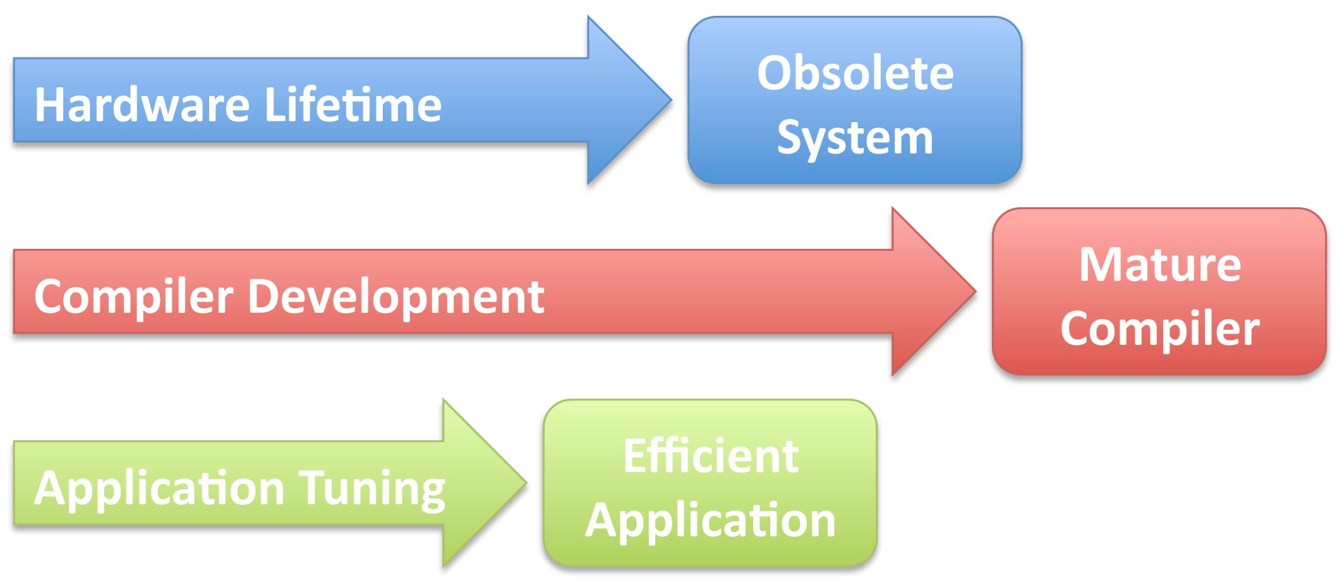
Detecting Instruction Cache for Platform Aware Compilation

PACE platform-aware compilation environment

Keith D. Cooper
Timothy J. Harvey
Jeffrey A. Sandoval

Problem

Compiler development is expensive and time-consuming. Rapid development cycles for modern computing systems can render a system obsolete before a mature compiler is even available.



Solution

Build a compiler that can automatically adapt to future systems.

