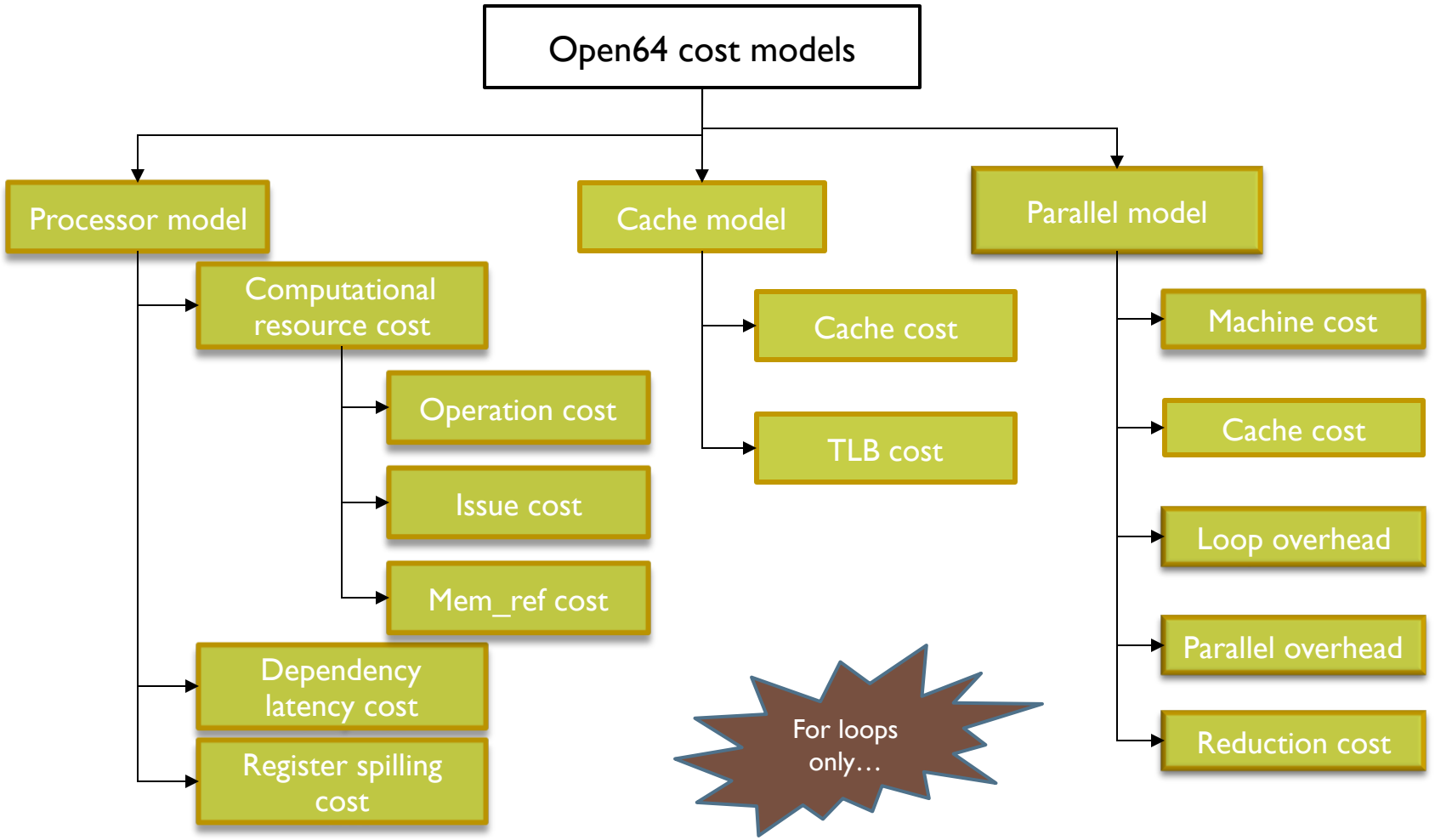
Introduction & Motivation

- Compiler Cost Model



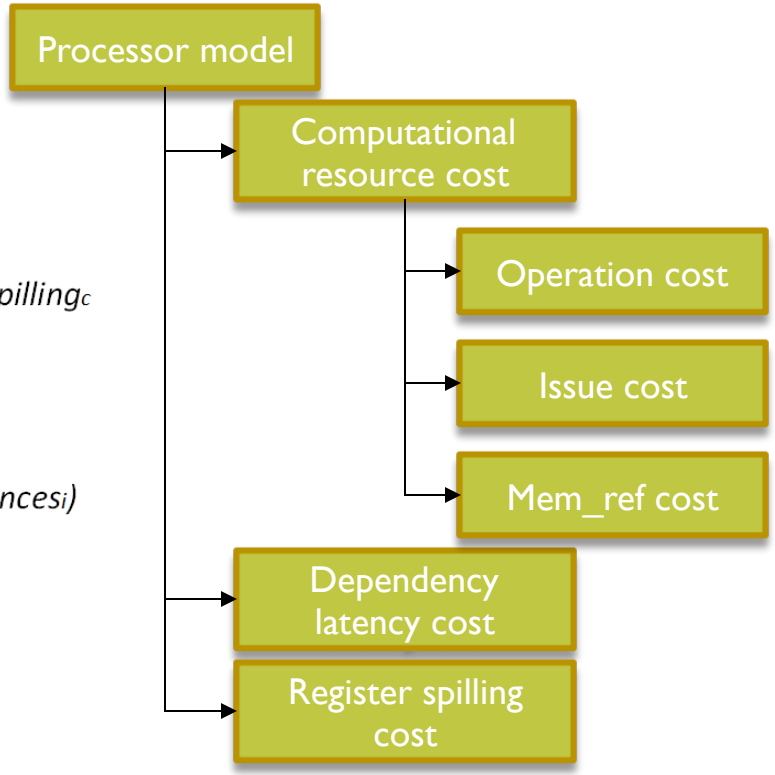
- Estimates the time needed to execute a specific section of code on a given system
- Considers performance impacting architectural features (Processor, Cache, Memory bandwidth, etc)
- Open64 cost models – the most sophisticated models among open source compilers

