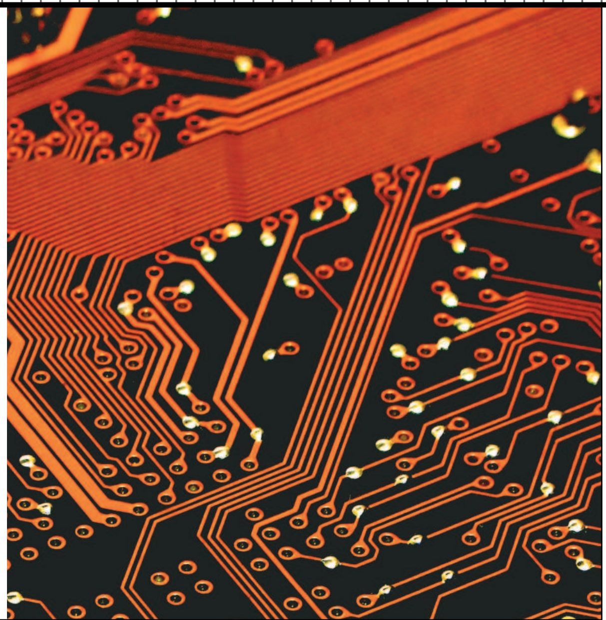
Refactoring for Data Locality



Kristof Beyls, *Tele Atlas*
Erik H. D'Hollander, *Ghent University*

Suggestions for locality optimizations (SLO), a cache profiling tool, analyzes runtime reuse paths to find the root causes of poor data locality, and suggests the most promising code optimizations. Refactoring using the hints of the SLO analyzer doubles the average execution speed of several SPEC2000 benchmark programs.

Refactoring a program means transforming its internal structure to improve its qualities, such as program organization, execution speed, or readability, without changing its functionality. Although refactoring is most often seen as a way to improve a program's internal architecture,[1] here we use the term to mean improving the execution speed. The main bottleneck often is not computation time, but rather memory access delay: Processors can execute hundreds of instructions in the time needed to fetch a single word from main memory.

A cache hierarchy narrows the performance gap between processor and memory. Only when the requested data is present in the cache does the system quickly deliver the data to the processor, saving it from data starvation. Basically, caches operate by retaining the most recently used data. If the processor reuses the data quickly, cache hits occur. Conversely, if it reuses the data after a long time, intervening data can evict the data from the cache, resulting in a *cache miss*. The majority of the processor chip area is typically reserved for caches. However, in many applications, cache misses cause the CPU to stall during more than half of the execution time. In these cases, execution speed benefits more from reducing the number of cache misses than from reducing the number of computations.

Data accessed infrequently exhibits *low temporal data locality*. The corresponding cache miss arises from the instructions touching too much other data between use and reuse. The instruction trace that occurs between use and reuse of the same data is called the *reuse path*, and all code along the reuse path that accesses data contributes to the cache miss. Several cache profiling tools, such as Intel VTune,[2] Cprof,[3] and Cachegrind,[4] measure the hot spots where most cache misses occur and highlight the source code lines with the misses. Those highlighted lines, however, are merely the ends of the reuse paths that generate a cache miss. In many cases, refactoring other code along the reuse path improves the temporal data locality.

Figure 1 shows an example using our *reuse* profiling tool, called *suggestions for locality optimizations* (SLO; http://slo.sourceforge.net). The horizontal highlighted bars indicate the source code lines with cache misses, 95 percent of which occur in Line 5 of the function `inproduct`. Using cache profiling tools, the natural tendency is to rewrite `inproduct` for better cache performance. Unfortunately, refactoring of `inproduct` cannot diminish the data volume the processor accesses between

 Published by the IEEE Computer Society 0018-9162/09/$25.00 © 2009 IEEE

Authorized licensed use limited to: Erik Dhollander. Downloaded on May 19, 2009 at 05:55 from IEEE Xplore. Restrictions apply.

The remainder of this paper is not included as this paper is copyrighted material. If you wish to obtain an electronic version of this paper, please send an email to bib@elis.UGent.be with a request for publication P109.051.pdf.