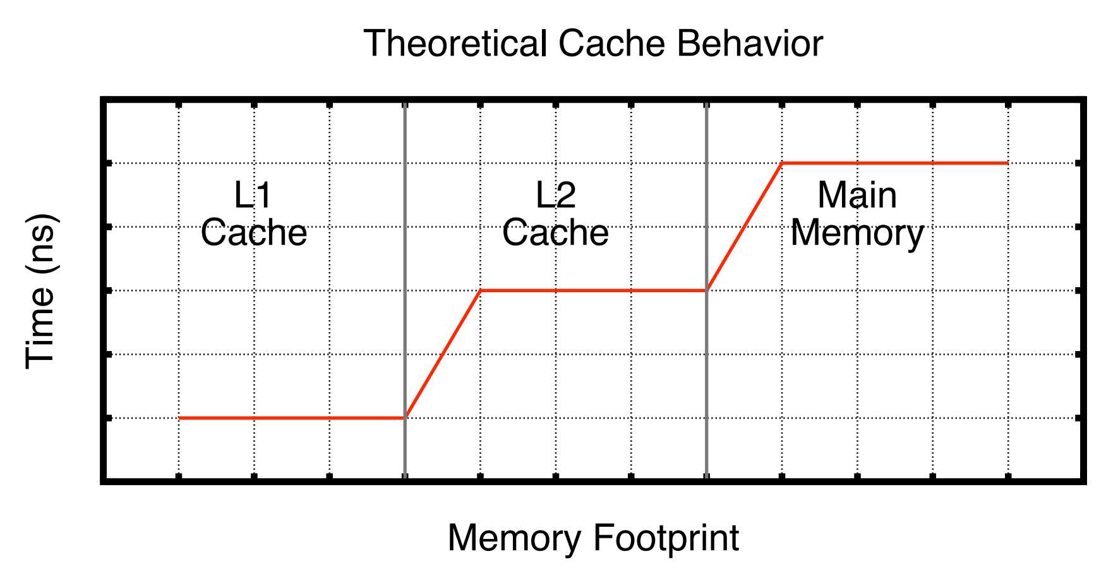
- Requires automatic characterization of performance-related system properties
  - Hardware: memory hierarchy, available registers, instruction latency, instruction level parallelism,etc.