Introduction & Motivation



Introduction & Motivation

- Processor model - predicts the scheduling of instructions given the available amount of resources
- Guides loop unrolling
- Finds the optimal loop unrolling level and factor



$Machine_c\_per\_iter = Resource_c + Dependency\_latency_c + Register\_spilling_c$

$Resource_c = maximum(Op_c,\ MEM\_ref_c,\ Issue_c)$

$MEM\_ref_c = (Num\_fp\_refs + Num\_int\_refs)/Num\_mem\_units$

$Issue_c = Num\_inst/Issue\_rate$

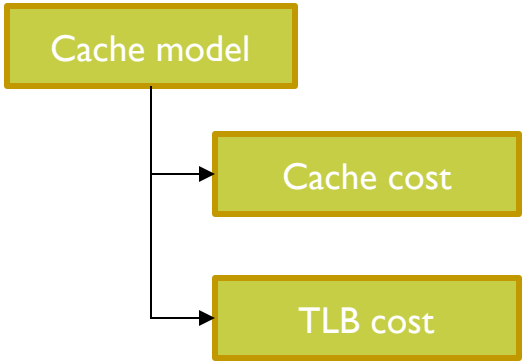
$Dependency\_latency_c = maximum(Sum\_of\_latencies_i/Sum\_of\_distances_i)$

$Register\_spilling_c = (Reg\_used - Target\_regs) * [Num\_reg\_refs/(Scalar\_regs + Array\_regs)]$

$Reg\_used = Base\_regs + Scalar\_regs + Array\_regs$

Introduction & Motivation

- Cache model - predicts the number of cache misses and estimates additional cycles needed to execute an iteration of an inner loop
- Guides loop tiling
- Finds the optimal loop tiling size



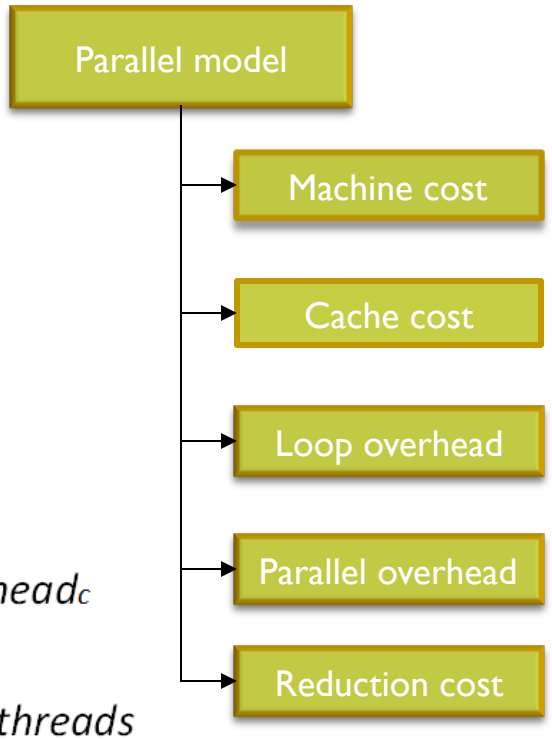
$$TLB\_miss = Num\_array\_ref - TLB\_entries,\ if(Num\_array\_ref - TLB\_entries > 0)$$

$$TLB_C = TLB\_miss\_penalty * TLB\_miss$$

$$Cache_C = \sum_{i}^{Levels} (Clean\_footprint_i * Clean\_penalty_i + Dirty\_footprint_i * Dirty\_penalty_i)$$

Introduction & Motivation

- Parallel model – decides loop level that is the best candidate for parallelization
- Evaluates the cost involved in parallelizing the loop
- Used in auto-parallelization phase



$Total_c = Machine_c + TLB_c + Cache_c + Loop\_overhead_c + Parallel\_overhead_c$

