

Complementing Incomplete Edge Profile by applying
Minimum Cost Circulation Algorithms

Roy Levin

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa
Faculty of Social Science
Department of Computer Science

August, 2007

Complementing Incomplete Edge Profile by applying
Minimum Cost Circulation Algorithms

By: Roy Levin
Supervised By: Prof. Ilan Newman and Dr. Gad Haber

University of Haifa
Faculty of Social Science
Department of Computer Science
August, 2007

Approved by: _____ Date: _____
(supervisor)

Approved by: _____ Date: _____
(supervisor)

Approved by: _____ Date: _____
(Chairman of M.A Committee)

Acknowledgment

This thesis is the outcome of my academic studies at the Haifa University and my work as a researcher at IBM's Haifa Research Labs. I would like to start off thanking my supervisor, Prof. Ilan Newman, for his brilliant guidance. His outstanding analytic abilities, his early observations and directions have been a tremendous help to me throughout my work on this thesis. I would also like to thank my IBM supervisor, Dr. Gadi Haber, which was also my manager at IBM for his excellent support, whom without his amazing "out of the box" thinking in suggesting solutions to problems we have encountered; this thesis may have not been possible. I like to thank my managers at IBM for allowing me to continue my full time position at IBM while studying for an M.Sc and also for allowing me to use my research fields at work to benefit my thesis. I wish to thank Daphi Stern, the Dean's administrative assistant, and her staff, for the extraordinary responsiveness and quick help in all the administrative issues that arose during my study period.

Last, but by no means least, I would like to thank my wonderful wife, Iris, for her confidence in me and her support despite the fact that so much of my time and efforts have been devoted to my work and studies.

This has all been vital to the successful completion of my thesis.

My apologies go to anyone relevant whom I failed to mention within this context.

Table of Contents

	Page
ABSTRACT.....	V
List of Tables	VII
List of Figures.....	VIII
1 Introduction.....	1
2 Related work for profiling techniques	6
3 Formulating the problem	8
4 A polynomial algorithm for finding an optimal feasible fixup vector.....	10
4.1 Vertex transformation.....	10
4.2 Fixup graph construction scheme.....	13
4.3 Mathematical proof.....	16
4.3.1 Flow conservation is preserved.....	16
4.3.2 A solution to the circulation problem always exists.....	18
4.3.3 The weighted deltas are minimal.....	18
4.4 Complexity of the algorithm	20
5 Estimating vertex and edge frequencies	21
6 Experimental results.....	23
6.1 Filling edge profile from vertex profile	24
6.2 Approximating dynamic control flow for external sampling	25
7 Future directions.....	29
8 Summary.....	31
Bibliography	32
תקציר.....	35

Complementing Incomplete Edge Profile by applying Minimum Cost Circulation Algorithms

Roy Levin

ABSTRACT

Edge profiling is a very common means for providing feedback on program behavior that can be used statically or dynamically by a compiler or a post-link tool to produce highly optimized binaries. The optimizer uses this information to benefit its profile driven optimizations by concentrating its efforts on the frequently executed code. However collecting full edge profile carries a very significant runtime overhead which makes it inappropriate for dynamic compilation. On the other hand collecting full edge profile offline requires the availability of instrumentation tools, and furthermore complicates and slows down the compilation/build process. In addition to this classic tradeoff between long and complex compilations/builds to runtime performance, in realtime applications there is an additional issue to be concerned with. By using heavy profile gathering techniques, such as intrusive instrumentation code, the system may be prevented from meeting runtime deadlines and thus alter its behavior.

In this paper we propose a novel technique that provides a highly accurate approximation to full edge profiling. This allows non-intrusive, low overhead profiling techniques, such as low-rate instruction complete sampling, to be used to collect partial and inaccurate profile information. Using this information, as is, produces suboptimal results, therefore we use our suggested approach on the collected, missing and inaccurate, profile information in order to approximate the full edge profile information. We translate the problem of filling the missing and inaccurate information to the **Minimum Cost Circulation Problem** and propose a solution that is based on the flow conservation rules. The assumption is that by creating a legal network flow, while

minimizing some criteria for the amount of change done to the given flow, we will provide a better estimate for the actual flow. To verify this assumption empirically, we gathered profiling using a low overhead sampling technique which slowed down the applications by a mere 2%-3% during the training set, and we then fixed this gathered profile using our scheme and fed the fixed profile to a postlink optimizer called FDPR-Pro to produce an optimized binary. When comparing this binary, which has been optimized using approximated profile, to a binary optimized by FDPR-Pro using a full and accurate profile we measured a performance improvement over the former original binary that was only 0.6% less than that which was produced using the full and accurate profile. This is a new approach to the approximation problem of missing or inaccurate profiling problem which provides a generic solution that does not require specific knowledge about the running architecture.

List of Tables

	Page
Table 1 - Degree of overlap comparison.....	26
Table 2 - Runtime comparison.....	27

List of Figures

	Page
Figure 1 - Partial vertex profile.....	3
Figure 2 - The complete vertex profile	3
Figure 3 - Possible edge estimation 1	4
Figure 4 - Possible edge estimation 2	4
Figure 5 - Vertex transformation	11
Figure 6 - Building the Fixup graph	15
Figure 7 - Fixed vs. unfixed sampling	28
Figure 8 - Using multiple epilogue basic blocks to determine flow	30

1 Introduction

Control flow profiling is the determination of the number of time each edge/vertex is traversed in the flow diagram of a program, when running a 'typical' input. Such profile can be obtained by adding instrumentation code or by using external sampling, and are extremely useful as they provide empirical information about the application such as determining performance critical areas in the code and deducing probabilities of conditional branches to be taken. Indeed, such methods have been used since the 70s. Profile driven optimizations are supported today in most modern compilers and post-link optimizers [6 – 13]. Profile-directed compilation uses information gathered in several ways: Run-time profiling which is mainly used today by dynamic optimizers such as the Java Just In Time (JIT) compiler, in which profile is collected at run-time. The problem with this approach is that it requires additional system resources at runtime, which may be undesirable in high performance low resource applications, such as embedded applications, especially if they carry real time constraints. Another method for profile-driven optimization uses pre-selected representative workload which trains the program on typical workloads and then uses this data to produce a highly optimized version. Finally, static profiling can be used by compilers as a method for predicting program's control flow, but such methods are not as effective as the previously mentioned options [4].

Collecting full edge profile requires creating instrumented code with edge counters that will increment upon execution and persist this data, typically by mapping it to some file. Thomas Ball and James R. Larus suggest algorithms for optimally inserting monitoring code to profile and trace programs and they report a slowdown of between 9% to 105% when comparing the original binary to the instrumented one [1]. Due to this slowdown, along with the extra build step which is required for instrumentation which also increases the complexity of the compilation/build process, many consider alternative approaches such as low overhead profiling [2]. In real time applications another problem arises, as intrusive instrumentation code may alter real-time application behavior as a result of deadlines in the real time program that may be missed [21]. Therefore collecting profile for real-time applications calls for less intrusive profile

collection techniques, such as low rate sampling, selective instrumentation or running the instrumented binary in a machine which is significantly stronger than target machine. However, using sampling techniques or selective instrumentation will produce both inaccurate and lacking profile information which may result in sub-optimal performance of an application optimized according to this profile.

