Extending Path Profiling across Loop Backedges and Procedure Boundaries *

Sriraman Tallam Xiangyu Zhang Rajiv Gupta
Department of Computer Science
The University of Arizona
Tucson, Arizona 85721
{tmsriram,xyzhang,gupta}@cs.arizona.edu

Abstract

Since their introduction, path profiles have been used to guide the application of aggressive code optimizations and performing instruction scheduling. However, for optimization and scheduling, it is often desirable to obtain frequency counts of paths that extend across loop iterations and cross procedure boundaries. These longer paths, referred to as interesting paths *in this paper, account for over 75% of the flow in a subset of SPEC benchmarks. Although the frequency counts of* interesting paths *can be estimated from path profiles, the degree of imprecision of these estimates is very high. We extend Ball Larus (BL) paths to create slightly longer* overlapping paths *and develop an instrumentation algorithm to collect their frequencies. While these paths are slightly longer than BL paths, they enable very precise estimation of frequencies of potentially much longer* interesting paths*. Our experiments show that the average cost of collecting frequencies of overlapping paths is 86.8% which is 4.2 times that of BL paths. However, while the average imprecision in estimated total flow of interesting paths derived from BL path frequencies ranges from -38 % to +138 %, the average imprecision in flow estimates derived from overlapping path frequencies ranges only from -4% to +8%.*
Keywords- *path profiles, overlapping path profiles, profile guided optimization, and instruction scheduling.*

1 Introduction

Program profiling has been used extensively to distinguish regions of a program that are hot from rest of the program. Hot regions are portions of the code where the program spends most of its execution time. Execution counts for basic blocks and edges were used extensively to locate hot regions of the code. However, for many applications, such as path-sensitive optimizations and instruction scheduling, the execution frequencies of paths are more desirable. Ball and Larus were the first to introduce path profiling in [3]. Path profiles were designed to subsume basic block and edge profiles. Moreover, Ball and Larus developed instrumentation algorithms that enabled their collection at a reasonable cost. Since their introduction, path profiles have been extensively used to guide the application of path-sensitive code optimizations, performing instruction scheduling, performing code layout optimizations, and improving static branch prediction [5, 6, 7, 8, 9, 12, 13, 14, 20, 11, 1, 10, 21].

Ball Larus paths (BL paths) are designed such that they neither extend across consecutive loop iterations nor do they extend across procedure boundaries. However, experience with some applications shows that this limitation is not desirable. Let us consider the partial redundancy optimizations which can be applied to eliminate redundant executions of expressions [7], array bounds checks [9], loads and stores [8], and conditional branches [5]. Often redundancy of these instructions arises when, during execution of loops, the same instruction is executed multiple times, once during each loop iteration. In [7, 8] it was shown that this situation is common when redundancy in arithmetic expressions and loads is considered. Therefore paths along which such redundancy appears extend *across loop backedges*. Another situation where redundancy is observed is when instruction executed before a call (return) causes another instruction executed after the call (return) to become redundant. In [5] it was shown that this situation is very frequent when redundancy in conditional branches is considered. Thus, paths along which such redundancy was observed extend *across procedure boundaries*. Finally, when instruction scheduling is carried out, if a loop is unrolled once before scheduling (e.g., before trace scheduling [11]), then profiles of paths that correspond to two loop iterations are needed.

*Supported by grants from IBM, Microsoft, Intel and NSF grants CCR-0324969, CCR-0220334, CCR-0208756, CCR-0105355, and EIA-0080123 to the Univ. of Arizona.

From the above discussion it should be clear that making available the frequencies of paths that extend across loop iterations and procedure boundaries is useful. One approach to obtaining frequencies of paths that are longer than BL paths is to make use of *whole program paths* (WPPs) which consist of a complete control flow trace of a program [16, 22]. However, in contrast to path profiles, WPPs are expensive to collect and require large amounts of storage to save. An alternative approach is to estimate the frequencies of longer paths from BL paths – this is analogous to the approach developed in [4] to estimate the lower and upper bounds on the frequency estimates of BL paths from edge profiles. However, as we show in this paper, this approach yields highly imprecise estimates.

Table 1. Flow attributable to interesting paths.

Benchmark	Flow of paths across		Total Flow
	Loop Backedges	Procedure Boundaries	
130.li	19.9 %	70.2 %	90.1 %
099.go	32.3 %	52.4 %	84.7 %
134.perl	9.6 %	75.9 %	85.5 %
008.espresso	56.4 %	26.1 %	82.5 %
147.vortex	2.1 %	94.1 %	96.2 %
197.parser	16.2 %	72.7 %	88.9 %
181.mcf	27.8 %	54.2 %	82.0 %
300.twolf	69.1 %	13.9 %	83.0 %
126.gcc	26.1 %	50.8 %	76.9 %

In this paper we address the problem of collecting profiles for paths longer than BL paths. We begin by characterizing the paths whose execution counts we desire to obtain. We will refer to these paths as the *interesting paths* for rest of the paper. The interesting paths for loops are the ones that extend across two consecutive loop iterations. Interesting interprocedural paths are formed in two ways. BL subpaths in the caller leading to a call site are concatenated with the BL paths that begin at the callee's entry. Also BL paths terminating at the callee's return point are concatenated with BL subpaths following the call site in the caller. The interesting paths as defined above are not only relevant for the applications discussed earlier, they also happen to account for most of the program's execution time. Data in Table 1 supports this assertion. The percentage of total flow (i.e., sum of the frequency counts of all BL paths) that is covered by all interesting paths varies from 76.9% to 96.2%. It is also clear that both types of interesting paths, those that cross loop backedges and those that cross procedure boundaries, account for significant part of the flow.

While a straightforward approach for obtaining profiles for interesting paths would be to instrument the program to collect such profiles, this approach can be too expensive. The increase in number of paths whose frequency counts would have to be maintained could greatly exceed the number of BL paths. For example, in the benchmark 099.go, a function has 283063 static loop paths and this would give 283063 * 283063 two iteration paths. Therefore we develop an alternative approach that allows control over this cost. We extend BL paths to create a new set of paths that are longer than BL paths but shorter than interesting paths. The extended paths overlap each other and thus we refer to them as *overlapping paths*. The amount by which BL paths are extended to create overlapping paths can be selected to control the cost and this amount is referred to as the *degree* of overlap. Using overlapping paths, the increase in space is not so dramatic. For the above mentioned function in 099.go, overlapping paths of degree 1 are 283063 * 2 in number and for degree 2 there are 283063 * 4 paths. Once we have collected the frequency counts of overlapping paths by running an instrumented program, we use them to derive estimates (lower and upper bounds) on the frequency counts of interesting paths.

We develop the instrumentation algorithms to collect profiles of overlapping paths and develop algorithms for deriving estimates of frequency counts of interesting paths from profiles of overlapping paths. Each of these algorithms are parameterized with respect to the degree of overlap. We have implemented these algorithms to compare the effectiveness and costs of deriving estimates of interesting path profiles from BL paths and overlapping paths. Our experiments show that while overlapping paths are slightly longer than BL paths, they enable very precise estimation of frequencies of potentially much longer interesting paths.

The remainder of this paper is organized as follows. In section 2 we consider paths that cross loop boundaries. We develop algorithms for computing lower and upper bounds on the frequencies of interesting paths from exact frequencies of overlapping paths collected by instrumenting loops. In section 3 we repeat the same process for paths that cross procedure boundaries. In section 4 we present results of experiments. Conclusions are given in section 5.