$Machine_c = Machinec\_per\_iter * Num\_loop\_iter/Num\_threads$

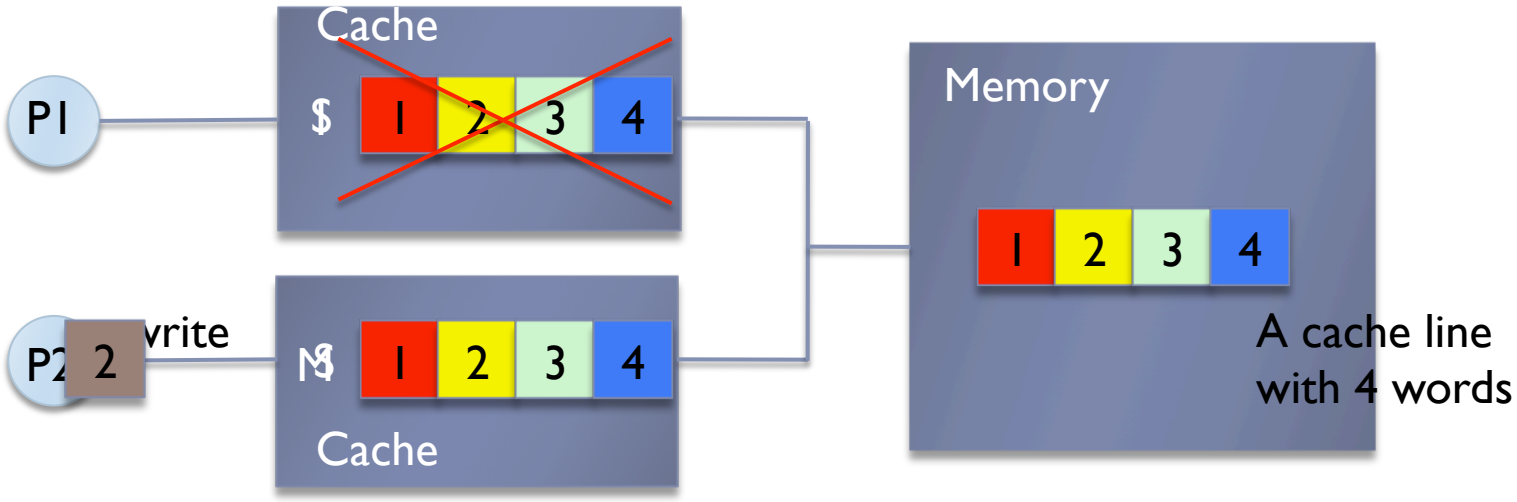
$Loop\_overhead_c = Loop\_overhead\_per\_iter_i * Num\_loop\_iter/Num\_threads$

$Parallel\_overhead_c = Parallel\_startup_c + Parallel\_const\_factor_c * Num\_threads$

Introduction & Motivation

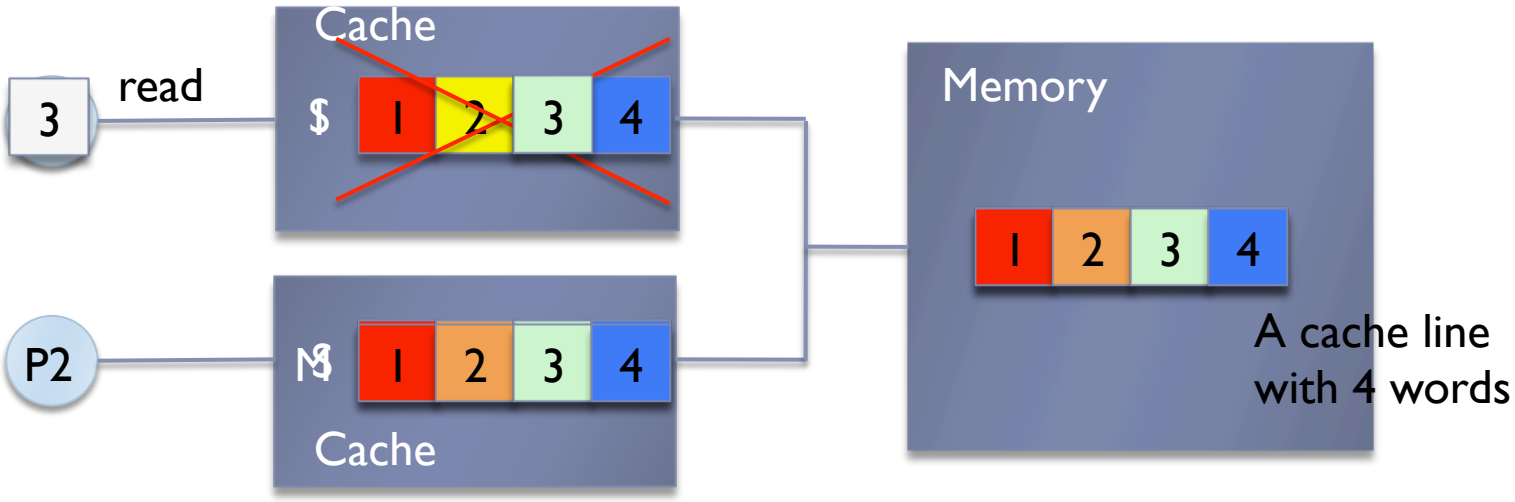
- State – of – Art
  - Optimize single CPU performance, not considering shared resource contention
  - Limited use of models for compiler optimizations and transformations
  - All false sharing detection techniques implemented at runtime

