Reducing False Sharing on Shared Memory Multiprocessors through Compile Time Data Transformations

Tor E. Jeremiassen, Susan J. Eggers
Presented by Tony Bai, Brandon Kayes, Daniel Hoekwater, Thomas Smith

What is false sharing?

- Multiple CPUs, each with their own cache
- Different CPUs using different pieces of data should be able to run completely in parallel
- **False sharing:** The different pieces of data end up on the same cache line, so the CPUs must coordinate whenever either piece of data is changed
    - A CPU that writes to a falsely shared cache line must invalidate all other CPUs' caches
    - Other CPUs will incur cache misses when they try to read the falsely shared cache line

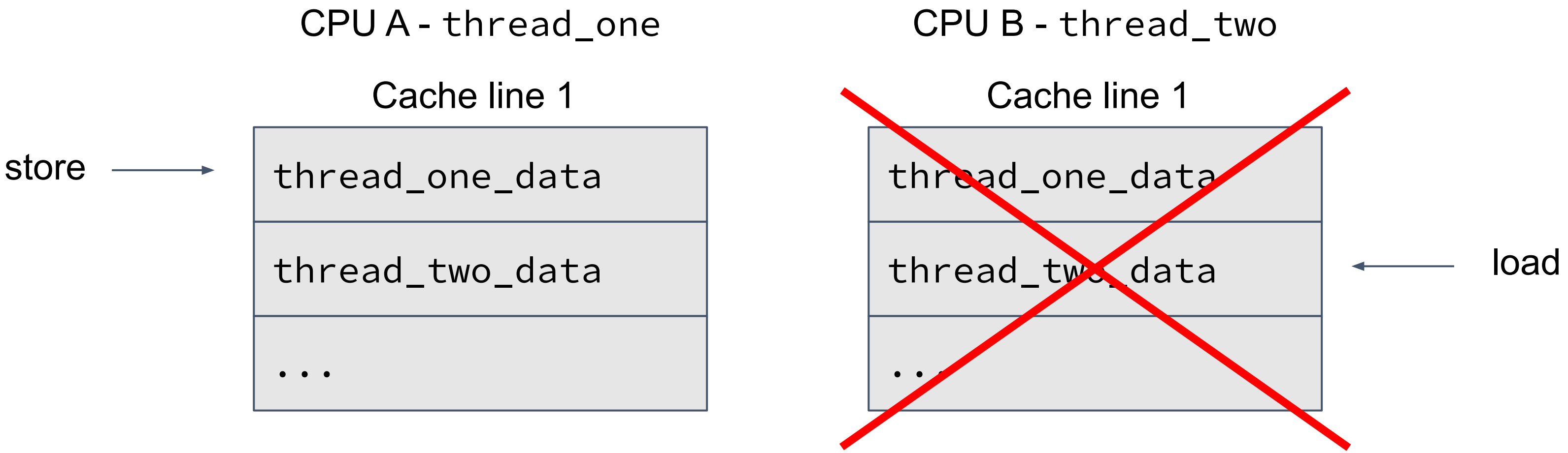
What is false sharing?

```
struct {
    int thread_one_data;
    int thread_two_data;
} shared;

void thread_one() {
    for (int i = 0; i < 1000; ++i) {
        ++shared.thread_one_data;
    }
}

void thread_two() {
    for (int i = 0; i < 1000; ++i) {
        printf("%d\n", shared.thread_two_data);
    }
}
```

What is false sharing?



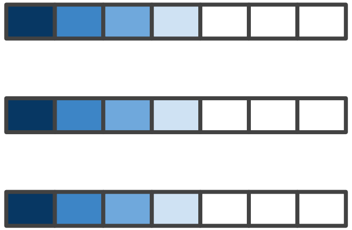
Group and Transpose

- Physically group per-process data together by changing the layout of data structures in memory
- If each processor's data is less than the cache block size, it may be padded
- May also improve spatial locality

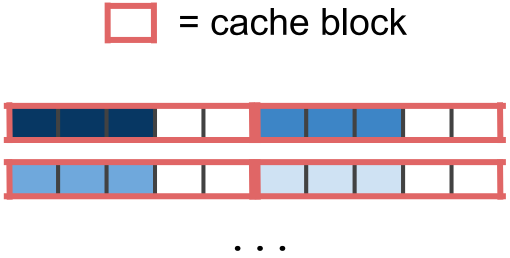
```
typeA Vect1[N]

typeB Vect2[N]

typeC Vect3[N]
```