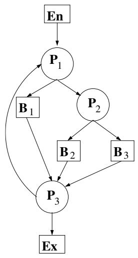
2 Paths Crossing Loop Backedges

Interesting paths are those paths that cross a loop backedge only once and they are paths that correspond to two consecutive iterations of a loop. *Overlapping paths* (OL paths) are shorter than two iterations and OL path profiles are used to estimate the execution frequencies of the interesting paths. The equations for deriving these estimates are parameterized by the degree of overlap. Overlap of zero corresponds to BL paths.

2.1 Overlapping Paths

The overlapping paths are constructed by simply extending BL paths. Consider the control flow graph and its corresponding BL paths in Table 2. The basic blocks marked with a 'P' are predicate blocks (i.e., they end in a conditional branch). There are 12 BL paths in this example that are divided into four groups: (i) Paths beginning at the basic block 'En' and ending at 'Ex'. Paths 1, 2 and 3 belong to this group; (ii) Paths beginning at the basic block 'En' and ending at 'P_3' and immediately followed by the backedge. Paths 4, 5 and 6 belong to this group; (iii) Paths beginning at 'P_1' following the backedge and ending at 'P_3' and immediately followed by the backedge. Paths 7, 8 and 9 belong to this group; and (iv) Paths beginning at 'P_1' following the backedge and ending at 'Ex'. Paths 10, 11 and 12 belong to this group.

Table 2. Ball-Larus paths in a CFG.



Group Id	Path Id	Program Path
1	1	$En \Rightarrow P_1 \Rightarrow B_1 \Rightarrow P_3 \Rightarrow Ex$
	2	$En \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3 \Rightarrow Ex$
	3	$En \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3 \Rightarrow Ex$
2	4	$En \Rightarrow P_1 \Rightarrow B_1 \Rightarrow P_3$
	5	$En \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3$
	6	$En \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3$
3	7	$P_1 \Rightarrow B_1 \Rightarrow P_3$
	8	$P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3$
	9	$P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3$
4	10	$P_1 \Rightarrow B_1 \Rightarrow P_3 \Rightarrow Ex$
	11	$P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3 \Rightarrow Ex$
	12	$P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3 \Rightarrow Ex$

We define an interesting path for the loop as the one that has a basic block sequence that starts with the loop entry node, P_1 in the above CFG, and corresponds to two iterations of the loop. As an example, $(P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3)$ is one such interesting path. This path is nothing but the basic block sequence of BL path 7 followed by BL path 8 and is denoted as $(7 \; ! \; 8)$, where '!' is used to mark the position of the backedge. Note that this interesting path also appears in $(4 \; ! \; 8)$, $(7 \; ! \; 11)$ and $(4 \; ! \; 11)$. Thus the frequency of the interesting path is the sum of the frequencies of these four paths. For the loop shown above, there are exactly 9 interesting paths formed from all pairs of the basic block sequences of the 3 loop paths. However, the basic block sequences corresponding to these 9 interesting paths appear in 36 different pairs of BL paths. In general $(i \; ! \; j)$ is used to denote that interesting path whose basic block sequence appears in the two iterations of the loop contained in $(i \; ! \; j)$.

We define an overlapping path, or an OL path, as the one that is obtained by extending a BL path to cross the backedge and then terminating at any predicate within the loop. By this definition, all the interesting paths are overlapping paths. More specifically, we define an overlapping path to be a k-overlapping path, OL-k path, if it terminates at the $(k+1)^{th}$ predicate block following the backedge (the terminating block of the loop is also treated as a predicate block). We also call k to be the degree of the overlapping path. For example, a OL-0 path for the above loop would be, $(P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1)$. Notice that this path is equivalent to BL path 7. Though P_1 is not included in 7, it is trivial to see that P_1 is the only basic block that can be executed after the backedge is taken. Similarly, paths 4 to 9 are all OL-0 paths, since they end at predicate block P_1. In other words, OL-0 paths are BL paths unextended. However, consider the OL-1 path, $(P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow P_2)$. It is clear that this path belongs to OL-1 as it ends at the 2nd predicate block following the backedge. Note that this path is nothing but BL path 7 extended to P_2, that is, extended by 1 predicate block. The basic block sequences for few OL paths of different overlaps are shown in Table 3. Also, the number of OL paths of each degree are shown.

Table 3. Overlapping paths in the CFG.

Degree k	OL-k Path Examples	Num. of Paths
0	$En \Rightarrow P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1$, $P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3 \; ! \; P_1$	6
1	$En \Rightarrow P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow B_1 \Rightarrow P_3$, $P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow P_2$	12
2	$P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3$	12

Notice that the maximum overlap possible for the loop in Table 2 is only 2. Also, notice that paths corresponding to this overlap are interesting paths. As stated earlier, when we estimate the profiles of the interesting paths, we choose a degree of overlap depending on the cost we are willing to incur. When we profile overlapping paths of degree k, we also profile all other OL paths whose degree is less than k but are not contained in the OL-k paths. Notice that as we increase the degree of overlap, the number of paths that have to be profiled increases and hence, the cost of profiling them.

The next section discusses the equations involved in estimating the flow across the interesting paths. By flow of a path we mean the frequency of a path.

2.2 Estimating Flow

Consider a loop which has k loop paths, numbered $1 \ldots k$, in the depth first search order. The number of different interesting paths in this loop is k^2. Let F_i denote the flow of the i^{th} loop path. If the frequency of the backedge is B, then, the following equation holds.

$$F_1 + F_2 + \ldots + F_k \geq B. \tag{1}$$

Now, we wish to estimate the frequencies of the k^2 interesting paths, which are nothing but all pairs ($i \; ! \; j$) formed out of the k loop paths. Let $F_{i \,!\, j}$ denote the real flow of the interesting path $i \; ! \; j$. Let $OF_{i \,!\, k(P_j)}$ denote the frequency of the OL path, $i \,!\, k(P_j)$, that begins with basic block sequence i, takes the backedge, and then follows the basic block sequence contained in path k, however, terminating at predicate P_j. If more than one OL path contains this basic block sequence as its suffix, we add their frequencies to obtain $OF_{i \,!\, k(P_j)}$. As an example, for the loop in Table 4, $OF_{1 \,!\, 2(P_2)}$ will be the flow of the OL path that has the following basic block sequence: $P_1 \Rightarrow B_1 \Rightarrow P_3 \; ! \; P_1 \Rightarrow P_2$.

$OF_{i \,!\, k(P_j)}$ can be computed using the following equation.

$$OF_{i \,!\, k(P_j)} = \sum F_{i \,!\, l} \tag{2}$$

where $i \; ! \; l$ are all the interesting paths that contain the basic block sequence of the OL path ($i \; ! \; k(P_j)$). For instance, for the loop in Table 4, $OF_{1 \,!\, 3(P_2)} = F_{1 \,!\, 2} + F_{1 \,!\, 3}$, and this can be easily proved.

We derive the estimates for the frequencies of the overlapping paths by getting the upper and lower bounds of the frequencies of the paths in the following sections.

2.2.1 Generating the Upper Bounds of the Frequencies

Let us assume we profile overlapping paths of some degree less than or equal to θ. Let us now find the upper bound of the frequency of the interesting path $p \; ! \; q$. Let $L_{p \,!\, q}$ and $U_{p \,!\, q}$ denote the lower and upper bounds for the flow of path $p \; ! \; q$ respectively. Also, let $OF_{p \,!\, a(P_j)}$ be the frequency of the OL path of degree less than or equal to θ we profiled such that,

$$OF_{p \,!\, a(P_j)} = F_{p \,!\, q} + F_{p \,!\, r} + \ldots F_{p \,!\, s} \tag{3}$$

where q, r, $\ldots$, s all have the same prefix as a, the prefix ending in P_j.