False Sharing



- Processors maintain data consistency via cache coherency
  - Data sharing is at cache line granularity
- A store to a single data invalidates the whole copy of a cache line
- Successive read suffers a cache miss
  - Reload entire cache line

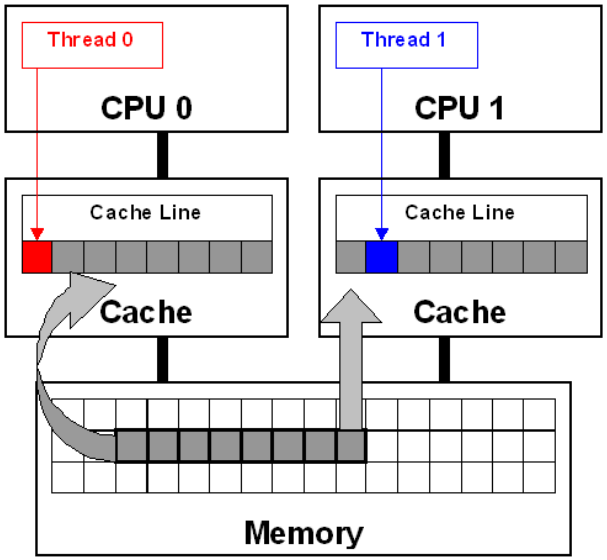
False Sharing



- Processors maintain data consistency via cache coherency
  - Data sharing is at cache line granularity
- A store to a single data invalidates the whole copy of a cache line
- Successive read suffers a cache miss
  - Reload entire cache line

Effects of False Sharing

False sharing is a performance degrading data access pattern that can arise in systems with distributed, coherent caches.



Code Version	Execution Time (sec)			
	Sequential	2 threads	4 threads	8 threads
Unoptimized	0.503	4.563	3.961	4.432
Optimized	0.503	0.263	0.137	0.078

False Sharing: Monitoring Results

- Cache line invalidation measurements

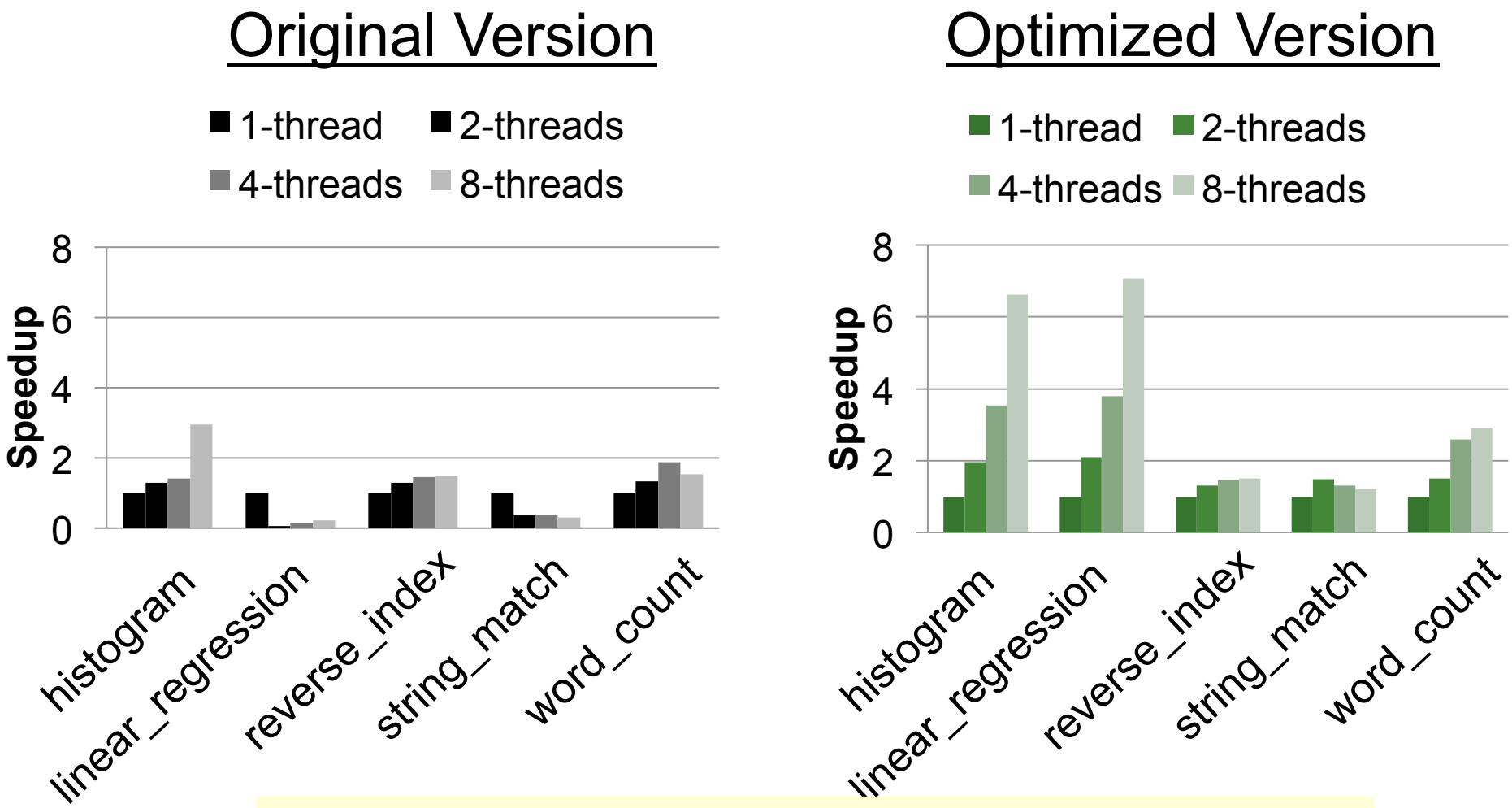
Program name	1-thread	2-threads	4-threads	8-threads
histogram	13	**7,820,000**	**16,532,800**	**5,959,190**
kmeans	383	28,590	47,541	54,345
linear_regression	9	**417,225,000**	**254,442,000**	**154,970,000**
matrix_multiply	31,139	31,152	84,227	101,094
pca	44,517	46,757	80,373	122,288
reverse_index	4,284	89,466	217,884	**590,013**
string_match	82	**82,503,000**	**73,178,800**	**221,882,000**
word_count	4,877	**6,531,793**	**18,071,086**	**68,801,742**