  - Software: compiler strengths/ weaknesses, OS bottlenecks, etc.
- *This poster focuses on measuring a processor's instruction cache.*

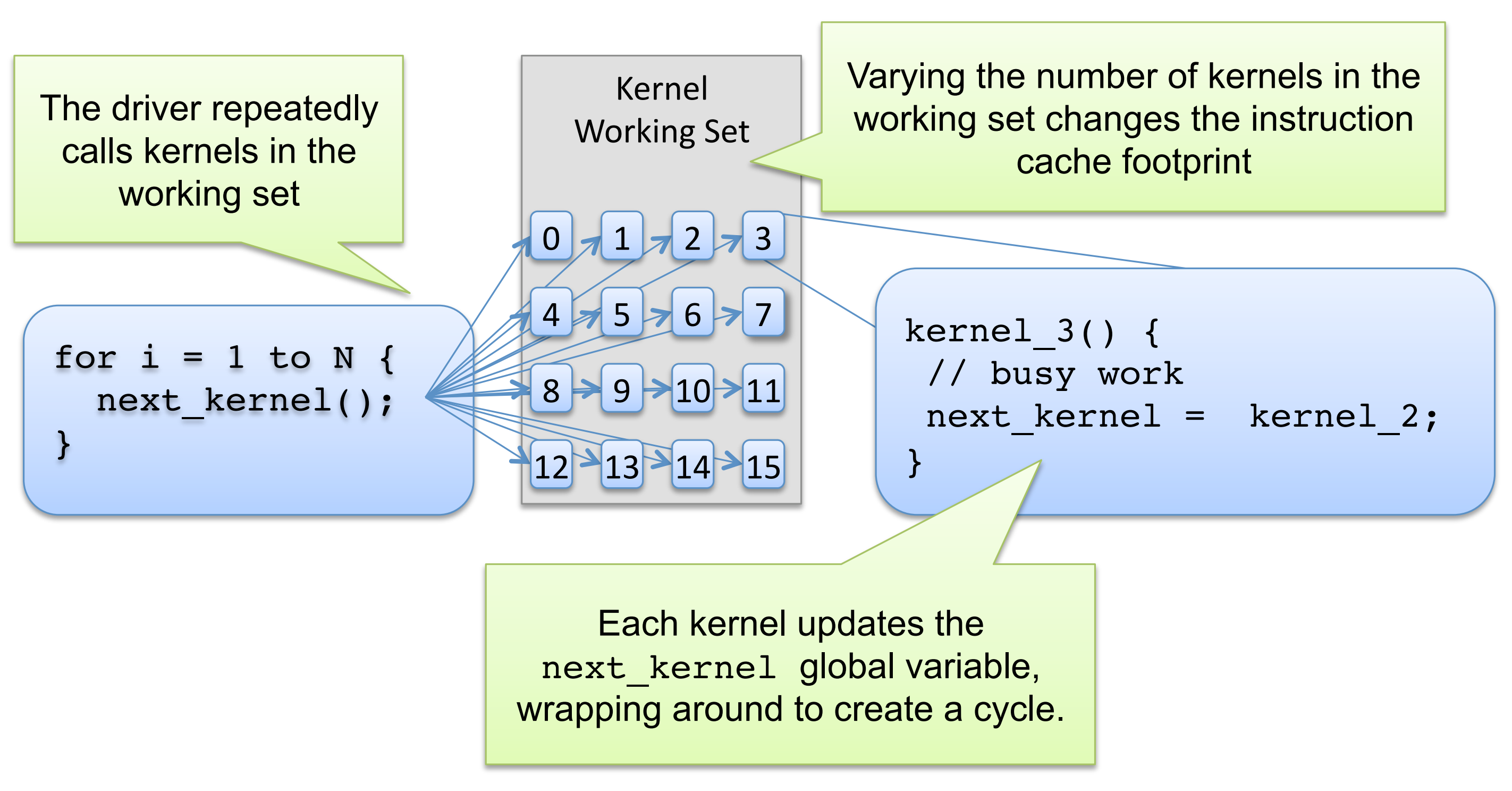
Why Instruction Cache?