From the above equation a possible value for $U_{p \,!\, q}$ is the right hand side of the inequality below.

$$F_{p \,!\, q} \leq OF_{p \,!\, a(P_j)} - L_{p \,!\, r} - \ldots - L_{p \,!\, s} \tag{4}$$

Subtracting the lower bounds of all paths other than $p \; ! \; q$ from $OF_{p \,!\, a(P_j)}$ gives us a candidate upper bound for the flow of path $p \; ! \; q$. We can derive two other candidates for $U_{p \,!\, q}$ as follows. Let E_q denote the number of times loop path q was executed as the first iteration and X_p be the number of times path p was executed as the last iteration. The value of E_q and X_p can be directly obtained from the BL path profile. Clearly,

$$F_{p \,!\, q} \leq F_p - X_p. \tag{5}$$

Also,

$$F_{p \,!\, q} \leq F_q - E_q. \tag{6}$$

Using equations 4, 5 and 6 we can compute U_{pq} to be

$$U_{p \,!\, q} = MINIMUM \left\{ \begin{array}{c} (OF_{p \,!\, a(P_j)} - L_{p \,!\, r} - \ldots - L_{p \,!\, s}), \\ (F_p - X_p), \;\; (F_q - E_q) \end{array} \right\}. \tag{7}$$

The minimum of the three candidate upper bounds ensures that our upper bound estimate is tight.

Equation 4 clearly shows that the upper bounds of the frequencies could depend on the lower bounds. Similarly, we will show in the next section that the lower bounds of the frequencies also depend on the upper bounds. This means that the equations have to be computed iteratively till the values of the bounds stabilize. However, to compute the initial values of the upper bounds, we can assume the lower bounds of all other paths to be zero.

2.2.2 Generating the Lower bounds of the Frequencies

Again, consider Equation 3. Using a similar reasoning as in the previous case, we can compute the lower bound of the frequency of the interesting path $p \; ! \; q$ to be

$$L_{p \,!\, q} = MAXIMUM\{ \, (OF_{p \,!\, a(Pj)} - U_{p \,!\, r} - \ldots - U_{p \,!\, s}), \; 0\} \tag{8}$$

The above equation shows that the lower bounds do depend on the upper bounds. The next section shows the use of these equations in an example.

As mentioned in [4], the sum of the lower (upper) bounds of the frequencies of the interesting paths is referred to as the Definite (Potential) flow.

2.2.3 An Example

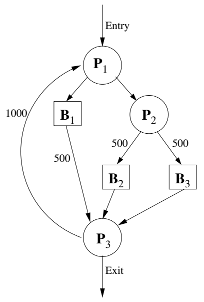
Consider the control flow graph shown in Table 4. It is the same loop as used before except that numbers mark the flow values of different loop paths. The basic block sequences of the 3 loop paths are shown below.

$$\begin{array}{c} 1 : P_1 \Rightarrow B_1 \Rightarrow P_3 \\ 2 : P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow P_3 \\ 3 : P_1 \Rightarrow P_2 \Rightarrow B_3 \Rightarrow P_3 \end{array}$$

Consider the following execution history of the loop during a program run. Lets say the loop was entered 500 times from outside. For the first 250 times, the path corresponding to the sequence 1 ! 1 ! 3 was taken and for the remaining 250 times, the path corresponding to the sequence 2 ! 2 ! 3 was taken. Thus, the frequencies of sequences 1, 2 and 3 are all 500. The frequency of the backedge B is 1000. This also means that E_1 and E_2 are 250 and X_3 is 500. Also, the only interesting paths that have non-zero frequencies are shown in Table 4.

Now, let us compute the estimates of the frequencies of the interesting paths using OL-0 (i.e., BL) paths. In particular let us first get estimates for interesting paths beginning with sequence 1. The frequency of OL-0 path of interest is

Table 4. Frequencies of interesting paths.



Interesting Path	Frequency
1 ! 1	250
2 ! 2	250
1 ! 3	250
2 ! 3	250

$OF_{1\,!\,1(P_1)}$ and is equal to 500, since $OF_{1\,!\,1(P_1)} = F_{1\,!\,1} + F_{1\,!\,2} + F_{1\,!\,3}$. Now, applying these values in Equations 7 and 8 we get, the following values: $U_{1\,!\,1} = 250,\ L_{1\,!\,1} = 0, U_{1\,!\,2} = 250,\ L_{1\,!\,2} = 0$ and $U_{1\,!\,3} = 500,\ L_{1\,!\,3} = 0$. The estimates of the other paths can be similarly computed and are given in Table 5.

Now to get the estimates of interesting paths beginning with sequence 1 using OL-1 paths, we use the frequency $OF_{1\,!\,2(P_2)}$ and $OF_{1\,!\,1(P_3)}$. Notice that $OF_{1\,!\,1(P_3)}$ is exactly equal to $F_{1\,!\,1}$. Again, we apply the equations as before and the estimates obtained are shown in the table.

Table 5. Estimates using OL paths.

Interesting Path	Real Frequency	Lower Bound		Upper Bound	
		OL-0	OL-1	OL-0	OL-1
1 ! 1	250	0	250	250	250
1 ! 2	0	0	0	250	250
1 ! 3	250	0	0	500	250
2 ! 1	0	0	0	250	0
2 ! 2	250	0	0	250	250
2 ! 3	250	0	0	500	250
3 ! 1	0	0	0	0	0
3 ! 2	0	0	0	0	0
3 ! 3	0	0	0	0	0

The real flow in the loop is the sum of the frequencies of the interesting paths and is equal to 1000. The definite and potential flows that we get using OL-0 paths are 0 and 2000 which are off the real flow by $\pm 100\%$. Whereas, using OL-1 paths, the flows are 250 and 1250, off by -75% and +25%. Also, notice that OL-2 paths should give us accurate profiles since the maximum overlap of this loop is 2.

2.3 Instrumentation

We extend the algorithm proposed by Ball and Larus so that instead of collecting BL path profiles, we can collect overlapping path profiles. This algorithm has two parts: deriving an acyclic *path graph* which is used to determine the required instrumentation and carrying out the *instrumentation* of the original code.

Deriving the Path Graph. Ball and Larus derive an acyclic graph which contains exactly the BL paths. This graph is used as a basis of assigning ids to paths and edges of this graph are assigned increment values. The sum of the increments on the edges along a given path generates the path id. Figure 1 shows a CFG in (a) which is converted into (b) to determine instrumentation for BL paths. To determine instrumentation for overlapping paths of degree 2, the graph in (c) is used. As we can see, an additional portion of the loop body is included to extend the paths upto the lengths of the desired overlapping paths. This additional portion of the *path graph* is called the *overlapping graph*.

Let us consider the construction of the path graph in detail. As a first step, given a loop body L and overlap k, three sets of edges are identified in L.

- *Definitely instrumented [DI] edges* , the number of predicates along all the paths to these edges is less than or equal to k. In Figure 1(c), edges $P'_1 \rightarrow P'_2$ and $B'_1 \rightarrow P'_3$ are examples for DI edges for overlap degree of 2.