False Sharing: Data Analysis Results

- Determining the variables that cause misses

Program Name	Global/static data	Dynamic data
histogram	-	main_221
linear_regression	-	main_155
reverse_index	use_len	main_519
string_match	key2_final	string_match_map_266
word_count	length, use_len, words	-

Runtime Handling of False Sharing



B. Wicaksono, M. Tolubaeva and B. Chapman. "Detecting false sharing in OpenMP applications using the DARWIN framework", LCPC 2011

Related Work

- False Sharing Detection Methods
  - Cache simulation and Memory tracing (Gunther and Weidendorfer WBIA'09, Marathe and Muller TPDS'07, Martonosi et al Sigmetrics'92)
  - Hardware Performance Counters (Marathe et al. Tech. rep.'06, Wicaksono et al. LCPC'11)
  - Memory Protection (Tongping and Berger OOPSLA'11)
  - Memory Shadowing (Zhao et al. VEE'11)
- False Sharing Elimination Methods
  - Tune scheduling parameters (chunk size, chunk stride) (Chow and Sarkar ICPP'97)
  - Compiler transformations (array padding, memory alignment) (Jeremiassen and Eggers PPoPP'94)
- All FS detection methods are applied at runtime, incur overhead

False Sharing Cost Model

- False Sharing (FS) Modeling
  - Estimates the performance impact of FS on OpenMP parallel loops at compile – time.
- Features:
  - Ability to output the total number of FS cases that will occur during execution of the parallel loop.
  - Ability to analyze the performance impact of FS on a parallel loop as a percentage of execution time.
  - Introduces a linear regression model to reduce the modeling time by approximation without impacting its accuracy.

Methodology

- False Sharing Model needs:
  - # of threads executing the loop
  - Loop boundaries
  - Step sizes
  - Index variables
  - Chunk size (if specified for OpenMP loop)

Methodology

False Sharing Modeling

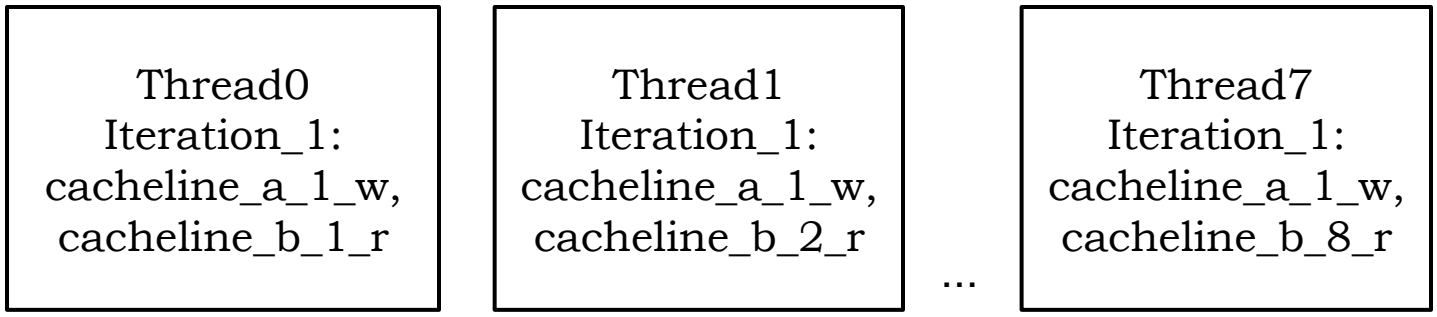
- Technique is comprised of 4 steps
  - Obtain array references made in the innermost loop of a loop nest
  - Generate a cache line ownership list for each thread
  - Apply a stack distance analysis to cache state of each thread
  - Detect false sharing