In this work, we present a new edge profile estimation algorithm based on given partial and possibly inaccurate vertex counts with costs attached to the edges and vertices. Our edge profile estimation is translated to the Minimum Cost Circulation Problem [5], which infuses a legal circulation in a flow network while keeping the weighted flow-costs on the edges at a minimum. The assumption is that by creating a legal network flow, while minimizing some criteria for the amount of global change done to the given flow, it will provide a better estimate for the actual flow. We provide empirical proof for this assumption in the experimental results which appear in section 6. Let us consider the problem of filling in missing profile data. As an example, consider the control flow graph shown in figure 1 which includes partial vertex profiling collected by sampling the Instruction Complete hardware event (the numbers represent the execution counts for each vertex). From the given example, it is clear that the complete vertex counters which a zero value should be fixed as shown in figure 2.

Figure 1 - Partial vertex profile

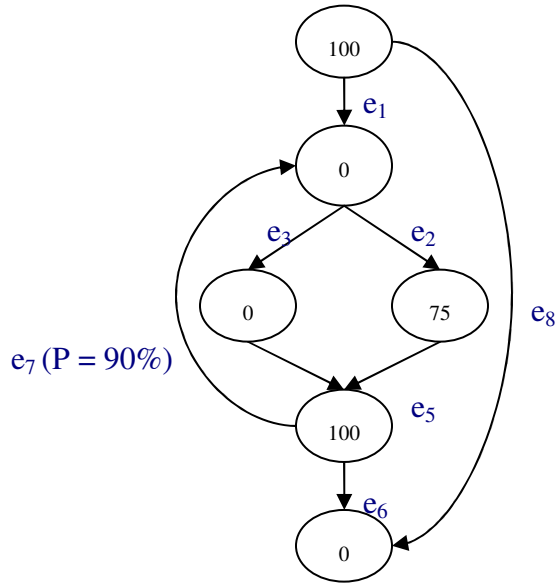
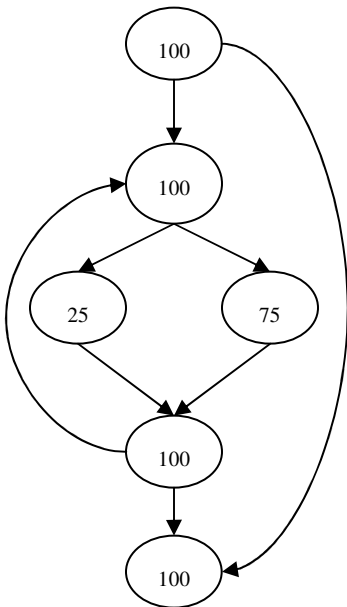


Figure 2 - The complete vertex profile



However, determining the edge profile from the vertex counters alone is not always possible. In this example, the control flow graph shown in figure 2 has two possible edge profile estimates as shown in figures 3 and 4.

Figure 3 - Possible edge estimation 1

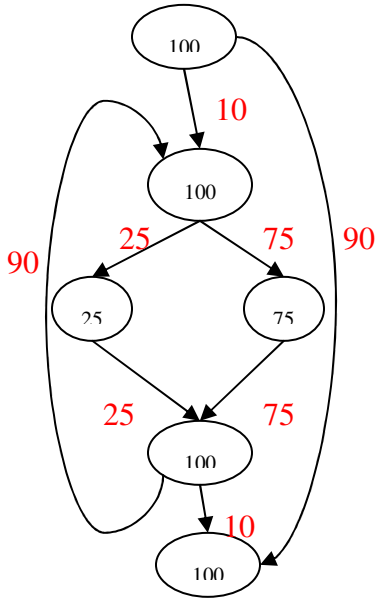
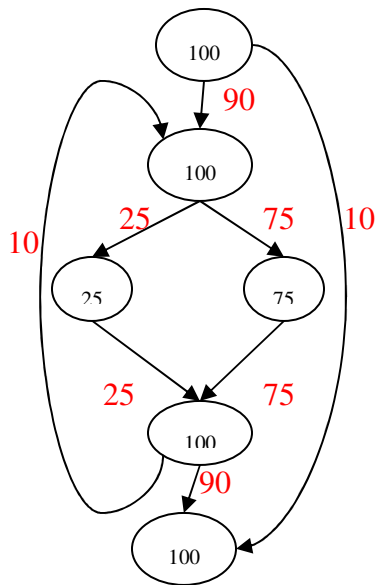


Figure 4 - Possible edge estimation 2



Both optional estimates adhere to the flow conservation constraints but the differences between the edge weights may be very significant. The freedom to choose weight for edges e_1 , e_6 , e_7 , e_8 may yield very different results. As one would expect, Youfeng Wu and James R. Larus [4], show that loop back edges have a very high probability to be taken. Keeping this in mind, we can determine that optional estimation 1 (from figure 3), that infuses much more flow on edge e_7 has a much higher chance of achieving a better approximation of the actual flow. Any attempt to infuse flow on the graph's edges will have to be made aware of the probabilities on the edges to be taken. However, examining the probabilities locally is not optimal since if one decides to infuse a count of 50 on e_1 and on e_8 (as the probabilities on them may be equal) then that will determine a count of 50 on e_6 and on e_7 as well. This is undesirable, since we would want to infuse most of the flow through e_7 which is a back edge with high probability to be taken. Therefore, in order for a complementing algorithm to be successful it should have some global view of the edge probabilities. Indeed, in this example, the high probability of 90% for e_7 , shown in figure 1, will guide our proposed algorithm's global view to prefer the 2nd estimated profile edge.

We know how to find optimal solution with respect to several cost functions. However, in order to test the applicability of our method for real life applications we, heuristically, tried several different cost functions that were assigned to a profile gathered by low-rate sampling profiling. This will be thoroughly explained in section 3, where we formulate the problem.

The paper is arranged in the following manner: section 2 describes additional related work on profile gathering techniques. Section 3 formulates the problem of complementing the missing profiling information to the Minimum Circulation problem. Section 4 and 5 describes the proposed algorithm for fixing the missing graph. The complete proof of the algorithm correctness is located in the appendix. Section 6 provides experimental results which were obtained by implementing the algorithm into FDPR post-link optimizer and running it on AIX POWER 5 SPEC INT 2000 suite. Finally, future directions are discussed in section 7 followed by a summary section.

2 Related work for profiling techniques

Many papers have been written on techniques for collecting profile information. The issue is of major importance since the collected profile is very useful at guiding optimizing both static and dynamic compilers and post-link optimizers while the collection of the profile carries a significant overhead. The goal is therefore to collect accurate profile while keeping the runtime overhead of the profile collection to minimum.

J. Anderson et al present a method for collecting profile samples at a high rate and with low overhead [2]. To describe performance at the instruction level, they address two issues: how long each instruction stalls and the reasons for each stall. To determine stall latencies, an average CPI is computed for each instruction, using estimated execution frequencies. Accurate frequency estimates are recovered from profile data by a set of heuristics that use a detailed model of the processor pipeline and the constraints imposed by program control flow graphs to correlate sample counts for different instructions. The processor pipeline model explains static stalls; dynamic stalls are explained using a “guilty until proven innocent” approach that reports each possible cause not eliminated through careful analysis. In section 6.1.4 they refer to their Local Propagation algorithm and add a note that they are experimenting with a global constraint solver.