- *Possibly instrumented [PI] edges*, the number of predicates along some paths to these edges is less than or equal to k. In Figure 1(c), edge $P'_3 \rightarrow B'_2$ is PI edge. Because path $P'_1 \rightarrow B'_1 \rightarrow P'_3 \rightarrow B'_2$ has 2 predicates and path $P'_1 \rightarrow P'_2 \rightarrow P'_3 \rightarrow B'_2$ has 3 predicates.

- *Definitely not instrumented [DNI] edges*, the number of predicates along all the paths to these edges is greater than k. In Figure 1(b), edge $P_3 \rightarrow B_2$ is DNI edge for overlap 1. Because all the paths from loop head P_1 to this edge has more than 1 predicates.

Given the DI, PI and DNI edges in a loop body L, the *overlapping graph* is constructed as follows: all DNI edges are removed from L; DI edges are distinguished from the PI edges (PI edges are drawn using dotted lines in the figure); and nodes which are not reachable from loop head are removed. The shaded area in Figure 1(c) contains the overlapping graph for overlap 2.

We first generate the path graph of Figure 1(b) from the CFG of Figure 1(a) using Ball and Larus algorithm, which removes the backedge $P_4 \rightarrow P_1$, adds dummy edges from En to P_1 and from P_4 to Ex. Next we replace the dummy edge from P_4 (loop exit) to Ex with an edge from P_4 to P'_1, the head of OG for the loop. For any node N in OG, if there is a path from loop head to N having $k+1$ predicates, we connect N to Ex with a dummy edge. For example, P'_3 in (c) and P'_4 in (c) are connected to Ex with dummy edges. Finally we assign chord increments to edges according to Ball and Larus's algorithm. The generated graph is the desired *path graph*.

Instrumentation. The steps of the instrumentation algorithm for a given CFG and its corresponding path graph PG are as follows:

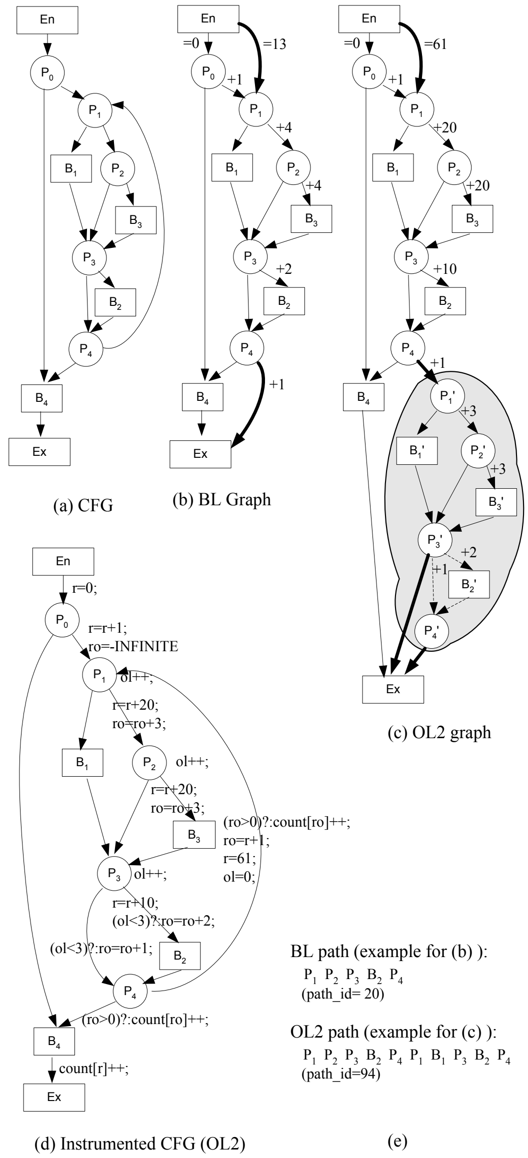


Figure 1. Instrumenting for OL paths.

- If an edge e of CFG is marked '$op\ x$' in the white area of PG and it is not a dummy edge, it is instrumented with '$r = r\ op\ x$'.
- If an edge e of CFG is marked '$op\ x$' in the dark area and it is a non-dummy DI edge in PG, it is instrumented with '$ro = ro\ op\ x$'.
- If e is marked '$op\ x$' in the dark area and it is a non-dummy PI edge in PG, it is instrumented with '$(ol < k)? : ro = ro\ op\ x$'.
- If e is the entry edge to the loop, it is instrumented with '$ro = -\infty$'.
- If p is a predicate inside the loop, it is instrumented with '$ol++;$'.
- If e is the exit edge from the loop, it is instrumented with '$(ro > 0)? : count[ro]++;$'.
- If e is the backedge, and the dummy edge from En to loop head is marked with '$= x$', the dummy edge entering the dark area is marked with '$op\ \ y$', the backedge is instrumented with:

  $(ro > 0)? : count[ro]++;$
  $ro = r + y; r = x; ol = 0;$

- If e is an edge to Ex, it is instrumented with '$count[r]++;$'.

The instrumented CFG for our running example is shown in Figure 1(d).

3 Paths Crossing Procedure Boundaries

In this section, we consider interesting and overlapping paths that cross procedure boundaries. These paths are the ones that start in one procedure and end in another procedure. It should be noted that while Melski and Reps [17] have also proposed an approach for profiling interprocedural paths, their approach is too expensive for use in practice. They create a single *supergraph* which connects all procedures and then use Ball Larus method to enumerate the paths in this combined flow graph. The number of paths will increase dramatically in this approach. Moreover, the presence of function pointers complicates the construction of the supergraph. Our approach based upon overlapping interprocedural paths is much more practical.

3.1 Interprocedural Overlapping Paths

Table 6. OL-k Paths of Type I in the CFG.

Degree k	I-OL-k Path Examples	Number of Paths
0	$fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1\ !\ gEn \Rightarrow P_1$	3
1	$fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1\ !\ gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow gEx$	6
2	$fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1\ !\ gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow P_3$	6
3	$fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1\ !\ gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow P_3 \Rightarrow B_3 \Rightarrow gEx$	12

Consider the interprocedural control flow graph shown in Figure 2. The CFG for f() has a call site where function

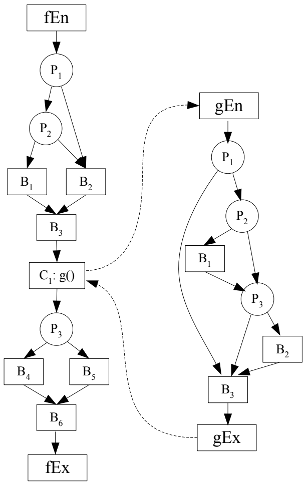


Figure 2. CFGs for two functions.

Table 7. OL-k Paths of Type II in the CFG.

Degree k	II-OL-k Path Examples	Number of Paths
0	$gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow$ $gEx\,!\,C_1 \Rightarrow P_3$	5
1	$gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow gEx\,!$ $C_1 \Rightarrow P_3 \Rightarrow B_4 \Rightarrow B_6 \Rightarrow fEx$	10

g() is called. There are two types of interesting paths we want to look at. The first type of paths start in f() and end in g(). The second type of paths start in g() and end in f(). Examples of paths are given below.