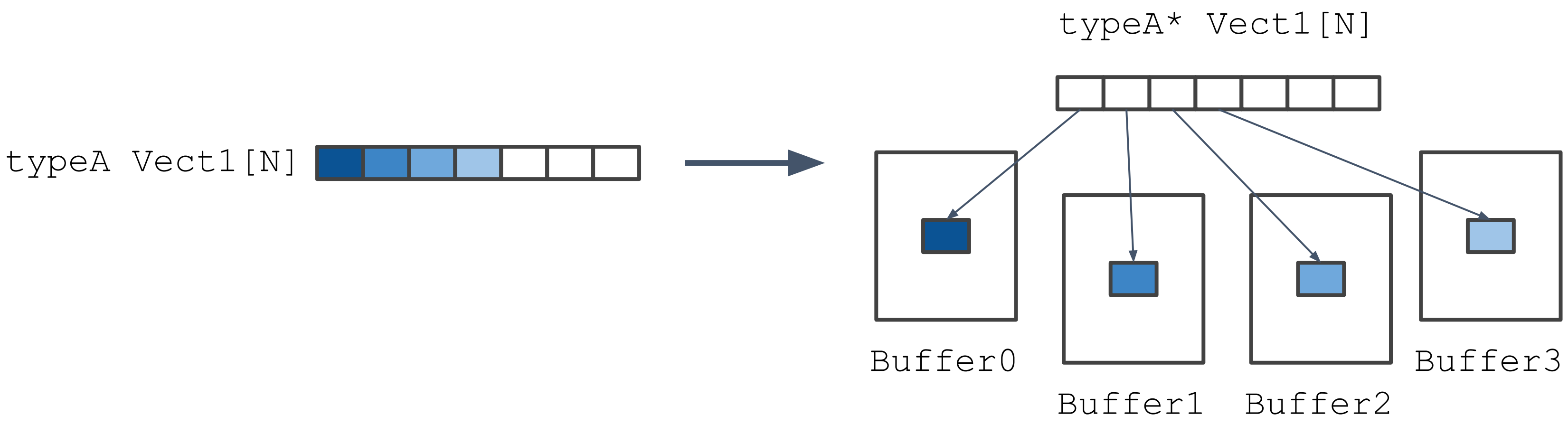


```
struct {
    typeA Vect1;
    typeB Vect2;
    typeC Vect3;
    gt_pad1 Padding;
} GTVect[N];
```



Indirection

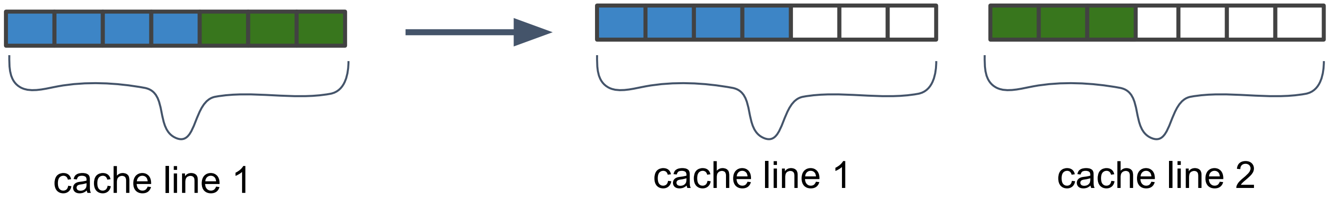
- When it's not physically possible to change the data layout, we can use indirection
  - Pointers are read-only
- Run time overhead
  - additional space for the pointers
  - additional memory access for each reference to the data



Pad and Align

- Pads and aligns scalars, array elements, and locks on cache block boundaries
- Increases data set size, so may increase conflict and capacity misses and reduce spatial locality
    - Therefore only pad data structures that lack processor locality, i.e., where the possible loss of spatial locality is insignificant relative to the savings in false sharing.

```
typeA Vect1[7]
```




```
struct {
   typeA Vect1[4];
   typeA padding[3];
   typeA Vect2[3];
} PadVect;
```