Additional papers suggest ways to collect profile information while reducing the overhead of the profile collection and attempting to keep the profile accuracy high at the same time. Matthew Arnold and Barbara G. Ryder [15] propose a framework that performs code duplication and uses compiler inserted counter-based sampling to switch between instrumented and non-instrumented code in a controlled fine-grained manner. Thomas Ball and James R. Larus suggest algorithms for optimally inserting monitoring code to profile and trace programs [1]. The algorithms optimize the placement of counting/tracing code with respect to the expected or measured frequency of each block or edge in a program's control-flow graph. Their idea is to find the lowest cost set of instrumentation points, when given a flow network and costs.

The technique proposed in our paper attempts to suggest a general, and global algorithm that can

address filling in missing or inaccurate profile information, such as that which occurs when performing sampling but not limited to. For filling in, and fixing, the profile for the sampling case, we use an *instruction complete* event counter instead of sampling the program counter and then estimating the basic block frequencies using machine dependant heuristics.

3 Formulating the problem

Our method is based on the assumption that by creating a legal network flow, while minimizing some criteria for the amount of weighted change done to the given flow, we can provide a better estimate to the actual flow that occurred. So the problem can be generalized and viewed as follows:

Given a directed graph $G = (V, E)$ with (integer) measured flow, $w(v)$, $w(e)$ for each vertex v and each edge e , the measured flows are assumed to be derived from inaccurate flow measurements in a given flow network. The inaccuracies may manifest themselves as discrepancies in the given flow network which break the flow conservation rule stating that for every $v \in V \setminus (S \cup T)$ the following should hold:

$$(1) \quad \sum_{e_{in} \in in(v)} w(e_{in}) = \sum_{e_{out} \in out(v)} w(e_{out}) = w(v)$$

S, T here are arbitrary given (possibly empty) subsets of V (of sources and sinks respectively) for which flow conservation is not demanded. In the application, these sets correspond to entry/exit points in the control flow graph. These can be the prologue/epilogue basic blocks if our control flow graph represents a single procedure or entry points and exit points of a program if our control flow graph refers to the entire program.

We refer to condition (1) as the *generalized flow conservation rule*.

The idea is that by fixing the flow to adhere to the *generalized flow conservation rule* while limiting the amount of weighted change to a minimum we will achieve a near approximation to the actual flow. Thus, a feasible solution to the problem is a *fixup vector*, $(\Delta(o) : o \in V \cup E)$, namely a vector of changes, one per each edge and vertex, for which the corrected flow $w^*(o) = w(o) + \Delta(o)$ yields a legal flow that satisfies the *generalized flow conservation* (1). We rank different feasible solutions by a cost function associated with each fixup vector and that is formally part of the input (in application this would be a heuristically chosen function as will be explained in the next section). Indeed our experimental results show that such optimal (or high rank) feasible solutions are a good approximation to the profiling problem.

Our algorithms can find optimal solution to the above formal problem for a wide variety of costs. Linear cost functions are theoretically easy to deal with, while not very practical, as it would imply that for each edge / vertex, either increasing the flow by an infinite amount, or decreasing it by an infinite amount is beneficial. A class of reasonable cost functions would be of the form:

$$\text{cost}(\Delta) = \sum_{o \in V \cup E} \text{cost}(\Delta(o)) = \sum_{o \in V \cup E} cp(o) \cdot |\Delta(o)|.$$

Where $cp(o)$ is a non negative vector of coefficients. Such functions are monotone with the absolute amount of change for each edge/ vertex and are referred as *weighted l_1 costs*. Such functions give the ability, by choosing the right weights, to prefer changes to some edges than to others (e.g due to some a-priory knowledge of the reliability of the measurements at different sites). We can, however, do a bit more. Weighted l_1 costs do not distinguish between increasing and decreasing the flow at a site. It might be important for some edges to charge more for increasing the flow than to decreasing it (again, due to some prior knowledge on the flow). Namely, we can define

$$\text{cost}(\Delta) = \sum_{o \in V \cup E} \text{cost}(\Delta(o)) = \sum_{o \in V \cup E} cp(o, \Delta) \cdot |\Delta(o)|$$

Where the coefficient $cp(o, \Delta) = k^+(o)$ if $\Delta > 0$ and $cp(o, \Delta) = k^-(o)$ if $\Delta < 0$. These coefficient are called the *confidence constants*. Clearly such cost function generalizes weighted l_1 costs and will be referred to as *generalized l_1* .

In the following we solve the problem optimally for generalized l_1 . Indeed we show in the experimental results that such functions can be used to obtain good practical results. We end this discussion with the note that other interesting cost functions, or additional constraints can be considered. For example, one could use the extra conditions on feasible solution stating that the amount of change is at most 10% of the original (or for that matter, a fractional quantity that is given for each edge/ vertex. One could also add constraints limiting form above or below the total amount of change. In addition to this, one could minimize the sum of square changes (the R.M.S norm), or even a weighted version of it. We can solve all such variants optimally in polynomial time however this is beyond the scope of this work.

4 A polynomial algorithm for finding an optimal feasible fixup vector

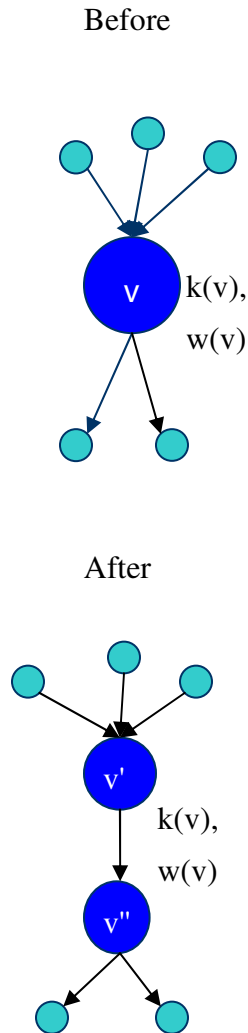
To find the optimal generalized l_1 solution we use a reduction to minimum-cost circulation algorithm [5] which we apply on a transformed graph we call the fixup graph (see section 4.1). A circulation is a generalization of flow in a network in the sense that there is no source and sink, and thus flow conservation must be maintained for each and every vertex in the graph. A minimum-cost circulation is a circulation that satisfies the flow conservation for every vertex in the graph while keeping the weighted cost of the flow at a minimum (here the cost is a simple linear function, that is, for a circulation $c=(c(e))$ for every edge e , its cost is: $cost(c) = \sum_{e \in E} k(e) \cdot c(e)$, where $k(e)$ is a non-negative weight for each edge e).

The reduction is composed of two steps. In the first step, denoted as *vertex transformation*, we remain in the domain of the optimum flow fixup problem, but get rid of vertex flows and their corresponding costs, and in the second step we build the fixup graph.

4.1 Vertex transformation

By peeling off the weights from the vertices, we can reduce our problem to a standard circulation problem with weighted edges and without weights on vertices. Figure 5 illustrates a given vertex along with its collocating edges before and after the suggested transformation. The vertex v is halved into two vertices v' and v'' , and the incoming and outgoing edges are distributed respectively as shown in figure 5. v' and v'' are then connected by an edge e^v , outgoing from v' and incoming to v'' , in addition, we define the weight and cost functions for this new edge as $w(e^v) \equiv w(v)$ and $k^\pm(e^v) \equiv k^\pm(v)$.