Methodology – Step 1

- Obtain array references made in the innermost loop of a loop nest
  - Array base name
  - Array indices
  - Memory offsets for arrays with structured data types

Methodology – Step 2

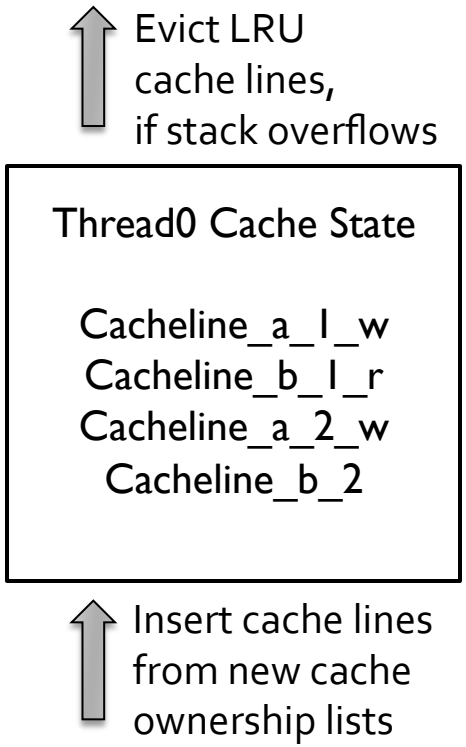
- Generate a cache line ownership list



- Assumption: all array variables are cache aligned

Methodology – Step 3

- Apply a stack distance analysis
- Simulate fully associative cache
  - impossible to know corresponding cache line in a set at compile time
  - modeling the fully associative cache is mostly valid especially for caches with high level of associativity[1]



1. A. Sandberg, D. Eklov, and E. Hagersten. Reducing cache pollution through detection and elimination of non-temporal memory accesses. SC 2010.

Methodology – Step 4

- Detect False Sharing
  - Perform 1 to All comparison

$$\varphi(cs_k, cl_i) = \begin{cases} 1, & if\ (cl_i \in cs_k\ \ and\ \ cs_k^{cl_i} = W) \\ 0, & \text{otherwise} \end{cases}$$

    - do other cache states contain my cache line?

- Perform comparison for each thread's new cache line ownership list at each iteration until all iterations in one chunk are evaluated

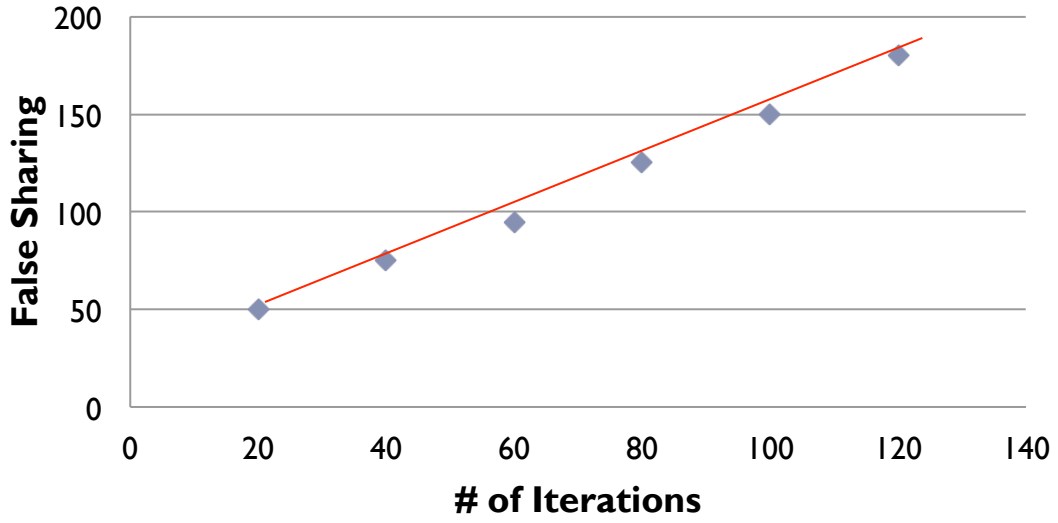
$$false\_sharing_{iter} = \sum_{j=0}^{k-1} \sum_{i=0}^{n} \varphi(cs_j, cl_i) \times mask(cs_j, cl_i)$$

$$mask(cs_j, cl_i) = \begin{cases} 0, & if\ (\mathrm{cl}_i \in \mathrm{CLOL}_j^{cs_j}) \\ 1, & \text{otherwise} \end{cases}$$

- Perform steps 2-4 until all iterations are finished

Methodology - Prediction with linear regression

- Full model is expensive when # iterations becomes large
- Prediction with Linear Regression
  - Predict the total false sharing cases by evaluating much lower number of iterations in much less time