- **Instruction cache optimizations**: we can limit code growth from loop unrolling and function inlining
- **Data cache optimizations**: it's helpful to know which levels of cache are unified (i.e., data and instructions)
- **Exploiting under-utilization**: under-utilized instruction cache can be seen as an opportunity for additional code cloning and specialization

Detecting Instruction Cache

Holding the dynamic instruction count constant while varying the static instruction count will reveal the instruction cache.

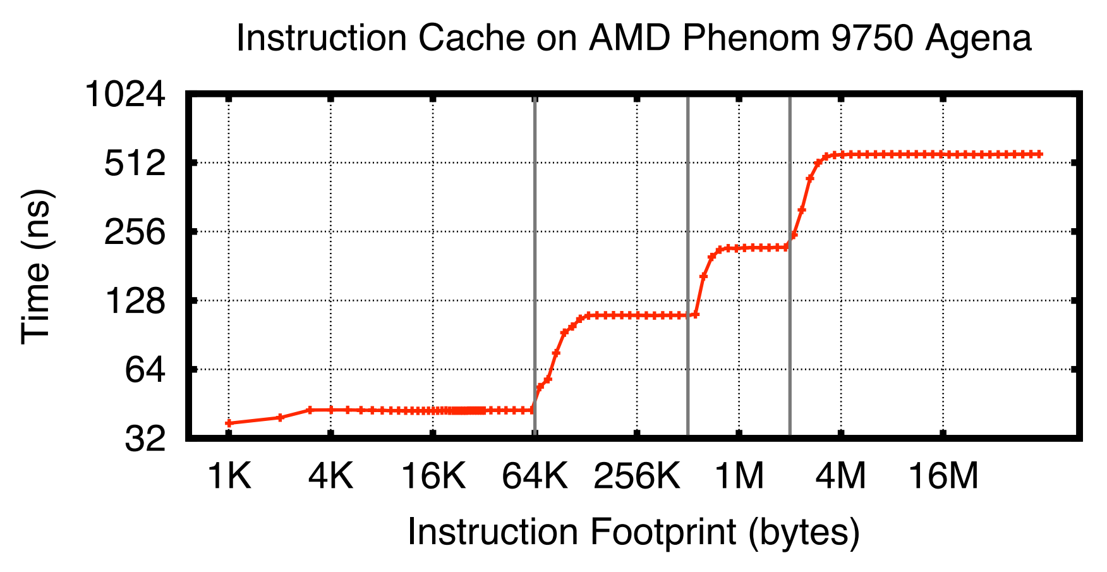


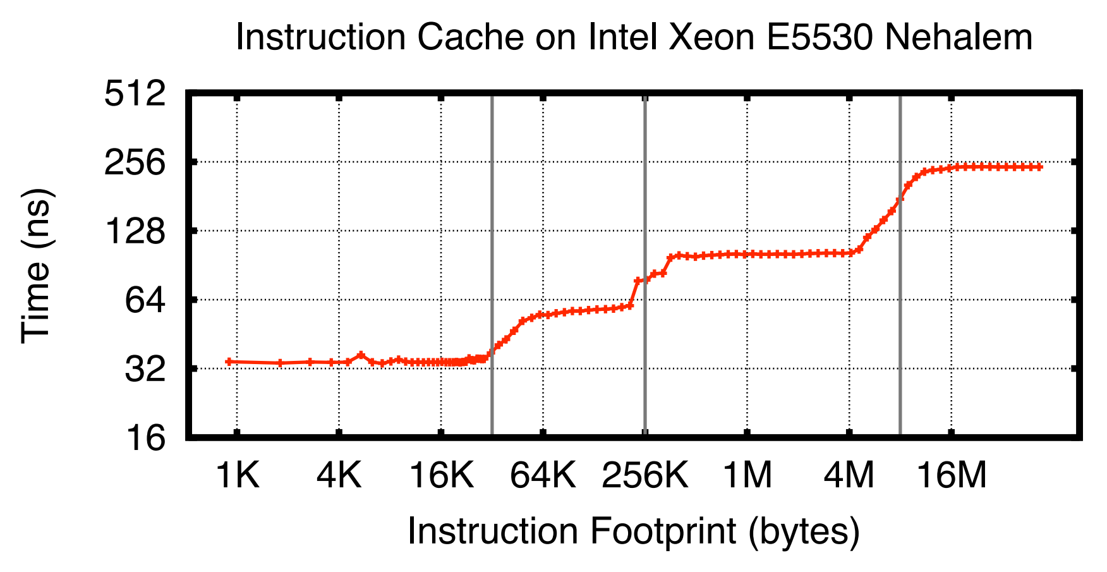

The driver repeatedly calls kernels in the working set
Kernel Working Set
Varying the number of kernels in the working set changes the instruction cache footprint
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```
for i = 1 to N {
   next_kernel();
}
```