Figure 5 - Vertex transformation



Clearly, the vertex transformation does not change the cost, thus we will assume in what follows that we only have edge flows and costs. In addition, we may assume that the set of sources is a singleton, that is $S=\{s\}$, as otherwise we just add a new source s' and connect it to every original source in S with an edge of cost 0. Thus original sources will conserve flow automatically with no additional costs while moving the surplus flow to the single source s' . Similarly, we may assume that there is a single sink t' by an analogue argument. Finally, we may insert a new directed edge of cost 0 from the single sink t to the single source s . This will turn

any feasible flow to a circulation of the same cost. Thus we may assume that in fact $S=T=\emptyset$.

4.2 Fixup graph construction scheme

Given a graph $G=(V,E)$ and its measured flow $w(v)$ and $w(e)$ for the vertices and edges respectively, as input for the optimal flow fixup, we wish to create a new graph $G'=(V',E')$ with given minimal and maximal capacity constraints (b, c) for each edge, herein the *fixup graph*.

This transformation is formally defined here:

Input:

- $G_t=(V_t,E_t)$ denotes the original graph after applying the vertex transformation.
- $w(e)$ denotes the initial flow estimation for every edge $e \in E_t$ (this is thoroughly explained in section 5 in "**setting the constants**").
- $k^\pm(e)$ denotes the negative/positive *confidence constants* on any $e \in E_t$ (see section 3).
- Let $D(v) \equiv \sum_{e_l \in out(v)} (w(e_l)) - \sum_{e_k \in in(v)} (w(e_k))$ for every $v \in V_t$.

Output:

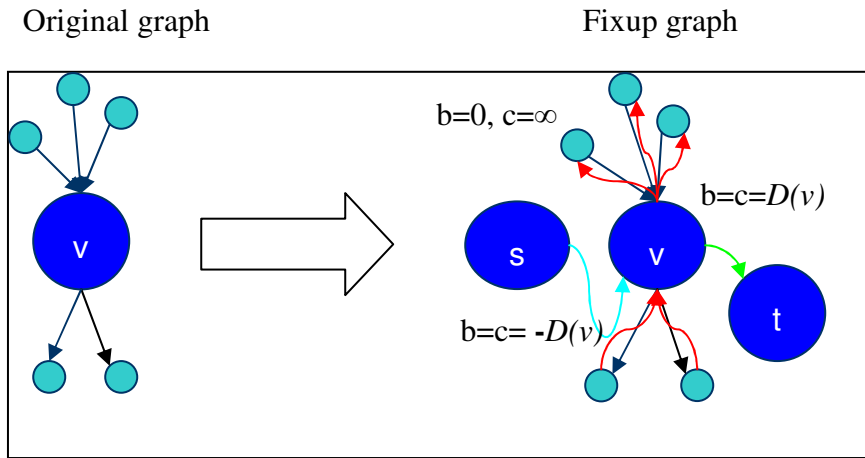
- $G'=(V',E')$ the *fixup graph*
- $b(e'),c(e')$ minimum/maximum capacities for flow on every edge $e' \in E'$
- $k(e')$ *positive confidence constant* for any $e' \in E'$ (note that infusing negative flow is not possible so here we do not need a *negative confidence constants*)

The output graph for the circulation problem is defined as follows:

1. $s' \leftarrow \text{new Vertex}, t' \leftarrow \text{new Vertex}$
2. $b(\langle t', s' \rangle) \leftarrow 0, c(\langle t', s' \rangle) \leftarrow \infty$
3. $cp(\langle t', s' \rangle) \leftarrow 0$
4. $E_r \leftarrow \phi, L \leftarrow \phi$
5. foreach $e \in E_t$ do:
 - $b(e) \leftarrow 0, c(e) \leftarrow \infty, k'(e) \leftarrow k^+(e)$
6. foreach $e = \langle v, u \rangle \in E_t$ such that $\langle u, v \rangle \notin E_t$ do:
 - $E_r \leftarrow E_r \cup \langle u, v \rangle$
 - $k'(\langle u, v \rangle) \leftarrow k^-(\langle v, u \rangle),$
 - $b(\langle u, v \rangle) \leftarrow 0,$
 - $c(\langle u, v \rangle) \leftarrow w(e)$
7. foreach $v \in V_t$ do:
 - $L \leftarrow L \cup \{v, t'\},$
 - if $D(v) > 0$ then $b(\langle v, t' \rangle) \leftarrow D(v),$
 - $c(\langle v, t' \rangle) \leftarrow D(v)$
 - $L \leftarrow L \cup \{s', v\},$
 - if $D(v) < 0$ then $b(\langle s', v \rangle) \leftarrow -D(v),$
 - $c(\langle s', v \rangle) \leftarrow -D(v)$
8. $V' \leftarrow V \cup \{s', t'\}$
9. $E' \leftarrow E \cup E_r \cup L \cup \langle t', s' \rangle$

Figure 6 illustrates a typical vertex in the *fixup graph* $G'=(V',E')$.

Figure 6 - Building the Fixup graph



Note that in the final phase of the construction we add an edge from t to s to create a circulation problem rather than a flow problem.

Any solution to the circulation problem is a flow function for each edge in E' . For each edge in the original graph, $e = \langle v, u \rangle \in E$ we then calculate the fixup vector, $\Delta(e)$ as follows:

$$\Delta(e) = \begin{cases} f(v, u) & f(v, u) \geq 0 \\ -f(u, v) & f(u, v) < 0 \end{cases}. \text{ By mapping back the edges which were derived from the}$$

vertices when we applied the vertex transformation (the vertex splitting at the beginning of this section) we can determine the values of $\Delta(v)$ for each v in V as well.

In the following sub sections we formally prove that the Δ function we found is a feasible fixup vector. In addition, if we assume that the circulation flow is optimal for the circulation problem we can show that the fixup is optimal for the flow fixup with the generalized l_1 .

4.3 Mathematical proof

Claim: applying the fixup vector $\Delta(o)$, where $o \in V \cup E$, to the initial flow estimation function $w(o)$, yields a flow that satisfies the flow conservation constraints and the fixup vector is minimal as defined in the "minimizing the weighted deltas" problem.

Note: the proof assumes that the vertex transformation defined in section 4, has already been applied on the original graph (which requires the fix).

