The following paper was originally published in the
Proceedings of the 2nd USENIX Windows NT Symposium
Seattle, Washington, August 3–4, 1998

Evaluating the Importance of User-Specific Profiling

Zheng Wang
Harvard University
Norm Rubin
Digital Equipment Corporation

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Evaluating the Importance of User-Specific Profiling

Zheng Wang
zhwang@eecs.harvard.edu, Division of Engineering and Applied Sciences, Harvard University
Norm Rubin
rubin@ives.amt.tay1.dec.com, Digital Equipment Corporation

Abstract

This paper examines common assumptions about user-specific profiling in profile-based optimization. We study execution profiles of interactive applications on Windows NT to understand how different users use the same program. The profiles were generated by the DIGITAL FX!32 emulator/binary translator system, which automatically runs the x86 version of Windows NT programs on NT/Alpha computers. We have found that people use the benchmark programs in different ways. These differences in program usage can have impact on the performance of profile-based FX!32 program translation and optimization, up to 9%.

1. Introduction

1.1 Background

Profile-based optimization is predicated on the assumption that profiles can be obtained that accurately depict how users run the application. There has been only limited research on the viability of this assumption [FF92, GY96]. We address this problem by examining different users' usage patterns of interactive applications on Windows NT. Here the term "usage pattern" refers to the way a particular individual uses the code in a particular program.

A common assumption in profile-based optimization is that people use applications in similar ways. This assumption is consistent with the behavior of batch-style and computation-intensive programs, and has an implication that user-specific profiling is unnecessary. The alternative to this common assumption is that people use applications in different ways. This is consistent with our intuition for complex and feature-rich programs such as GUI-based interactive applications. It implies that user-specific profiling may be necessary for effective optimization.

The assumption that users are similar or users are different suggests two different models for applying profile-based optimization. In the traditional model, an application is profiled and optimized before its release. Developers run the program with a fixed or arbitrary training workload and use the profile to guide optimizations. Based on the assumption that users are similar, the training workload is considered to be representative. Spike [CG97] is an example of this approach. Some recent systems, such as Morph [ZW97] and FX!32 [HH97], extend the optimization process beyond an application's release by profiling and optimizing the application continuously while it is used[1]. Based on the assumption that users are different, the current versions of Morph and FX!32 operate on a per-user basis. Another assumption in this model is that a particular user's usage pattern may change over time but seldom changes abruptly.

In this study, we compare execution profiles from different users of the same program module. We did not tackle the question of how a particular user's usage pattern changes over time. Since the purpose of the profiles was to guide optimizations, we investigate how the differences in profiles affect optimization performance. We also examine whether we can combine profiles from a group of users to optimize programs for those users and for other users.

Our study shows that users of interactive applications have different usage patterns. For most programs, each individual uses a set of procedures that no other users do, although frequently executed procedures tend to be used by most users. For some benchmarks, profiles from another user or a group can be less effective for optimization than a particular user's own profile.

1.2 Related Work

Although the majority of today's personal computers run mostly interactive applications on Windows systems, there has been little research on how people use these programs. Several research projects investigated Windows operating system performance [CE96, EW96, PS96]. A recent paper [LC98] presented measurements and simulation results of instruction set and architectural characteristics during program execution on x86 Windows NT. These projects focused on the characteristics and comparison of the general system performance, while this paper focuses on the application usage patterns.

[1] Usually, profiling is done continuously when the application is running and optimization is delayed until off-line.

There has been some research on profile comparison for the purpose of branch prediction. Fisher and Freudenberger [FF92] examined the accuracy of predicting conditional branch directions from previous runs of a program. Their experiments focused on batch-computation programs from SPEC benchmark suit, and used subjectively selected datasets to generate profiles. Gloy et al. [GY96] compared user-only traces and full-system traces for dynamic branch prediction. They used standard traces as well as traces from instrumented runs of selected programs. Our profile analysis is aimed for optimization in general, and our profiles were collected from users' unscripted usage of interactive applications.

