All-window Data Liveness

Pengcheng Li and Chen Ding
Department of Computer Science, University of Rochester, Rochester, NY14627, USA
{pli,cding}@cs.rochester.edu

Abstract
This paper proposes a new metric called all-window liveness, which is the average amount of live data in all time windows of a given length. The paper gives a linear-time algorithm to compute the average liveness for all window lengths and discusses potential uses of the new metric.

Categories and Subject Descriptors D.2.8 [Metrics]: Performance measures

General Terms algorithms, performance, measurement

Keywords liveness, Lifetime, reachability, all-window statistics

1. Introduction
Intuitively, liveness shows how much data a program possesses, i.e., allocated but not freed, during its execution. The exact liveness depends on where we look and how long we look in an execution. In other words, the answer changes in different execution windows. One way to have a fixed result is to measure the liveness in all windows and take the average. We call it average all-window liveness. It is unique or deterministic for a given execution.

Traditionally, the liveness of an object is measured by its lifetime. The lifetime is individual but not collective. At each point in execution, we can count the number of live objects, or the population, and compute the average. This is a special case of the all-window metric: the average of all unit-length windows.

The new metric considers all windows of all lengths. A window is a consecutive series of time points. The set of the live data in the window is the union of the population at all time points. The liveness of the window is the size of the union set. The average liveness of all k-length windows, live(k) is the sum of the liveness of all k-length windows divided by the number of k-length windows. The average all-window liveness is defined by the function live(k) for all k. In the following, we will show that the average liveness can be computed precisely and computed in linear time.

All-window liveness is a type of time-varying, multi-scale statistics. We will show an example result in a real benchmark program and discuss possible uses of the new metric in adaptive garbage collection and in memory allocation for heterogenous computing systems.

2. Precise All-window Liveness
The base problem of computing live(k) is counting—to count the number of live objects in each k-length window. We will transform the problem to make it easily solvable.

Instead of counting window by window, we count lifetime by lifetime. Consider an object A whose lifetime spans < s_A, e_A >, that is, from the birth time s_A to the death time e_A. Next consider how many k-length windows in which the object is live. Ignoring boundary conditions (i.e. s_A ≥ k and e_A ≤ n – k + 1, n is the length of the trace), A is live in every k-length window starting from time s_A – k to time e_A. In other words, A contributes one appearance to the liveness and only in e_A – s_A + k windows. Therefore, summing the liveness in all windows is the same as summing the contributed appearances by all objects.

\[ \text{live}(k) = \frac{\sum_{\text{all window } w} (\text{num live objects in } w)}{n - k + 1} \]
\[ = \frac{\sum_{\text{all object } A} \text{contribution}(A)}{n - k + 1} \]  

(1)

The two boundary conditions are for objects born before time k or died after time n – k + 1, where n is the length of the trace. To illustrate, Figure 1 shows three example lifetimes. From left to right in the execution order, the middle lifetime is the normal case and contributes to every k-length window starting from time S_1 – k to time E_1 for E_1 – S_1 + k windows/appearances.

![Figure 1. Three cases when computing the object appearance. Object 1 is born after k and dies before n – k + 1. Object 2 is born before k. Object 3 dies after n – k + 1. The number of appearances is given in “app.”](image)
Using the Java trace tool ElephantTracks [5], we extract both the reachable lifetime and the actual lifetime. Figure 3 shows the liveness curves of the luindex program in the DaCapo benchmark [1]. The gap between the actual and the reachable liveness shows the potential space for improving garbage collection. We are studying how liveness information can be used to choose the best heap size and the nursery size. The figure also shows the approximate liveness, which is not accurate for this program. In most other programs, the approximation is accurate. We will study this discrepancy to understand the object allocation pattern in different programs and find more accurate approximation methods.

\[
live(k) = \sum_{i=1}^{m} (\min(n-k+1, e_i) - \max(k, s_i) + k)
\]

The total time cost to compute \(live(k)\) for all \(k\) is \(O(n)\). The space cost for computing the lifetimes is proportional to the maximal number of the objects that can be alive at one time, which in the worst case is \(O(m)\). The number of window lengths is \(n\). If we consider a logarithmic rather than linear scale and let \(k\) range from 1 to \(\log n\), the space cost of the \(live(k)\) function is \(O(\log n)\).

3. Approximate Liveness

In the precise solution, if we assume \(k \ll n\) and ignore all boundary cases, the liveness simplifies to

\[
live(k) \approx L \frac{n}{n} + \frac{m}{n} k
\]

where \(L = \sum_{all \, object \, \text{A}} \text{lifetime(A)}\) and it is a constant. The average liveness is a linear function with the slope of \(\frac{m}{n}\).

Here is an intuitive view. Let \(w\) be a \(k\)-length window and \(w'\) be the \(k+1\) length window that includes \(w\). Let the liveness of \(w\) and \(w'\) be \(l\) and \(l'\) respectively. If \(w'\) extends \(w\) at the start boundary, \(l'\) is \(l\) plus the birth rate. If \(w'\) extends \(w\) at the end boundary, \(l'\) is \(l\) plus the death rate. Under the assumption that there is no boundary case, the birth rate equals to the death rate, and both equal to \(\frac{m}{n}\), which is the slope of the approximate \(live(k)\) function.

The approximate function is linear, and it is easier to understand and use. Based on the precise equation, we see the necessary conditions for it to be correct. For example, if many objects live till the end of the execution, then the birth rate is greater than the death rate. The approximation will have an error but we can correct it using the precise formula.

4. Future Work

There are three types of lifetimes, as illustrated in Figure 2. The useful lifetime spans between the first and the last access of an object. The reachable lifetime spans between the allocation and the moment the object is unreachable. This time can be measured using the precise formula.

\[
live(k) = \sum_{i=1}^{m} (\min(n-k+1, e_i) - \max(k, s_i) + k)
\]

The total time cost to compute \(live(k)\) for all \(k\) is \(O(n)\). The space cost for computing the lifetimes is proportional to the maximal number of the objects that can be alive at one time, which in the worst case is \(O(m)\). The number of window lengths is \(n\). If we consider a logarithmic rather than linear scale and let \(k\) range from 1 to \(\log n\), the space cost of the \(live(k)\) function is \(O(\log n)\).

Figure 3. The liveness of the luindex program

Another use of liveness may be in heterogeneous computing systems. These systems have more complex memory structures, making it difficult for both memory allocation and data caching [4]. Devices such as GPUs and SoCs use on-chip memory as program-controlled scratch-pad space. A program selects what data to store in the scratch-pad memory and how long to store them. Hence the problem has the aspects of both liveness and locality. Starting with all-window liveness, we plan to develop a higher-theory for memory allocation to mirror the theory for locality. The joint study of liveness and locality also helps to understand cache replacement algorithms. A related metric is the generalized working set (GWS) [2]. GWS is defined by the time-space product. We will study the relation between liveness and GWS and leverage the previous work, especially the algorithm for computing the optimal GWS to optimize locality and liveness together [2].

References