4.3.1 Flow conservation is preserved

Given a solution for the circulation problem on the fixup graph we need to show that the transformation we presented from the flow function f on the fixup graph to the Δ function will yield a feasible flow in the original graph as follows:

$$\forall v \in V \setminus (S \cup T) \Rightarrow \sum_{e_k \in \text{in}(v)} (w(e_k) + \Delta(e_k)) = \sum_{e_l \in \text{out}(v)} (w(e_l) + \Delta(e_l))$$

Since the minimum cost circulation algorithm finds a legal flow in the graph we may deduce the following:

$$\forall v \notin S \cup T \Rightarrow \sum_{e' \in \text{in}(v) \cap E} f(e') + \sum_{e_r' \in \text{in}(v) \cap E_r} f(e_r') + \sum_{l \in \text{in}(v) \cap L} f(l) = \sum_{e' \in \text{out}(v) \cap E} f(e') + \sum_{e_r' \in \text{out}(v) \cap E_r} f(e_r') + \sum_{l \in \text{out}(v) \cap L} f(l)$$

$$\forall v \notin S \cup T \Rightarrow \left(\sum_{e' \in \text{in}(v) \cap E} f(e') + \sum_{e_r' \in \text{in}(v) \cap E_r} f(e_r') \right) - \left(\sum_{e' \in \text{out}(v) \cap E} f(e') + \sum_{e_r' \in \text{out}(v) \cap E_r} f(e_r') \right) = \sum_{l \in \text{out}(v) \cap L} f(l) - \sum_{l \in \text{in}(v) \cap L} f(l)$$

From examining the definition of the fixup graph, we know that for every edge $e = \langle v, u \rangle$ there is a corresponding edge $e = \langle u, v \rangle$. Therefore it follows that for every incoming edge there is a corresponding outgoing edge in the fixup graph. We may therefore write the following:

$\forall v \notin S \cup T \Rightarrow$

$$\sum_{\langle u,v \rangle, \langle v,u \rangle \in E/L} (f(u,v) - f(v,u)) = \sum_{l \in out(v') \cap L} f(l) - \sum_{l \in in(u') \cap L} f(l)$$

Since the flow on each edge must be within the boundaries set by the minimal and maximal capacity function for that edge we can conclude that $\forall l \in L \Rightarrow f(l) = b(l) = c(l) = \pm D(v)$ and therefore we can write:

$$\begin{aligned} &\forall v \notin S \cup T \Rightarrow \\ (*) \sum_{v \in V} (f(u,v) - f(v,u)) &= \sum_{l \in out(v') \cap L} f(l) - \sum_{l \in in(u') \cap L} f(l) = \pm D(v) \end{aligned}$$

Without loss of generality, let us assume that $D(v) > 0$ for the corresponding vertex $v \in V$ in G .

Recall that, by definition, $D(v) = \sum_{e \in out(v)} w(e) - \sum_{e \in in(v)} w(e)$, thus plugging this into the right hand

side of (*) and rearranging we get the following:

$$\sum_{(u,v) \in in(v)} f(u,v) + w(u,v) = \sum_{(v,u) \in out(v)} f(v,u) + w(v,u) \text{ and thus setting } w^*(e) = w(e) + f(e) \text{ satisfies flow}$$

conservation.

4.3.2 *A solution to the circulation problem always exists*

Given a fixup vector Δ , which is a solution to the flow-fix problem for the original graph, we can use a reverse mapping to determine the flow function f' for the fixup graph which yields a feasible circulation in the fixup graph. Hence, since there is always a fix to the flow-fix problem, e.g making all flows zero, there must also be a solution to the circulation problem on the fixup graph.

4.3.3 *The weighted deltas are minimal*

Next we need to show that the term $\sum_{o \in V \cup E} k(o) \cdot |\Delta(o)|$ is minimal.

By applying the minimal cost circulation algorithm on the fixup graph $G' = (V', E'); (b, c)$ we ensure that $\sum_{e \in E'} cp(e) \cdot f(e)$ is minimal.

Hence we may write the following:

$$\begin{aligned} \sum_{e \in E'} cp(e) \cdot f(e) &= \\ \sum_{e \in E} cp(e) \cdot f(e) + \sum_{e \in E_r} cp(e) \cdot f(e) + \sum_{e \in L} cp(e) \cdot f(e) &= \\ \sum_{e \in E_v} cp(e) \cdot f(e) + \sum_{e \in E_v^r} cp(e) \cdot f(e) + \sum_{e \in E} cp(e) \cdot f(e) + \sum_{e \in E_r} cp(e) \cdot f(e) + \sum_{e \in L} cp(e) \cdot f(e) \end{aligned}$$

since $cp(e) = 0, \forall e \in L$, it follows that:

$$\begin{aligned} \sum_{e \in E'} cp(e) \cdot f(e) &= \\ \sum_{e \in E_v} cp(e) \cdot f(e) + \sum_{e \in E_v^r} cp(e) \cdot f(e) + \sum_{e \in E} cp(e) \cdot f(e) + \sum_{e \in E_r} cp(e) \cdot f(e) &= \\ \sum_{o \in V \cup E} cp(o) \cdot |\Delta(o)| \end{aligned}$$

It can be trivially shown that $\forall \langle u, v \rangle = e \in E$ either $f(\langle u, v \rangle) = 0$ or $f(\langle v, u \rangle) = 0$, since otherwise it would be possible to reduce the total cost which contradicts the fact that f is minimal (this is assuming all the c_p are positive which is the case here). Therefore it follows that the cost in G' is the same as that in G which means they are both minimal.

4.4 Complexity of the algorithm

Goldberg & Tarjan [5], present an algorithm for the circulation problem. Theoretically the worst running time of this algorithm is $O(|V|^2 \cdot |E|^2 \cdot \min(\log(|V| \cdot C), |E| \cdot \log(|V|)))$ where C is the maximum absolute value of an arc cost. Despite this frightening worst case complexity we found that in practice the algorithm performed very well on the benchmarks from SPECint which we used for our analysis, and the algorithm's runtime was not an issue worth addressing when we applied it in procedure granularity (thus applying it on control flow graphs derived from procedures). Even as some of the control flow graphs which correspond to procedures from SPECint benchmarks contained thousands of vertices and edges.

5 Estimating vertex and edge frequencies

After gathering many experimental results and studying several cost coefficient functions we choose to define the cost coefficient function for the vertices and edges as follows:

$$cp(o) = \frac{k'(\Delta(o))}{\ln(w(o) + 2)},$$

$$k'(\Delta(o)) = \begin{cases} \Delta(o) \geq 0 & k_o^+ \\ \Delta(o) < 0 & k_o^- \end{cases}$$

Let us examine the terms in $cp(o)$:

The confidence constants k_o^+ / k_o^- , represent the confidence we have in the measurement of a vertex or edge. k_o^+ / k_o^- effect the cost of increasing/decreasing the flow on $o \in V \cup E$ (thus setting positive/negative values to $\Delta(o)$). Note that the higher the confidence we have in the measurement the higher the cost to change its measured value will be.

$w(o)$ represents an initial flow estimation on $o \in V \cup E$, this is explained thoroughly in section 5, but for now it is sufficient to think of it as the inaccurate flow as measured on $o \in V \cup E$.

The \ln function is used to normalize the weight of the edge/vertex which is in the denominator of the cost function, thus creating a denser distribution of costs.

For any application of the technique for filling in the missing/inaccurate gathered profile information we limit ourselves to filling in intra-procedural (local) missing frequencies.

The algorithm for applying the intra-procedural fixes is as follows:

1. foreach f in the list of functions do
 - a. build control flow graph g for f
 - b. foreach $o \in V \cup E$ in G assign values to the confidence constants $k^\pm(o)$ and the weight function $w(o)$ (see Setting the constants ahead)

- c. Build a fixup graph G' (see section 4)
- d. apply the minimum cost circulation algorithm to G' to find the minimum flow function f
- e. Retrieve the fixup vector Δ from f as explained in section 4.