```
kernel_3() {
  // busy work
  next_kernel =  kernel_2;
}
```

Each kernel updates the `next_kernel` global variable, wrapping around to create a cycle.



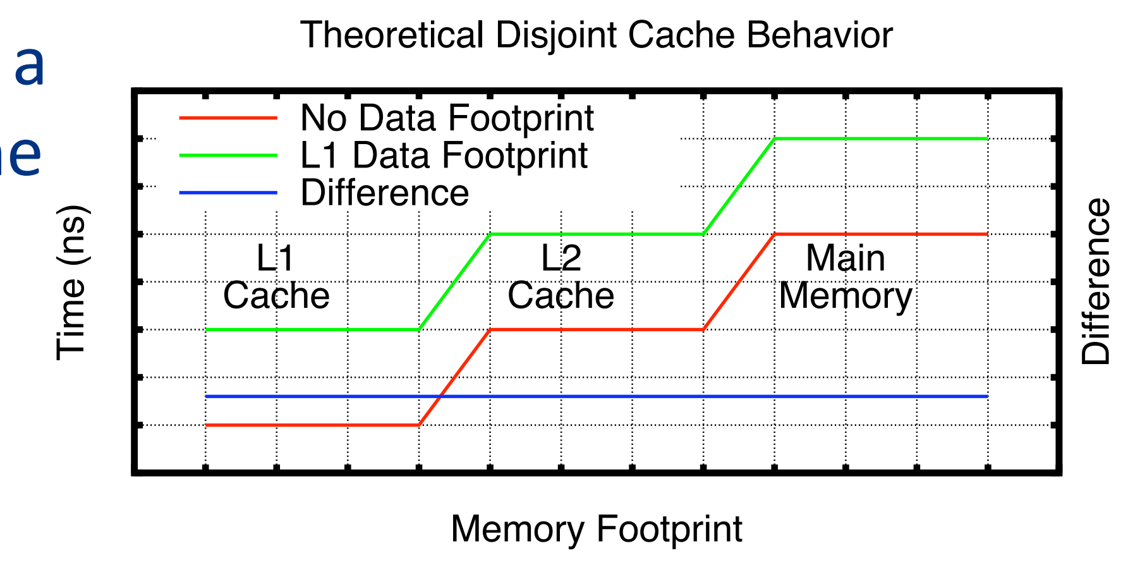


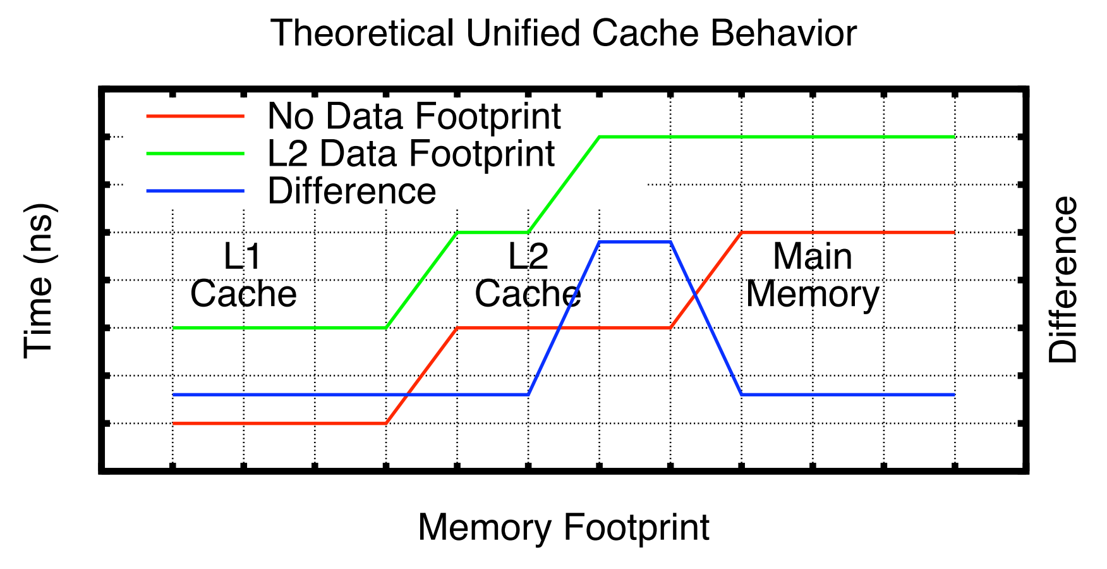
Difficulties

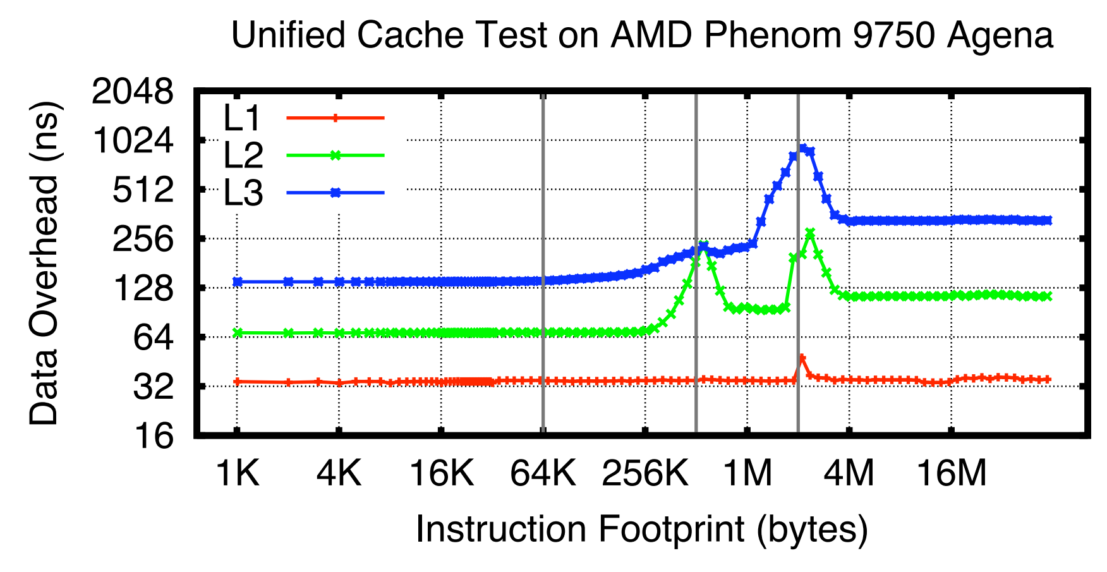
- Hardware
  - Must overwhelm instruction fetch
  - Must fool branch prediction
  - Must detect unified caches
- Software
  - Cannot precisely control/measure code size
  - Cannot eliminate all data accesses
  - Must scale as code size increases
- Analysis
  - Must automatically interpret noisy or unexpected results

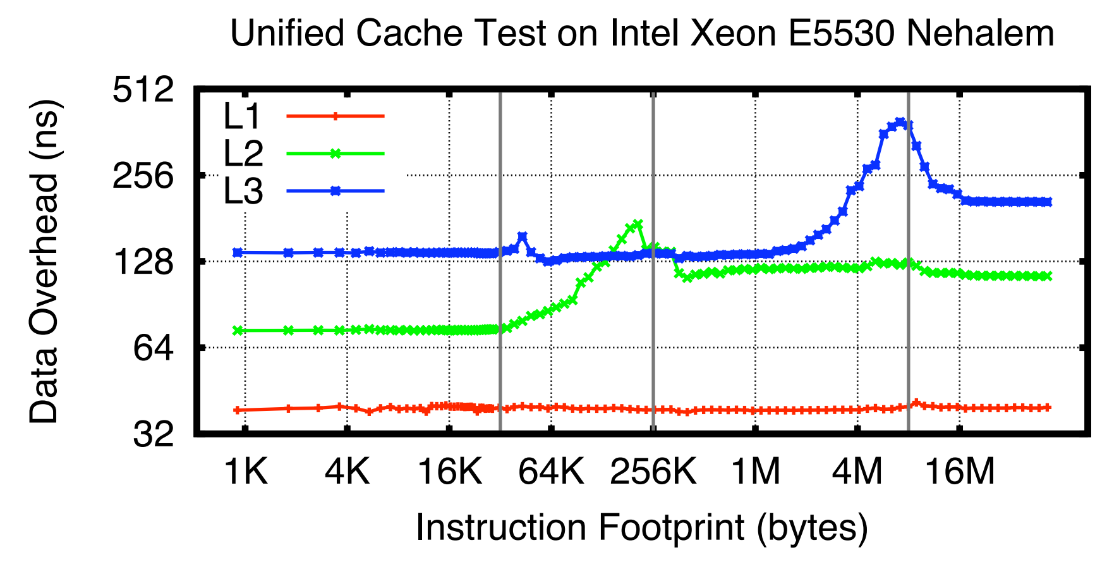
Detecting Unified Cache