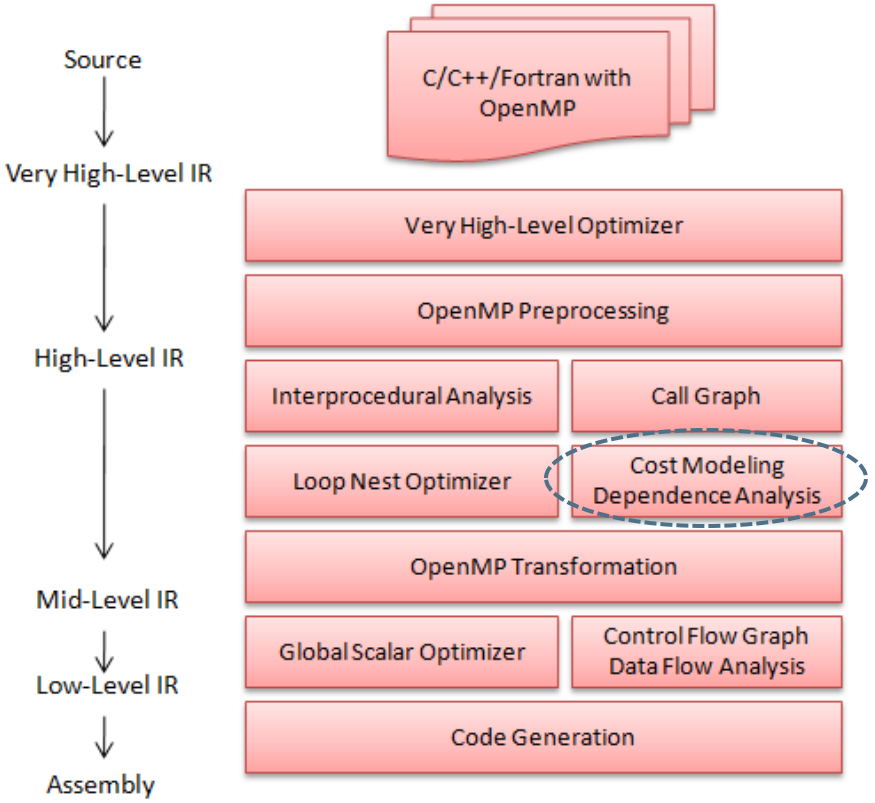


Methodology - Prediction with linear regression

- First *n* iterations: $\vec{x} = \{x_1, .. x_n\}$ where *n*<<*N*
- False sharing cases in *n*: $\vec{y} = \{y_1, .. y_n\}$
- Prediction can be modeled as $\vec{y} = a\vec{x} + b$
- *Least Square Solution:* $f(a,b) = \left\| a\vec{x} + b - \vec{y} \right\|_2$
  - We want $a, b = \arg\min_{a,b} f(a,b) = (a\vec{x} + b - \vec{y})^T (a\vec{x} + b - \vec{y})$
- Differentiate the function: $\partial f / \partial a = 0, \ \partial f / \partial b = 0$
- Then, $b = \sum_{i=0}^{n-1} y_i - \frac{a}{n} \sum_{i=0}^{n-1} x_i \qquad a = \sum_{i=0}^{n-1} x_i y_i \Big/ \sum_{i=0}^{n-1} (x_i)^2$
- Total number of iterations: $x_{\max}$
- Predict $y_{\max}$ as: $y_{\max} = a x_{\max} + b$

Implementation and Experiments

- Implemented in OpenUH compiler
- Separate IR pass to collect memory load/store details, loop details
- No modification to compiler's IR
- Can be implemented in similar approach in other compilers



Evaluation

- Memory access dominates the total execution
- Accuracy evaluation of full false sharing model
  - Compare the percentages of measured and modeled FS overhead costs on the total loop execution time.

$$\frac{T_{fs\_measured} - T_{nfs\_measured}}{T_{fs\_measured}} \approx \frac{T_{fs\_modeled} - T_{nfs\_modeled}}{T^{*}_{fs\_modeled}} \approx \frac{N_{fs\_modeled} - N_{nfs\_modeled}}{N^{*}_{fs\_modeled}}$$

- Efficiency of false sharing prediction with linear regression
  - Compare the number of false sharing cases estimated by full false sharing model and the prediction with linear regression model

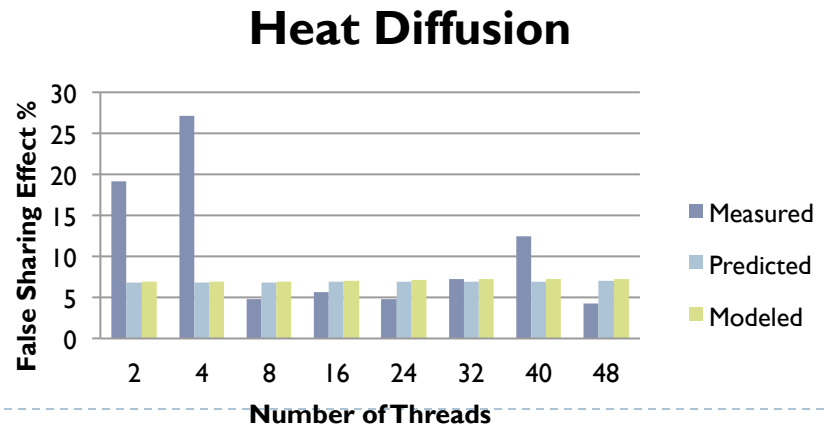
Experiments

- Architecture:
  - Four 2.2. GHz 12-core processors (48 cores in total).
  - Dedicated L1 and L2 caches, 64Kb and 512KB per core.
  - L3 cache of 10240KB shared among 12 cores.
  - Cache line size for all caches, 64 bytes.