Setting the constants: $k^\pm(o), w(o)$

Setting $w(o)$: Youfeng Wu and James R. Larus suggest techniques to estimate edge probabilities [4]. We determine the probability for each edge $p(\langle u, v \rangle)$ by using static profile techniques as suggested in their paper. The weight for each edge is then set as follows:

$$w(\langle v, u \rangle) = w(v) \cdot p(\langle v, u \rangle)$$

Setting $k^\pm(o)$: we set the value for the confidence constants as follows:

$$k_o^\pm = a^\pm \cdot b, \quad a^+ = 1, a^- = 50, b = \sqrt{\text{avg_vertex_weight}(cfg)}$$

Note that the b parameter is just for normalization so it's not very important. Setting a^+ and a^- as shown above, worked well because it made the cost of decreasing flow on a vertex/edge significantly larger than that of increasing the flow on it. If we would have given a^+ and a^- similar values we would end up with a fixup vector that cancels most of the measured flow, as the trivial solution that cancels all the flow on the edges may be very appealing. On the other hand setting them farther apart caused edges and vertices with $w=0$ to increase dramatically which is also undesirable.

6 Experimental results

To evaluate the effectiveness of our profile fixup technique, we applied our algorithm on benchmarks from the SPECint2000 suite and measured the accuracy using a criterion called degree of overlap. The comparison was done between the fixed dynamic control flow graph of the program and the actual dynamic control flow graph collected using full edge instrumentation and profile collection on the ref input. In order to gather complete edge profiling we used the IBM post-link optimizer FDPR-Pro [8] which can statically instrument a given executable and generate an instrumented version which produces an accurate edge profile file when run.

We also measured the performance impact on each of the SPECint2000 benchmarks when applying FDPR-Pro -O3 optimizations when using as profile input the low-rate sampling profiling fixed by our technique versus full accurate profiling gathered by FDPR-Pro instrumentation.

In addition we also refer to an addition measure called degree of overlap. The degree of overlap metric is used to compare the completeness of one control flow graph with respect to another, and has been used in several other research papers [15, 16, 17, 18]. The definition is as follows:

$$overlap(cfg1, cfg2) = \sum_{e \in E(cfg1) \cap E(cfg2)} \min(pw(e, cfg1), pw(e, cfg2))$$

Where $pw(e, cfg)$ is defined as the percentage of cfg 's total edge weights represented by the edge weight on e . Only edges on both CCT1 and CCT2 are counted, in our specific problem there edge sets of $cfg1$ and $cfg2$ are identical so that $E(cfg1) \cap E(cfg2) = E(cfg1) \cup E(cfg2)$. The degree of overlap indicates how $cfg2$ overlaps with $cfg1$ or how $cfg2$ is covered by $cfg1$. The degree of overlap range is from 0% to 100%.

The experiments were conducted on the IBM AIX POWER 5 platform with SPECint2000 benchmarks compiled for 32bit. The SPECint2000 or CPU2000 suite is primarily used to measure workstation performance but was designed to run on a broad range of processors as stated in [20]: "SPEC designed CPU2000 to provide a comparative measure of compute intensive performance across the widest practical range of hardware". Although it may be hard

to imagine that applications such as gcc (C compiler), vpr (circuit placement), or twolf (circuit simulation) running on hand held devices, others such as gzip (compression), parser (word processing), and eon (visualization) are sure to be.

The sampling data was collected using the IBM AIX tprof command which samples running applications and makes use of the POWER 5 hardware counters. The cell Oprofile tool provides similar capabilities for the cell embedded processor [22, 23]. For collecting the sampled frequency for the SPECint2000 programs we sampled the Instruction Complete hardware counter (PM_INST_CMPL) every ~1,000,000 instructions. This created a relatively small overhead of 2% - 3% on the runtime of each sampled program.

6.1 Filling edge profile from vertex profile

We first began by using FDPR-Pro to collect basic block profile alone, on the SPECint2000 benchmarks and then apply our technique to fill in the missing edge profile. Our measurements show, that when using our technique in such a way that the confidence constants for changing counts on basic blocks are set to ∞ , and costs for changing costs on edges is set using the heuristics proposed by Youfeng Wu and James R. Larus [4], we reach an average degree of overlap that is higher than 99%, which obviously yields a negligible immeasurable performance delta when comparing to using full edge profile collected by FDPR-Pro. The conclusion is therefore, that when using our proposed method for filling in the missing edge profile, vertex profile is as good as edge profile for any practical purpose.

6.2 Approximating dynamic control flow for external sampling

A more challenging problem is creating an approximation for full edge profile when only partial, external sampling information exists. For this purpose we used tprof, which is an AIX tool for externally collecting hardware events using sampling. We used tprof to collect instruction complete events once every 1,000,003 events. Collecting the profile at such a low rate reduces the run-time of the SPECint2000 applications by 2-3%. Note that selecting a prime number as the event counter is advised since it reduces the chance for synchronization in a loop. If, for example, a trace containing 100 instructions occurs many millions of times sequentially and we would sample every 1,000,000 events we would hit the same instruction every time, and this would yield a false view on the counts in that trace of instructions.

The initial flow estimate $w(v)$ is set as follows:

$$w(v) = \frac{\sum_{ins \in V} sampled(ins)}{num_of_instrs(v) \cdot sample_rate}$$

To measure the effectiveness of our technique we compared the degree of overlap and the performance gain with the full edge profile collected by FDPR-Pro and compared it to the degree of overlap and performance gain that was achieved without applying our method, which uses the calculated $w(o)$ (see above) for each $o \in V \cup U$. The results of our measurements are presented in tables 1, 2 and in figure 7.

Table 1 - Degree of overlap comparison

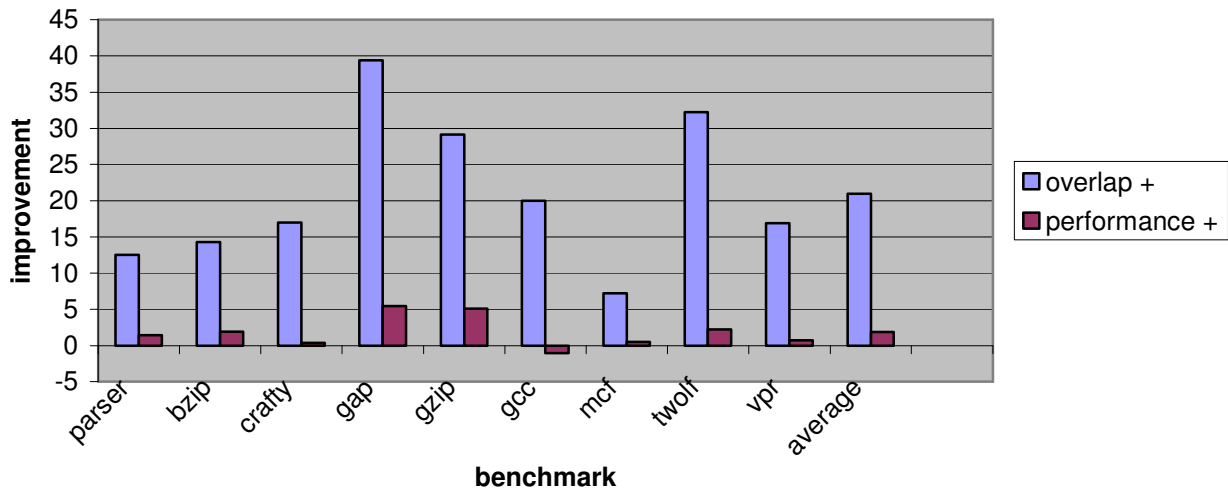
Benchmark	degree of overlap: Sampled profile vs. Full profile	Degree of overlap: fixed Sampled profile vs. Full profile
parser	72%	81%
bzip	70%	80%
crafty	53%	62%
gap	66%	92%
gzip	72%	93%
gcc	65%	78%
mcf	83%	89%
twolf	66%	82%
vpr	71%	83%
Average	69%	82%