Detecting and Applying Transformations

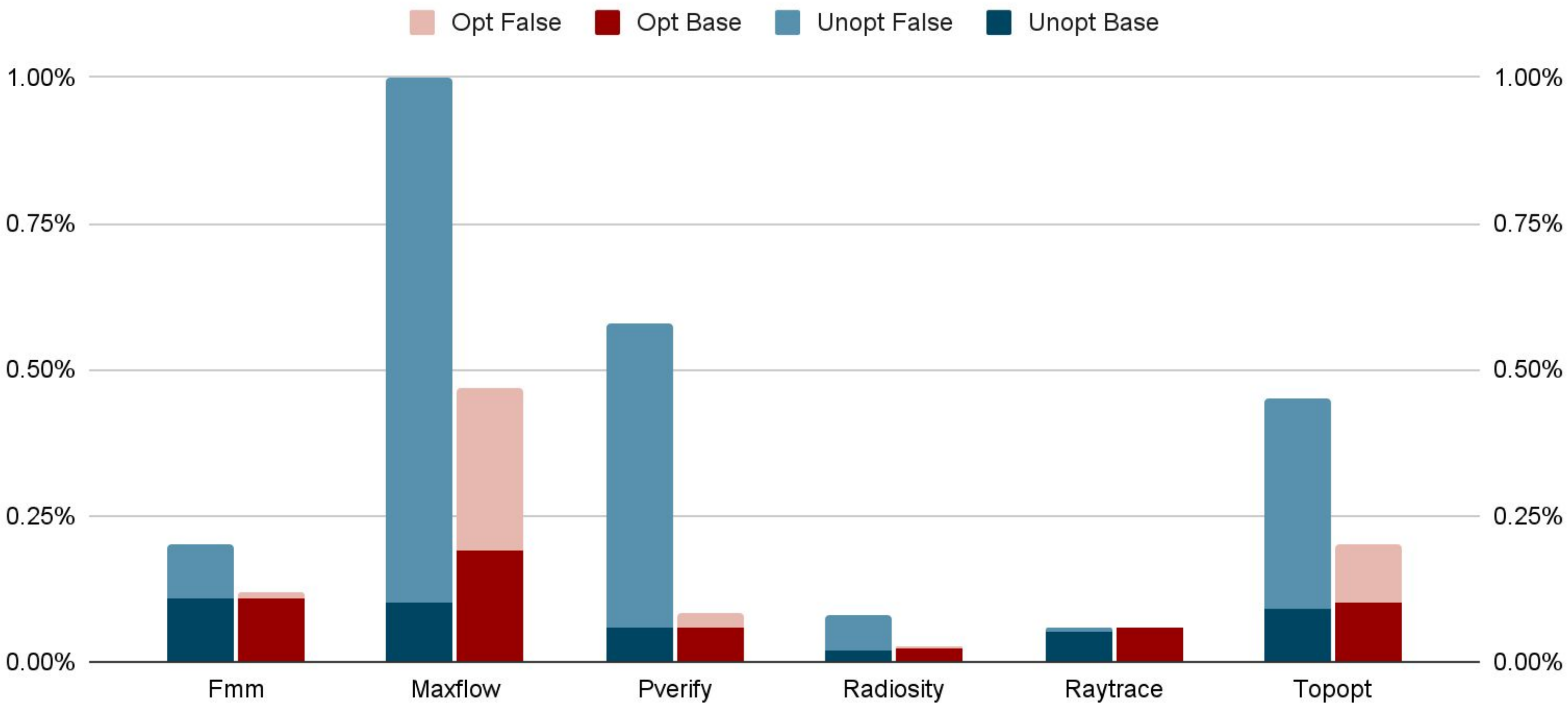
- Compile-time analysis is used to pinpoint data structures that are susceptible to false sharing. Approximate memory access pattern is computed.
    - **Stage 1**: Inter-procedural analysis of the control flow. CFG nodes annotated accordingly.
    - **Stage 2**: Non-concurrency analysis using barrier synchronization points to determine which portions of a program can execute in parallel and which cannot.
    - **Stage 3**: Summary side-effect analysis and static profiling on a per-process basis (based on the control flow determined in Stage 1) for each phase (determined in Stage 2).
- After static analysis, **heuristics** were used to determine where to mitigate false sharing
    - Compare results of the **per-process side-effect analysis** to **profiling information from simulations** that showed the number of false sharing misses per data structure.
    - Transformations **applied when reduction in false sharing exceeds any performance loss from reduced spatial locality**

Results

- Evaluated performance for multiple different programs
    - Maxflow - maximum flow in a directed graph
    - Pverify - logical verification
    - Topopt - topological optimization
    - Fmm - fast multipole method
    - Radiosity - equilibrium distribution of light
    - Raytrace - rendering of 3-dimensional scene

    - More: LocusRoute, Mp3d, Pthor, Water

Results - Comparison



Results - Performance Breakdown

Program	Total Reduction in False Sharing	Fraction of Reduction by Transformation			
		Group & Transpose	Indirection	Pad & Align	Locks
Maxflow	56.5%			**49.2%**	7.3%
Pverify	91.2%	6.4%	**81.6%**		3.1%
Topopt	79.9%	**61.3%**	18.6%		
Fmm	90.8%	**84.8%**			6.0%
Radiosity	93.5%	**85.6%**		1.0%	6.8%
Raytrace	78.3%	**70.4%**		3.3%	4.6%

Paper Pros

- Novel ideas to reduce false-sharing
- Static Analysis
- Techniques help with both performance and scalability!

Paper Cons

- Reordering members of struct may be disallowed
- Can lead to extra memory usage and less locality
- Indirection creates extra overhead
- Static analysis
  - Heuristics are unclear

Conclusion

- False sharing is a silent killer of performance in concurrent programs
- By using static analysis, Jeremiassen and Eggers identify heuristics to detect where false sharing occurs and determine how best to fix it
- If locality is not hurt too much, the Group & Transpose, Indirection, and Pad & Align transformations can be applied to reduce false sharing
- Performance is demonstrably improved on a variety of benchmarks