Type I: $fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1\,!\,gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow gEx$
Type II: $gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow gEx\,!\,C_1 \Rightarrow P_3 \Rightarrow B_4 \Rightarrow B_6 \Rightarrow fEx$

Let us call them interesting paths of Type I and Type II. Notice, that there are 15 different paths of Type I and 10 different paths of Type II. For paths of Type I, there are 3 different ways of reaching call site C_1 from fEn and each of these could end with any of the 5 paths in g() giving rise to a total of 15 different paths. A similar argument holds for paths of Type II. To estimate the profiles of these interesting paths, we form overlapping paths as follows. To accommodate the interesting paths of Type I, we break the BL paths of function f() at the call site and extend them into function g(), the extension being dependent on the overlap in question. Lets call these overlapping paths as I-OL paths, I used to indicate that they correspond to interesting paths of Type I.

I-OL paths are defined as follows. A I-OL-k path starts at the node fEn and extends into function g() after the call site and stops at the $(k+1)^{th}$ predicate node following C1 in g(). Table 6 shows some I-OL paths of different degrees. We assume that gEx is a predicate node. Note that the maximum degree for the example CFG is 3.

Now, let us discuss the overlapping paths of Type II, called II-OL paths. These paths correspond to the interesting paths of Type II and are used to estimate their frequencies. A II-OL path of degree k, written as II-OL-k, is a BL path in g() that extends into function f() after the return, and ends at the $(k+1)^{th}$ predicate block following the return edge. Table 7 shows some II-OL paths of different degrees. We assume that fEx is a predicate node. Notice that the maximum degree possible in the CFG is 1.

3.2 Estimating Flow

This section presents the equations involved in estimating the flow of paths across procedure boundaries. First, the estimation process using BL paths is described and then the estimation using overlapping paths is presented.

3.2.1 Estimation using BL Paths

Consider an interprocedural CFG of two functions f() and g(). Let f() be the caller and g() be the callee. Let us consider a call site in f() that has a call to function g(). Let us assume that there are k paths in function f() starting from the entry node to the call site. Let us number them $1, \ldots, k$ according to a depth first search order. Also, let us assume there are l paths in function g() numbered $1, \ldots, l$ similarly. Let us first estimate the flow of Type I paths. The analysis for Type II paths is similar.

Now, all the interesting paths of Type I have the form $p\,!\,q$, where p and q are the paths from f() and g() respectively. Let $F_{p\,!\,q}$ represent the actual frequency of interesting path $p\,!\,q$. Let $U_{p\,!\,q}$ and $L_{p\,!\,q}$ be its upper and lower bounds. Also, let F_p (F_q) be the frequency of the basic block sequence represented by p (q). We can find the frequencies of each of these basic block sequences by summing the frequencies of all BL paths that contain these sequences. Let C be the number of calls made from f() to g(). This can be obtained by instrumenting the program appropriately. Then, the following equation holds.

$$\sum_{i=1,j=1}^{i=k,j=l} F_{i\,!\,j} = C \qquad (9)$$

From equation 9 we can derive a candidate estimate for the upper bound to be the right hand side of the following equation.

$$U_{p\,!\,q} \leq C + L_{p\,!\,q} - \Big(\sum_{i=1,j=1}^{i=k,j=l} L_{i\,!\,j}\Big) \qquad (10)$$

Also, we can derive two other candidate estimates for $U_{p\,!\,q}$ as follows.

$$U_{p\,!\,q} \leq F_p \qquad (11)$$

and

$$U_{p\,!\,q} \leq F_q \qquad (12)$$

Using equations 10, 11 and 12, we can compute the value of $U_{p\,!\,q}$ to be

$$U_{p\,!\,q} = MINIMUM \left\{ \begin{array}{l} (C + L_{p\,!\,q} - \sum_{i=1,j=1}^{i=k,j=l} L_{i\,!\,j}), \\ F_p,\ F_q \end{array} \right\} \tag{13}$$

Equation 13 shows that the upper bound is dependent on the lower bounds. We will also show by Equation 14 that the lower bound is also dependent on the upper bounds. Hence, we need to recompute the bounds iteratively till their values stabilize. For, computing the initial values of upper bounds, we can assume that the lower bounds are zeroes. Equation 14 shows the equation for calculating the lower bound.

$$L_{p\,!\,q} = MAXIMUM\{\ (C + U_{p\,!\,q} - \sum_{i=1,j=1}^{i=k,j=l} U_{i\,!\,j}),\ 0\ \} \tag{14}$$

The equations for estimating the frequencies of the interesting paths of Type II are similar.

3.2.2 Estimation using Overlapping Paths

Let us derive equations for estimating the frequencies of the interesting paths using overlapping paths. Again, let us first derive estimates of interesting paths of Type I. The analysis of Type II follows. Let $OF_{p\,!\,k(P_j)}$ be the frequency of the overlapping path that starts with the basic block sequence of the p^{th} path in function f(), and follows the basic block sequence of the k^{th} path in function g(), however, terminating at predicate P_j in function g(). For instance, in the CFG in Figure 2, $OF_{3\,!\,1(P_1)}$ represents the frequency of the overlapping path given below.

$$fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1 \Rightarrow gEn \Rightarrow P_1$$

Now, the following equation holds

$$OF_{p\,!\,k(P_j)} = \sum F_{p\,!\,l} \tag{15}$$

where l represents all the sequences in g() that have the same prefix as k, prefix ending in predicate P_j.

Let $OF_{p\,!\,k(P_j)}$ be the frequency of the overlapping path of degree θ we profiled such that

$$OF_{p\,!\,k(P_j)} = F_{p\,!\,q} + F_{p\,!\,r} + \ldots \tag{16}$$

the equation got by merely expanding the right hand side of Equation 15. We can then compute the upper bounds of frequency of path $p\,!\,q$ using Equation 16 as follows.

$$U_{p\,!\,q} = MINIMUM\{\ (OF_{p\,!\,k(Pj)} - L_{p\,!\,r} - \ldots),\ F_q\ \} \tag{17}$$

and then compute the lower bounds using Equation 18 as shown below.

$$L_{p\,!\,q} = MAXIMUM\{\ (OF_{p\,!\,k(Pj)} - U_{p\,!\,r} - \ldots),\ 0\ \} \tag{18}$$

As before, since the values of upper and lower bounds are dependent on each other, we recompute them iteratively till they stabilize.

The equations for the upper and lower bounds are similar for Type II and can be derived analogously.

3.2.3 An Example

Consider the interprocedural CFG shown in Figure 2. The paths in function f() are

$$\begin{array}{c} 1: fEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_1 \Rightarrow B_3 \Rightarrow C_1 \\ 2: fEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1 \\ 3: fEn \Rightarrow P_1 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow C_1 \end{array}$$

and the paths in function g() are

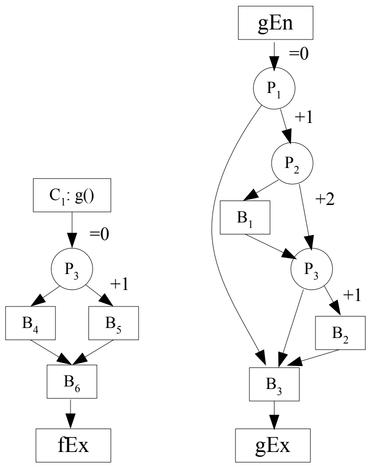
$$\begin{array}{c} 1: gEn \Rightarrow P_1 \Rightarrow B_3 \Rightarrow gEx \\ 2: gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_1 \Rightarrow P_3 \Rightarrow B_3 \Rightarrow gEx \\ 3: gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow B_1 \Rightarrow P_3 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow gEx \\ 4: gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow P_3 \Rightarrow B_3 \Rightarrow gEx \\ 5: gEn \Rightarrow P_1 \Rightarrow P_2 \Rightarrow P_3 \Rightarrow B_2 \Rightarrow B_3 \Rightarrow gEx \end{array}$$

Consider an execution in which the number of calls made from f() to g() is 100. Also, let 1 ! 1 is the only interesting path to have a non-zero frequency of 100. Also, let the frequency of each of the 8 sequences shown above be 200. An estimate using BL path profiles alone would give a flow of anywhere between 0 and 100 for all 15 interesting paths. Whereas, an estimate using I-OL-1 paths would give us exact values. Note that the maximum overlap possible is 3.