Table 2 - Runtime comparison

benchmark	Runtime after FDPR -O3 using sampled	Runtime after FDPR -O3 using fixed sampled	Runtime after FDPR -O3 using full profile
parser	5.2%	6.6%	7.2%
bzip	3.8%	5%	5%
crafty	4.1%	4.5%	6%
gap	8.5%	13.5%	13.2%
gzip	12%	16.5%	16.5%
gcc	4.35%	3.4%	4.4%
mcf	8%	8.5%	8.5%
twolf	10.25%	12.25%	14.1%
vpr	8.2%	8.8%	9.3%

Note: The percentages above refer to performance improvement compared to the base runtime.

Figure 7 - Fixed vs. unfixed sampling



The average degree of overlap, using our technique, calculated on SPECint2000 is 82% compared to 62% without using the fix. The average performance gain is only 0.6% less than when using the full edge profile, while without using the suggested fix, the average performance gain is 2.2% less than the full edge profile. Finally, the average improvement in degree of overlap is 21% and we reach a 1.8% average improvement in performance when compared to not using our fixup algorithm.

7 Future directions

Our fixup technique can be used for a wide variety of profiling problems. Collection of inaccurate or lacking profile information may be due to several reasons, other than those addressed in the paper, such as the following:

After applying several optimizations, such as function cloning, inlining, or after applying optimizations such as constant/value-range propagation which may eliminate edges in the control flow graph, the original profile information becomes inconsistent and needs to be corrected. In most cases, re-running the profiling phase on the modified program is not desirable.

When profiling a multithreaded or multiprocessed application some counter promotions may be missing as a result of multiple threads/processes incrementing the same counter without synchronization. Adding synchronization to each vertex's/edge's counter may be undesirable due to additional runtime overhead and additional memory to be used as a mutex for each basic block/edge.

When reusing profile information from older versions of the program.

The former is an interesting problem that may benefit from applying our technique. When inlining or cloning a method for a particular call site a few types of static analysis can be performed, such as constant or value range analysis, global value numbering etc ... As a result some of the edges in the static control flow graph may become infeasible. For example an if statement that checks if a pointer is NULL and the value of the pointer is known not be NULL at a specific call site, will eliminate the if statement altogether. By eliminating edges in the control flow graph, some of the dynamic counts will become infeasible as well, for example all the paths within the eliminated if statement or values that are set within the if statement that can not be executed anymore. In addition to the static analysis techniques, such as the ones suggested above, an inlined function may have multiple return instructions. See figure 7 for such an example.

Figure 8 - Using multiple epilogue basic blocks to determine flow

```
void foo () {  
    if (globA > 100) {  
        ...  
        return; //return 1  
    }  
    if (globA < 50) {  
        ...  
        return; // return 2  
    }  
    ...  
    return; // return 3  
}
```

Assume we have the following callers:

caller1: globA = 101; foo();

caller2: globA=49; foo();

When inlining foo to caller1, if full edge profile exists, we could find that only return 1 actually returns to caller1. Therefore when inlining foo to caller1 we should cancel all paths leading from foo's prologue basic block to return2 and return3. Using our proposed technique we could cancel the infeasible paths effectively.

Another future direction can be finding the optimal flow-fix, our fixup vector, with respect to different cost types such as minimizing L_∞ or L_2 (the least mean squares) and the weighted version of each. The weighted L_∞ can be minimized by using linear program along with a form of binary search. The weighted L_2 can be minimized by convex optimization using interior-point methods, Lagrange methods and several others.

8 Summary

We defined a technique for effectively fixing inaccurate and incomplete control flow profile information. This allows using non-intrusive low overhead profile techniques to collect profile for real-time embedded applications and then effectively using our technique to enhance the profile data and make it reasonably accurate. We implemented our technique and measured how well it performs when filling in edge profile from vertex profile and from instruction complete event counter run on a set of representative benchmarks. We showed that when applying over vertex profile, edge profile can be derived almost perfectly and when applying over the suggested sampling technique, we may reach an average overlap degree of 82%. When applying our technique into a post-link optimizer called FDPR-Pro, we reach an average performance gain which is only 0.6% less than when using full, accurate edge profile gathered using edge instrumentation. More generally, this suggests a platform independent, low overhead profiling scheme (2-3% overhead) with a high degree of accuracy. In addition we also show that when applying our technique over vertex profile we can fill in the missing edge profile with almost perfect overlap.

Bibliography

- [1] Thomas Ball, James R. Larus, "Optimally profiling and tracing programs" in *Appears in ACM Transactions on Programming Languages and Systems*, July 1994.
- [2] J. Anderson, L. M. Bert, J. Dean, S. Ghemawat, M. R. Henzinger, S-T. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. "Continuous profiling: Where have all the cycles gone?" *In Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.
- [3] Reps, T., Ball, T., Das, M., and Larus, J., "The use of program profiling for software maintenance with applications to the Year 2000 Problem". In *Proceedings of ESEC/FSE '97: Sixth European Software Engineering Conference and Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, (Zurich, Switzerland, Sept. 22-25, 1997), *Lecture Notes in Computer Science*, Vol. 1301, M. Jazayeri and H. Schauer (eds.), Springer-Verlag, New York, NY, 1997, pp. 432-449.
- [4] Youfeng Wu, James R. Larus. "Static Branch Frequency and Program Profile Analysis". In *27th IEEE/ACM InterÕl Symposium on Microarchitecture (MICRO-27)*, Nov. 1994.
- [5] V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36:873–886, 1989. Preliminary version appeared in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 388–397, 1987.
- [6] G. Haber, E.A. Henis, and V. Eisenberg, "Reliable Post-link Optimizations Based on Partial Information", *Proceedings of the 3rd Workshop on Feedback Directed and Dynamic Optimizations*, December 2000.
- [7] E. A. Henis, G. Haber, M. Klausner and A. Warshavsky, "Feedback Based Post-link Optimization for Large Subsystems", *Second Workshop on Feedback Directed Optimization*, Haifa, Israel, November 1999, pp. 13-20.