Accessing data in a disjoint data cache will not affect instruction cache performance, while accessing data in a unified cache will cause conflicts between instruction and data accesses.







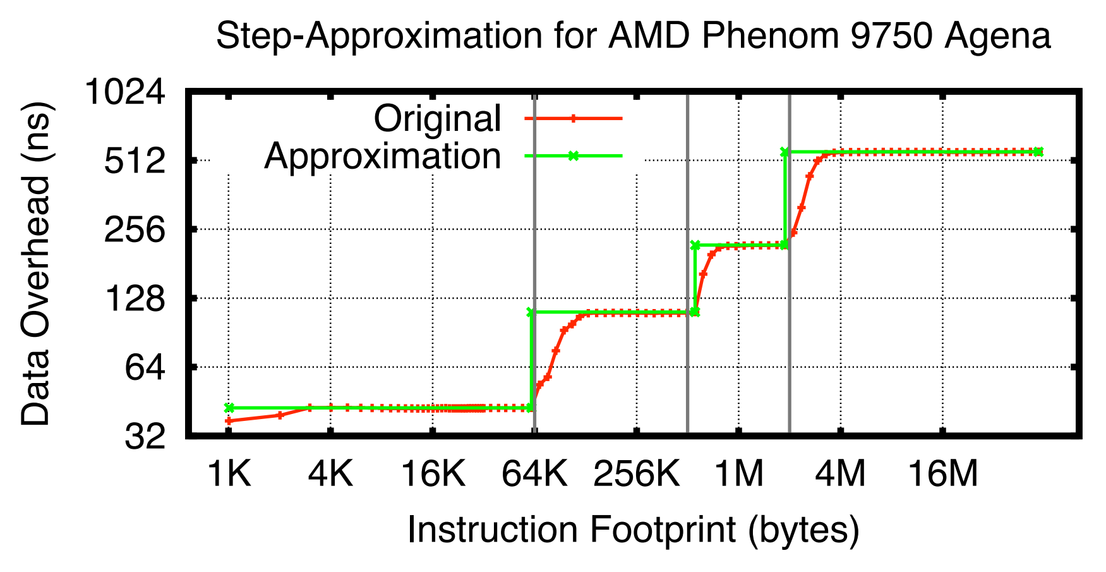


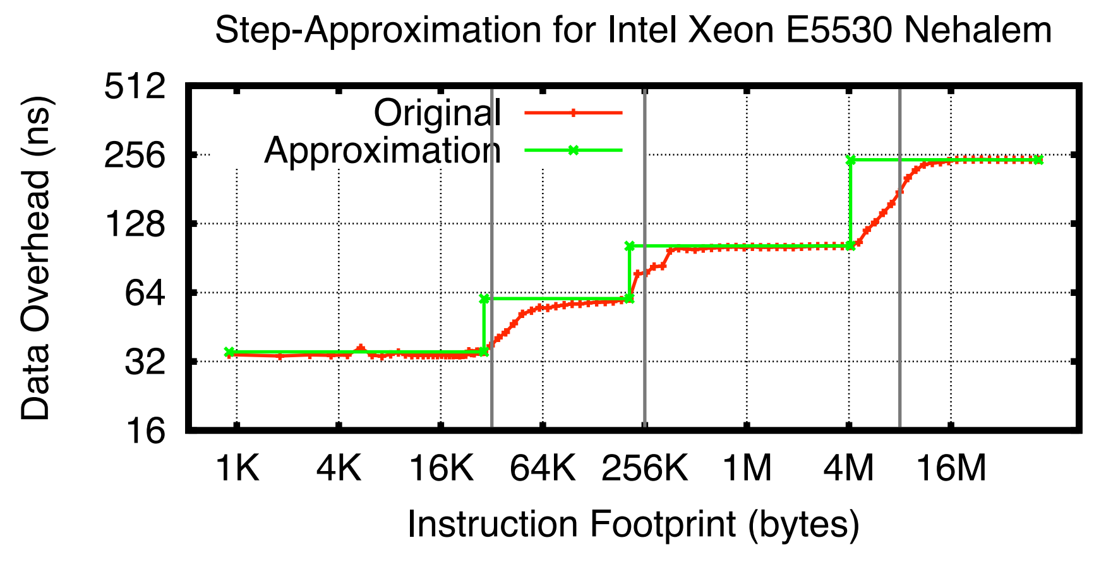
Automatic Analysis

We cannot rely on human intervention for analysis, because doing so would require an expert and may become subjective in the presence of ill-formed data.

- Insights: we know the expected shape of the graph
- Solution: modified polygonal approximation*
  - Captures the general trend
  - Identifies precise transition points
  - Interprets conservatively

* Perez, J.-C., and Vidal, E. Optimum polygonal approximation of digitized curves. Pattern Recogn. Lett. 15, 8 (1994), 743–750.





What is effective size?

Effective cache size is the total cache capacity that is available for program use before performance begins to degrade. Effective cache size may be less than actual cache size for several reasons:

- Inclusive caches
- Multi-core shared caches
- Unified caches
- Physical address mapping

RICE UNIVERSITY

This work is funded by the Defense Advanced Research Projects Agency through AFRL Contract FA8650-09-C-7915 with Rice University. The opinions and findings in this document do not necessarily reflect the views of either the United States Government or of Rice University.