3.3 Instrumentation

In the previous section, we mentioned that there are two types of interprocedural overlapping paths. In this section, we explain how to instrument these paths. For the instrumentation of overlapping loop paths we simply extended BL's instrumentation by enumerating the overlapping loop paths. But it is inappropriate to make a similar extension for interprocedural overlapping paths because the existence of function pointers can make the extended graph very huge. So we use a different instrumentation for interprocedural overlapping paths. Instead of using an one dimensional counter array, we use four dimensional array $count[i][j][k][l]$ where i is the function id of the callee, j is the global id for the call site, k is the path id in callee, and l is the path id in the caller. In other words, we use a four tuple to represent an interprocedural overlapping path.

We demonstrate how to do instrumentation by an example. Figure 2 shows the CFGs for two functions f and g. There is a call C_1 to function g in f. We are going to instrument the II-OL-1 paths in this example. For the call site C_1, we construct the *Overlapping Graph* (OG) of Figure 3(a) in a way similar to the overlapping loop paths. The entry node of OG here is a call site instead of a loop head. The caller's parts of the interprocedural paths are enumerated by this OG. The enumeration graph for the callee is the same as BL's enumeration graph. Given these enumeration graphs, we instrument the marked edges like BL's algorithm. Furthermore, we instrument the call site by adding two parameters, r and $func$. Here $func$ is to return the function id of callee. This is necessary because if function pointers are used, the caller has no idea about who is the callee unless the



(a) Overlapping Graph for call site C_1 in f (b) BL graph for g

Figure 3. Path enumeration.

callee explicitly tells the caller. In the instrumented CFGs of Figure 4, we can see $func$ is assigned the id of function g at the beginning of g's execution. r is used to enumerate the paths in the callee, which are the subpaths of the interprocedural overlapping paths. ro is used to enumerate the subpath in the caller. The concatenated path $r\ !\ ro$ is the interprocedural overlapping path. We can see, at the exit edge of caller's OG, the counter corresponding to the path $< func, C_1, r, ro >$ is incremented by 1.

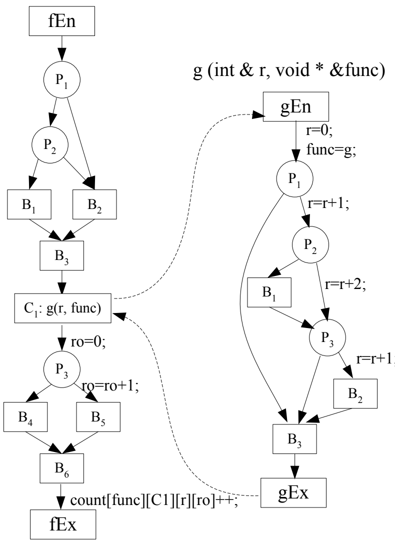


Figure 4. Instrumentation.

4 Experimental Results

We have implemented our algorithms using the Trimaran compiler infrastructure [19]. Based upon this implementation we carried out an experimental evaluation whose results are reported in this section. There were two objectives of the experimentation: to evaluate the improvement in precision of estimates of *interesting path profiles* computed from *overlapping path profiles* of varying degrees of allowable overlap; and to evaluate the *overhead* of collecting overlapping path profiles. It was also our goal to compare the precision and overhead associated with overlapping path profiles with that of BL path profiles. We also collected the whole program path profiles [16] to determine the precise frequency of any path for a program run.

4.1 Precision: OL vs. BL path profiles.

The detailed results of our experiments for nine benchmark programs are presented in Figures 5 and 6.

First we computed the real total flow of the interesting paths and then compared it with the total estimated flow (lower/upper bounds or definite/potential flows) for the same paths when estimates were derived from overlapping path profiles of varying degrees of overlap. As we can see from Figure 5, the precision improves as degree of overlap increases. The point corresponding to overlap of -1 represents estimates derived using BL paths. It is clear that while BL paths provide highly imprecise estimates, an overlap set at approximately one-third of maximum possible overlap gives us fairly good estimates. In Table 8 the estimates corresponding to this threshold are presented. While the average imprecision in estimated total flow of interesting paths derived from BL path frequencies ranges from -38% to +138%, the average imprecision in flow estimates derived from overlapping path frequencies range only from -4% to +8%.

Second we also looked at the number of interesting paths for which estimated frequency matched the actual frequency (i.e., upper and lower bounds derived were identical). This data is plotted in Figure 6. The interesting observation here is that in benchmarks where there are large number of interesting paths (e.g., `mcf` and `twolf`) and the maximum amount of overlap possible is very high, a small overlap is sufficient to get precise estimates for vast majority of the paths. This observation further shows that it is sufficient to use small overlaps to derive highly precise flow estimates.

4.2 Overhead: OL vs. BL path profiles.

We also present the overhead of collecting overlapping path profiles for varying degrees of overlap in Figures 7, 8 and 9. The first graph shows the overhead of collecting overlapping loop path profiles for varying degrees of overlap. The overhead corresponding to zero overlap represents the overhead of collecting BL path profiles in our implementation. The second graph presents the data for overhead associated with collecting overlapping interprocedural path