- [8] Nahshon and D. Bernstein. "FDPR - A Post-Pass Object Code Optimization Tool", *Proc. Poster Session of the International Conference on Compiler Construction*, pp. 97-104, April 1996.
- [9] T. Romer, G. Voelker, D. Lee, A. Wolman, W. Wong, H. Levy, B. Bershad. and B. Chen, "Instrumentation and Optimization of Win32/Intel Executables Using Etch", *Proceedings of the USENIX Windows NT Workshop*. August 1997, pp. 1-7.
- [10] B. Schwarz, S. Debray, G. Andrews, and M. Legendre, "PLTO: A link-Time Optimizer for the Intel IA-32 Architecture", In *Proceedings of Workshop on Binary Rewriting*, Sept 2001.
- [11] R. Cohn, D. Goodwin, and P. G. Lowney, "Optimizing Alpha Executables on Windows NT with Spike", *Digital Technical Journal*, vol. 9, no. 4, Digital Equipment Corporation 1997, pp. 3-20.
- [12] R. Muth, S. Debray, S. Watterson, "alto: A Link-Time Optimizer for the Compaq Alpha", Technical Report 98-14, Dept. of Computer Science, The University of Arizona Dec. 1998.
- [13] Pohua Chang, et al., "Using Profile Information to Assist Classic Code Optimizations," *Software-Practice and Experience*, vol. 21, no. 12 (1991): 1301-1321.
- [14] T. Ball and J. R. Larus, "Optimally Profiling and Tracing Programs," *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 4, pages 1319-1360, July 1994.
- [15] M. Arnold and B. Ryder. A framework for reducing the cost of instrumented code. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 168–179, 2001.
- [16] M. Arnold and P. F. Sweeney. Approximating the calling context tree via sampling. IBM Research Report, July 2000.
- [17] P. T. Feller. Value profiling for instructions and memory locations. *Masters Thesis CS98-581, University of California San Diego*, April 1998.
- [18] Xiaotong Zhuang, Mauricio J. Serrano, Harold W. Cain. Accurate, Efficient, and Adaptive Calling Context Profiling, In *PLDI 2006*.

- [19] D.E Knuth and F.R Stevenson. Optimal measurement points for program frequency counts. BIT, 13:313-322, 1973.
- [20] SPEC CPU2000: <http://www.spec.org/cpu2000>
- [21] Madalene Spezialetti and Rajiv Gupta, “Timed Perturbation Analysis: An Approach for Non-Intrusive Monitoring of Real-Time Computations”, ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems, Orlando, Florida, June 1994
- [22] Cell SPE Oprofile patch: <http://patchwork.ozlabs.org/linuxppc/patch?id=9627>
- [23] Cell alphaworks SDK: <http://www.alphaworks.ibm.com/topics/cell>

השלמת פרופיל קשתות ע"י שימוש באלגוריתמים למציאת סירקולציה מינימאלית

רועי לוין

תקציר

שימוש בפרופיל קשתות היא שיטה מאוד נפוצה לקבל משוב על התנהגות תוכנית מחשב שניתן להשתמש בו באופן סטטי או דינאמי ע"י מהדר או כלי אופטימיזציה שרצים אחרי שלב הלינק (post-link) על מנת להפיק תוכניות יעילות ומהירות ביותר. אותו כלי משתמש באינפורמציה של המשוב כדי להפיק תועלת עבור האופטימיזציות מונחות הפרופיל שלו ע"י השקעת המשאבים בקוד שמתבצע בתדירות גבוהה יותר. למרות זאת, איסוף פרופיל קשתות מלא היא פעולה המשפיעה לרעה באופן משמעותי על זמן הריצה של תוכנית, דבר אשר גורם לשיטה זו להיות בלתי ישימה עבור הידור דינאמי. מאידך, לאיסוף פרופיל קשתות מלא באופן סטטי יש צורך בכלי אינסטומנטציה שמסבכים ומאיטים את זמן הידור ובניית התוכנית. בנוסף לאיזון הקלאסי שבין תהליך הידור ובניה ארוך ומורכב לזמן ריצה מהיר של התוכנית, יש בעיה נוספת שמתעוררת בתוכניות זמן אמת. השימוש בטכניקות איסוף פרופיל כבדות, כמו הכנסת קוד אינסטרומנטציה כבד ופולשני, המערכת עלולה לא לעמוד במשימותיה בזמן, ובכך לשנות את התנהגותה. בעבודה זו אנו מצעים שיטה חדשנית לקירוב גבוהה של פרופיל קשתות מלא. שיטה זו מאפשרת שימוש בשיטות איסוף פרופיל בלתי פולשניות ובעלות תקורה נמוכה, כמו דגימת מספר הפקודות שבוצעו, לאיסוף פרופיל כללי ולא מדויק. שימוש באינפורמציה זו, כפי שהיא, תפיק תוצאת בלתי אופטימליות, לכן אנו משתמשים בשיטה המוצאת בעבודה זו על מנת להשלים ולתקן את הפרופיל, הלא מדויק, שנאסף בכדי להגיע לקירוב טוב של הפרופיל האמיתי, כלומר פרופיל הקשתות המלא. אנו מתרגמים את הביעה הזו, של השלמת הפרופיל הבלתי מדויק, לבעיית **מציאת סירקולציה מינימאלית** ומצעים פתרון המבוסס על חוקי שימור הזרימה. ההנחה היא שעל ידי יצירת זרימה, אשר שומרת על חוקי שימור הזרימה, אך ע"י מינימיזציה של קריטריון המהווה מדד לכמות השינוי שבוצע, נגיע לקירוב טוב יותר של הפרופיל האמיתי. על מנת לאשש הנחה זו, אספנו פרופיל ע"י שימוש בכלי דגימה בעלי תקורה נמוכה שגרמו לתאווה של בין 2%-3% בלבד בזמן ריצת האימון, ולאחר מכן השתמשנו בסכימה שלנו כדי לשפר ולתקן את

הפרופיל, הלא מדויק, שנאסף והזננו את הפרופיל המתוקן לכלי אופטימיזציה שמופעל אחרי הלינק (post-link optimizer) שנקרא FDPR-Pro בכדי להפיק תוכנית הרצה יעילה. כשהשוונו את התוכנית, שהופקה ע"י שימוש בפרופיל המתוקן, לתוכנית שהופקה ע"י אותו כלי (FDPR-Pro) כשהונחה לאסוף פרופיל קשתות מלא, מצאנו כי אחוז השיפור מעל התוכנית המקורית (ללא אופטימיזציות) קטן ב 0.6% בלבד מזה שניתן להשיג אם משתמשים בפרופיל הקשתות המלא, על כל חסרונותיו. גישה חדשה זו לבעיית קירוב הפרופיל החלקי לפרופיל קשתות מלא מציעה למעשה פתרון גנרי בלתי תלוי בפרטי הארכיטקטורה של המכונה בה אנו מריצים את התוכנית.

השלמת פרופיל קשתות ע"י שימוש באלגוריתמים למציאת סירקולציה מינימאלית

מאת: רועי לויך
בהנחיית: פרופ' אילן ניומן
דר' גדי הבר (מעבדות IBM)

עבודת גמר מחקרית (תיזה) המוגשת כמילוי חלק מהדרישות

לקבלת התואר "מוסמך האוניברסיטה"

אוניברסיטת חיפה

הפקולטה למדעי החברה

החוג למדעי המחשב

אוגוסט 2007

השלמת פרופיל קשתות ע"י שימוש באלגוריתמים למציאת סירקולציה מינימאלית

רועי לוי

עבודת גמר מחקרית (תיזה) המוגשת כמילוי חלק מהדרישות

לקבלת התואר "מוסמך האוניברסיטה"

אוניברסיטת חיפה

הפקולטה למדעי החברה

החוג למדעי המחשב

אוגוסט 2007