Introduction: Locality analysis is an important problem in program optimization. Compile time locality analysis can provide detailed feedback related to code structure. Compared to trace based analysis, compiler analysis does not require the program input and does not execute a program.

We present a new technique that is based on the recent Higher-Order Theory of Locality (HOTL). HOTL shows when and how reuse time can be used to derive the cache performance, both for cache misses by Xiang et al. [4] and writebacks by Chen et al. [3] Here reuse time is the number of memory accesses between use and its next reuse. The new technique differs from past techniques in targeting the reuse time instead of the reuse distance, as in [2], or the miss ratio, which we can now compute from the reuse time using HOTL.

In this paper, we show how to derive the reuse time distribution using dependence analysis, in particular the dependence distance described in section 2.2 of [1]. Furthermore, we extend the analysis to reuses at the cache line granularity.

Reuse Time Analysis: Reuse Time Analysis (RTA) assumes a loop nest whose dependence distances for all pairs of memory references to each array are all vectors of constants. Among all the dependences from a single source src, the shortest distance \( v(\text{src}) \) is contributed by its reuse, calculated in Equation 1.

\[
v(\text{src}) = \min\{v(\text{src}, \text{snk}) : \text{snk} \in \text{ref}\}
\]  

By iterating \( v(\text{src}) \) for all the loop ranges containing the source, the reuse time histogram can be constructed. However, not all dependence distances give valid reuses in all iterations. Some sink iterations may be outside the loop bound. For each dependence distance, precise RTA must consider the iteration range for which the sink is valid.

Using Equation 1, we can derive the reuse time of each reference from the dependence distances originated from the reference. This analysis assumes that the granularity of data access is a single data element.

Cache Line Granularity: A cache line contains \( b > 1 \) data elements, which is calculated by the ratio of the cache line size (CLS) to the data element size (DS), \( b = \frac{\text{CLS}}{\text{DS}} \). A cache line reuse can happen between accesses to different data elements. For cache line RTA, we extend the basic RTA with additional information: the data position \( p \) inside the cache line. For 32B cache line size and 8B data element size, \( b \) is 4 and \( p \) ranges from 0 to 3.
Adding the position information, we expand the dependence distance to find the reuses between the source cache line accessed at data position $p_{src}$ and the sink cache line accessed at data position $p_{snk}$. The block granularity dependence distance $v_b$ is given by Equation 2:

$$v_b (src, p_{src}, snk, p_{snk}) = v(src, snk) + (0, 0, ..., 0, p_{snk} - p_{src})$$  

Equation 2 relaxes the requirement of reuse. That is, instead of requiring that the source and the sink access the same data element, it requires that they access the same cache lines. The dependence distance $v_b (src, p_{src}, snk, p_{snk})$ can be derived by adding the position difference between source and its sink to the innermost dimension of the original distance $v(src, snk)$. Note this calculation of distance assumes: (1) The innermost loop accesses the array contiguously. Otherwise it will need more sophisticated calculation, and the result may be the same as that of element granularity. (2) Some iterations of the last dimension of $v_b (src, p_{src}, snk, p_{snk})$ may be outside the loop bound.

The cache line reuse time for source $src$ at position $p_{src}$ is given by $v_b (src, p_{src})$, which is the shortest dependence distance $v_b (src, p_{src}, snk, p_{snk})$ for all its sinks that access the same cache line, shown by Equation 3:

$$v_b (src, p_{src}) = \min\{v_b (src, p_{src}, snk, p_{snk}) | snk \in ref, p_{snk} \in 0...b-1\}$$  

In addition to making sure the sink happens within the iteration space, we also need to make sure the iteration space we are analyzing does not contain a cache line with data across different iterations in higher dimensions (except the innermost dimension). By iterating $v_b (src, p_{src})$ for all the loop ranges containing the source, we obtain the cache line granularity reuse time distribution.

Acknowledgements. We thank Chunling Hu, Kath Knobe, Zoran Budimlic for discussion of the ideas. The research is partially supported by the National Science Foundation (Contract No. CCF-1717877, CCF-1629376) and IBM CAS Faculty Fellowship.

References