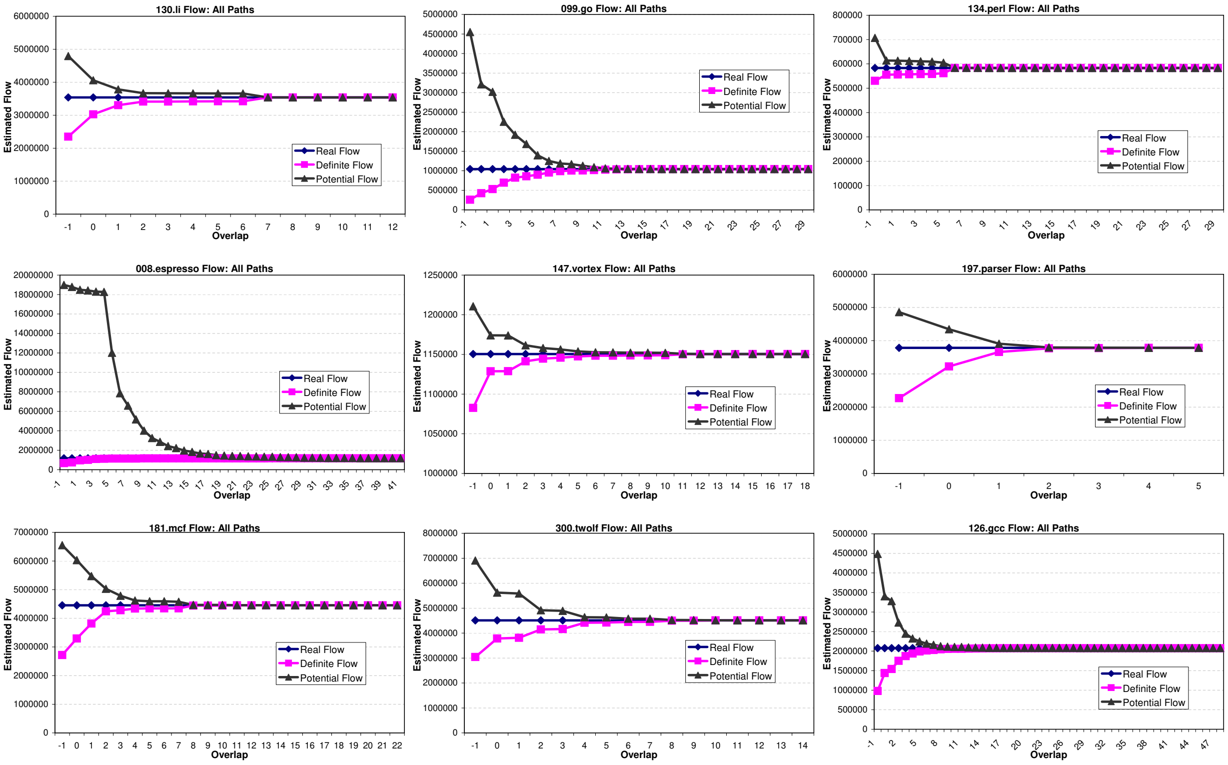


Figure 5. Total flow.

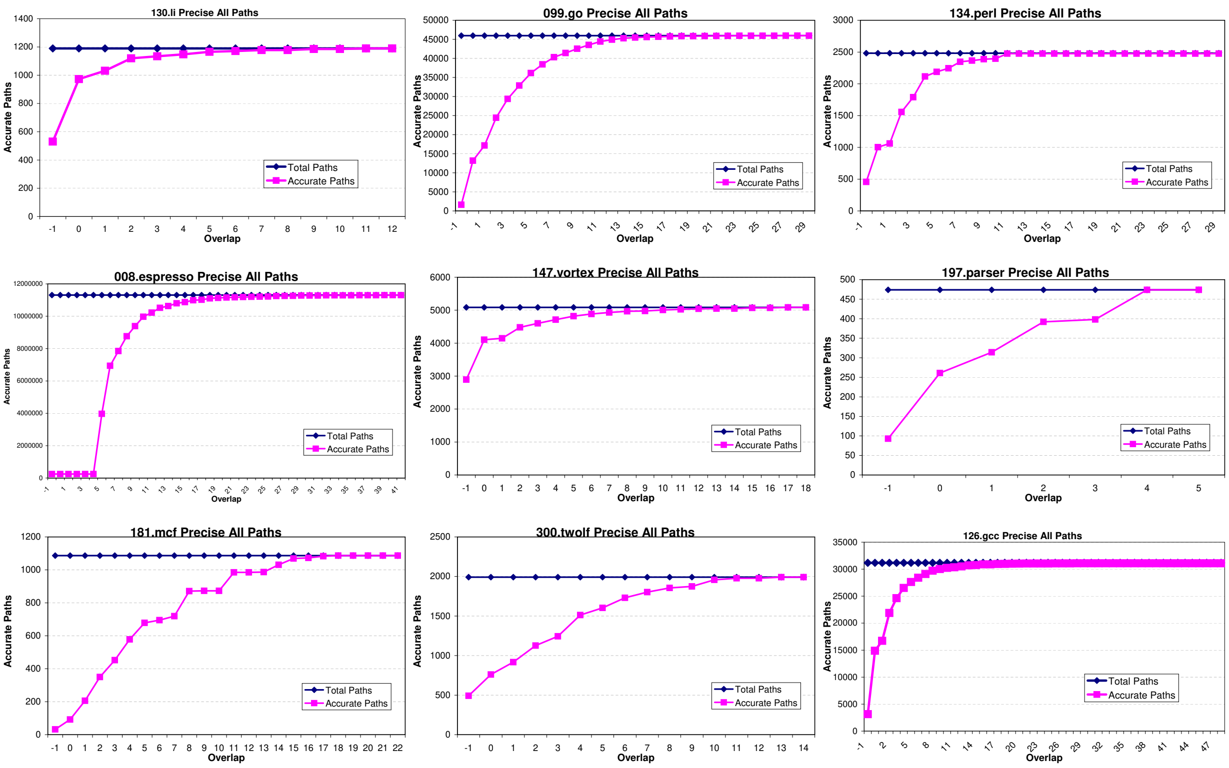


Figure 6. Precisely estimated paths.

Table 8. Definite and potential flow.

Benchmark	Real Flow	Using BL		Using OL-k			
		Definite Flow	Potential Flow	Definite Flow	Potential Flow	k Chosen	k Max
130.li	3539310	2350308 (-33.6 %)	4793897 (35.4 %)	3411257(-3.6 %)	3666315 (3.5 %)	2	12
099.go	1043000	260200 (-75 %)	4549045 (336.2 %)	1017335 (-2.4 %)	1089760 (4.4 %)	10	29
134.perl	583202	530643 (-9 %)	706814 (21.1 %)	583163 (0 %)	583289 (0 %)	10	29
008.espresso	1176329	677091 (-42 %)	18988987 (1514 %)	1173703 (-0.2 %)	1970368 (67 %)	14	45
147.vortex	1150460	1082634 (-5.8 %)	1210629 (5.2 %)	1148352 (-0.2 %)	1152686 (0.2 %)	6	18
197.parser	3783648	2268678 (-40 %)	4861451 (28.4 %)	3224556 (-14.8 %)	4344380 (14.8 %)	2	5
181.mcf	4454330	2718793 (-38.9 %)	6553449 (47.1 %)	4345860 (-2.4 %)	4577679 (2.7 %)	7	22
300.twolf	4509841	3041739 (-32.5 %)	6908699 (53.2 %)	4415177 (-2.1 %)	4636763 (2.8 %)	4	14
126.gcc	2076527	978781 (-52.8 %)	4487127 (116 %)	2068457 (-0.4 %)	2087944 (0.55 %)	14	48
Average	2479627	1545429 (-37.6 %)	5895566 (138 %)	2376428 (-4.1 %)	2678798 (8 %)	8	25

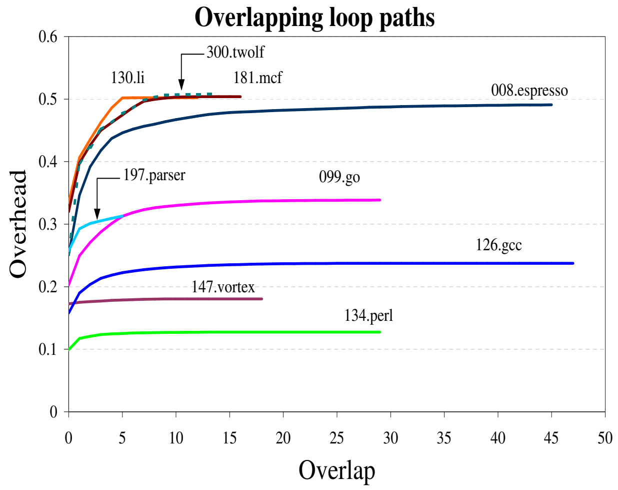


Figure 7. Overhead of profiling OL loop paths.