Experimental Results – Heat Diffusion

# of threads	Measured Time with chunk size=1 FS case (sec)	Measured Time with chunk size=64 non-FS case (sec)	Measured FS effect on execution time (%)	Modeled FS cases effect (%)
2	0.3593	0.2901	19.2%	6.9%
4	0.2263	0.1646	27.2%	6.9%
8	0.1639	0.156	4.8%	6.9%
16	0.6586	0.6205	5.7%	7.0%
24	1.0049	0.9564	4.8%	7.1%
32	1.4671	1.3608	7.2%	7.2%
40	1.8455	1.6130	12.5%	7.2%
48	2.247	2.1501	4.3%	7.2%

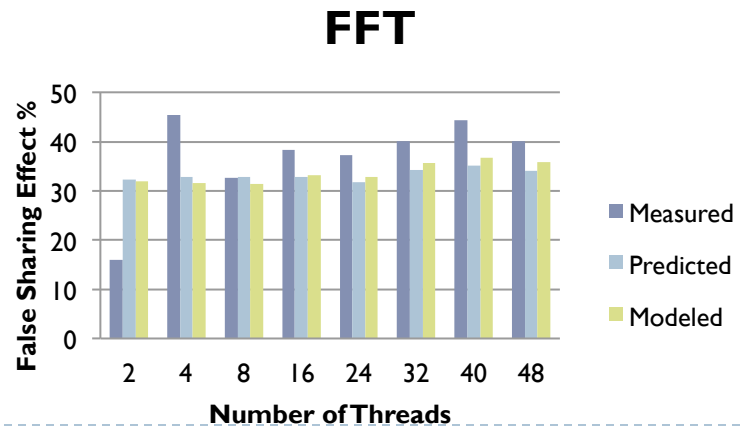
# of threads	Pred. # of FS cases chunk size=1 (chunk run=20)	Pred. # of FS cases chunk size=64 (chunk run=20)	Pred. FS cases effect	Modeled # of FS cases chunk size=1	Modeled # of FS cases chunk size=64	Modeled FS cases effect
2	91,991K	1,595K	6.8%	94,421K	2,107K	6.9%
4	92,979K	1,625K	6.8%	94,446K	2,145K	6.9%
8	93,496K	1,702K	6.8%	94,458K	2,070K	6.9%
16	93,990K	1,724K	6.9%	96,043K	1,888K	7.0%
24	94,155K	1,609K	6.9%	96,938K	1,699K	7.1%
32	93,986K	1,456K	6.9%	97,159K	1,509K	7.2%
40	94,286K	1,826K	6.9%	97,730K	1,889K	7.2%
48	94,319K	1,107K	7.0%	97,935K	1,126K	7.2%



Experimental Results - FFT

# of threads	Measured Time with chunk size=1 FS case (sec)	Measured Time with chunk size=16 non-FS case (sec)	Measured FS effect on execution time (%)	Modeled FS cases effect (%)
2	2.0978	1.7624	15.9%	32.0%
4	1.762	0.9618	45.4%	31.6%
8	0.8976	0.6033	32.7%	31.5%
16	0.599	0.3688	38.4%	33.2%
24	0.5041	0.3163	37.2%	32.8%
32	0.4727	0.2827	40.1%	35.6%
40	0.4792	0.2669	44.3%	36.7%
48	0.4664	0.279	40.1%	35.8%

# of threads	Pred. # of FS cases chunk size=1 (chunk run=50)	Pred. # of FS cases chunk size=16 (chunk run=50)	Pred. FS cases effect	Modeled # of FS cases chunk size=1	Modeled # of FS cases chunk size=16	Modeled FS cases effect
2	52,233K	26,468K	32.4%	53,058K	27,358K	32.0%
4	52,697K	26,491K	32.8%	53,088K	27,702K	31.6%
8	52,928K	26,612K	32.8%	53,311K	27,882K	31.5%
16	52,936K	26,526K	32.9%	54,411K	27,257K	33.2%
24	52,967K	27,475K	31.8%	54,956K	28,003K	32.8%
32	52,983K	25,523K	34.2%	55,245K	25,865K	35.6%
40	53,077K	24,895K	35.1%	55,510K	25,154K	36.7%
48	52,998K	25,649K	34.1%	55,542K	25,878K	35.8%



Summary

- False Sharing Cost Model is useful:
  - Assist compiler in optimizing code in high-level loop transformation, low-level instruction scheduling and code generation.
  - Guide traditional loop transformations.
  - Assist in generating efficient code.

Thank You