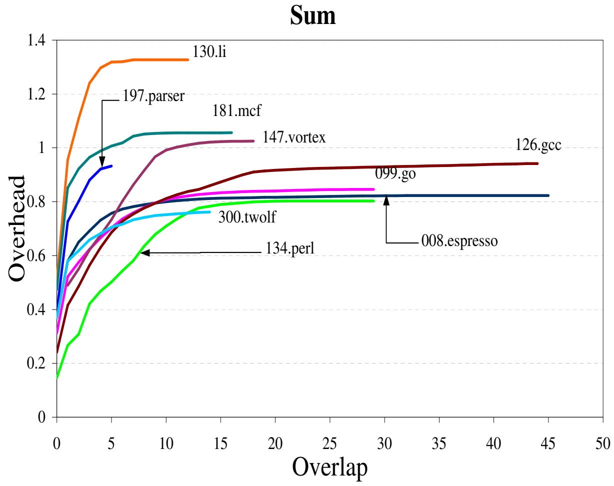


Figure 9. Overhead of profiling all OL paths.

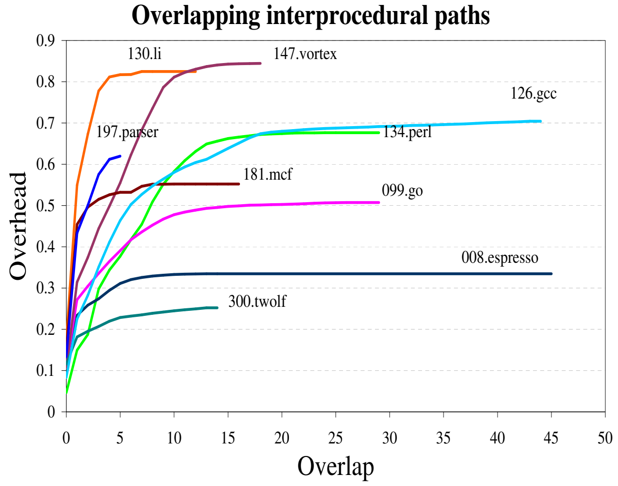


Figure 8. Overhead of profiling OL interprocedural paths.

profiles. The third graph shown the overhead of collecting profile data for all overlapping paths. The overhead of profiling overlapping interprocedural paths is higher than that of profiling overlapping loop paths.

Table 9 gives the overhead of collecting frequencies of BL paths and overlapping paths when the overlap is set at approximately one-third of the maximum possible overlap. The average overhead of collecting overlapping path profiles is 86.8%. On an average this overhead is 4.2 times that of profiling BL paths.

Recently techniques have been proposed that exploit edge profiles to selectively profile subset of program paths [2, 15]. These techniques can be used to reduce the overhead of our approach.

5 Conclusions

In this paper we identified a new class of paths: overlapping loop and interprocedural paths. Collecting profiles for such paths is useful because using them we can estimate the frequencies of interesting paths that cross loop and procedure boundaries. The interesting paths are relevant for

Table 9. Instrumentation overhead.

Benchmark	BL (%)	OL-k paths (%)			All
		Loop	Interproc.	All	BL
130.li	33.5	43.5	67.3	110.8	3.3
099.go	20.3	33.0	47.8	80.8	3.98
134.perl	9.9	12.7	58.2	70.9	7.16
008.espresso	25.1	47.7	33.5	81.2	3.23
147.vortex	17.3	17.9	62.3	80.2	4.64
197.parser	25.7	30.1	50.0	80.1	3.12
181.mcf	31.9	49.6	54.7	104.3	3.27
300.twolf	25.3	46.2	41.1	87.3	3.45
126.gcc	15.8	23.4	62.5	85.9	5.44
Average	22.7	33.8	53.0	86.8	4.2

many code optimizations and global instruction scheduling. The cost of collecting overlapping path profiles is reasonable and the frequency estimates for interesting path profiles derived from overlapping path profiles are highly precise. In contrast similar estimates derived using BL path profiles are highly imprecise.

References

[1] G. Ammons and J.R. Larus, "Improving Data Flow Analysis with Path Profiles," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 72-84, Montreal, Canada, 1998.

[2] T. Apiwattanapong and M.J. Harrold, "Selective Path Profiling," *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (PASTE), pages 35-42, Charleston, South Carolina, 2002.

[3] T. Ball, and J. Larus, "Efficient Path Profiling," *IEEE/ACM International Symposium on Microarchitecture* (MICRO), pages 46-57, Paris, France, December 1996.

[4] T. Ball, P. Mataga, and M. Sagiv, "Edge Profiling versus Path Profiling: The Showdown," *ACM Symposium on Principles of Programming Languages* (POPL), pages 134-148, San Diego, CA, January 1998.

[5] R. Bodik, R. Gupta, and M.L. Soffa, "Interprocedural Conditional Branch Elimination," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 146-158, Las Vegas, Nevada, June 1997.

[6] R. Bodik and R. Gupta, "Partial Dead Code Elimination using Slicing Transformations," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 159-170, Las Vegas, Nevada, June 1997.

[7] R. Bodik, R. Gupta and M.L. Soffa, "Complete Removal of Redundant Expressions," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 1-14, Montreal, Canada, June 1998.

[8] R. Bodik, R. Gupta, and M.L. Soffa, "Load-Reuse Analysis: Design and Evaluation," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 64-76, Atlanta, Georgia, May 1999.

[9] R. Bodik, R. Gupta, and V. Sarkar, "ABCD: Eliminating Array Bounds Checks on Demand," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 321-333, Vancouver B.C., Canada, June 2000.

[10] E. Duesterwald and V. Bala, "Software Profiling for Hot Path Prediction: Less is More," *ACM 9th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), pages 202-211, Nov. 2000.

[11] J.A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," *IEEE Transactions on Computers*, C-30:478-490, 1981.

[12] R. Gupta, D. Berson, and J.Z. Fang, "Resource-Sensitive Profile-Directed Data Flow Analysis for Code Optimization," *IEEE/ACM International Symposium on Microarchitecture* (MICRO), pages 558-568, December 1997

[13] R. Gupta, D. Berson, and J.Z. Fang, "Path Profile Guided Partial Dead Code Elimination Using Predication," *International Conference on Parallel Architectures and Compilation Techniques* (PACT), pages 102-115, San Francisco, 1997.

[14] R. Gupta, D. Berson, and J.Z. Fang, "Path Profile Guided Partial Redundancy Elimination Using Speculation," *IEEE International Conference on Computer Languages* (ICCL), pages 230-239, Chicago, Illinois, May 1998.

[15] R. Joshi, M. Bond, and C. Zilles, "Targeted Path Profiling: Lower Overhead Path Profiling for Staged Dynamic Optimization Systems," *IEEE-ACM International Symposium on Code Generation and Optimization* (CGO), March 2004.

[16] J.R. Larus, "Whole Program Paths," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 259-269, Atlanta, GA, may 1999.

[17] D. Melski and T. Reps, "Interprocedural Path Profiling," *8th International Conference on Compiler Construction* (CC), pages 47-62, LNCS 1575, Springer-Verlag, 1999.

[18] G. Ramalingam, "Data Flow Frequency Analysis," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 267-277, Philadelphia, 1996.

[19] *The Trimaran Compiler Research Infrastructure*. Tutorial Notes, November 1997.

[20] C. Young and M.D. Smith, "Better Global Scheduling Using Path Profiles," *IEEE/ACM International Symposium on Microarchitecture* (MICRO), pages 115-123, 1998.

[21] C. Young and M.D. Smith, "Improving the Accuracy of Static Branch Prediction Using Branch Prediction," *ACM 6th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), pages 232-241, October 1994.

[22] Y. Zhang and R. Gupta, "Timestamped Whole Program Path Representation and its Applications," *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), pages 180-190, Snowbird, Utah, June 2001.