The next section introduces our experimental methodology, including the collection of the profiles and the statistical analysis methods. Section 3 presents the results, and Section 4 summarizes.

2. Methodology

2.1 FX!32 and FX!32 profiles

We used the DIGITAL FX!32 system to collect execution profiles for Windows NT programs. FX!32 automatically runs x86 applications on Alpha NT, using a combination of emulation and binary translation. When an x86 image is executed under FX!32 for the first time, the FX!32 emulator interprets the x86 code and generates an execution profile. This profile is later used by the FX!32 translator to generate translated and optimized Alpha code. Subsequent executions of the program use the translated code instead of the x86 code. If a certain run of the program uses part of the x86 code that has not been translated, new profile data are generated and merged into the existing profile. The merged profile can be used to re-translate the program.

The contents of the profile reflect program usage over time by a particular user and the addition of new profile data reflects new or changed usage. Therefore, we can learn about the usage patterns of the x86 applications by studying the users' FX!32 profiles. FX!32 profiles are generated during the emulation of x86 code, so they are based on x86 traces, not Alpha traces.

Currently, FX!32 profiles contain information on procedure calls, indirect control transfers, and unaligned memory references. For our statistical analysis of the profiles, we consider only the procedure execution information. In the optimization results study, however, the whole profile is used for the FX!32 program translation/optimization.

One side note is that FX!32's view of program procedures may differ from the set of procedures in the source code. FX!32 works on the binary image and discovers program procedures during emulation. It combines two procedures into a single "FX!32 procedure" if one contains a jump into the other. Therefore, an FX!32 procedure may be the combination of several original procedures. This does not occur frequently, nor does it fundamentally affect our profile analysis. In the rest of the paper, we simply use the term "procedure" to refer to an FX!32 procedure.

2.2 Profile collection

We chose a set of interactive desktop applications as benchmarks for this study. Since FX!32 profiles are generated separately for each program module, we regard each module as a separate benchmark. Different versions of the same program are treated as different modules because they have different code images. For each module, we collected multiple (four or more) profiles, each from a different user.

Our benchmark selection includes executables and DLLs from the Microsoft Office suite as well as other commonly used applications. Table 1 lists the 14 benchmark modules used for this paper. The Office executables and DLLs are noted with their version numbers: 95 (Office Version 7.0 for Windows 95) and 97 (Office 97).

A group of computer system researchers and software developers ran the x86 version of the benchmarks using FX!32 on Alpha computers running Windows NT 4.0. Profiles were generated from their spontaneous and natural usage of the programs. For each module, we collected individual profiles from a selected group of users, each of whom had made significant use of the module[2]. These individual profiles were generated from copies of the module on different machines. Since the machines had comparable hardware and software configuration, differences in the profiles were mostly artifacts of the users' usage patterns and not the execution environment[3].

We calculate a *combined profile* from the individual profiles using the same merging algorithm used by the FX!32 Manager, which sums up the execution counts for each entry. The combined profile represents the

[2] This is evaluated by looking at the profile size, the run count (number of times a module has been executed), and asking the users themselves. The run count alone is not a good indication, because one run of an interactive application may involve a varying number of tasks. Typically, the run count is larger than 10 for these profiles.

[3] As verification, we compared the profiles generated from running an automated script on two machines, and found them virtually identical.

Benchmark Module		Description	Time/Date Stamp	File Size (KB)
MS Office	excel.exe (97)	Office 97 Excel main executable	16:22:31 11/15/96	5469
	mso95.dll (95)	Office 95 (Version 7.0) DLL	01:48:53 07/08/95	918
	mso97.dll (97)	Office 97 DLL	01:02:35 11/07/96	3686
	outllib.dll (97)	Office 97 Microsoft Outlook DLL	20:33:23 11/13/96	4254
	powerpnt.exe (97)	Office 97 PowerPoint executable	05:08:38 11/17/96	3411
	winword.exe (95)	Office 95 Word executable	02:20:46 07/12/95	3755
	winword.exe (97)	Office 97 Word executable	12:33:37 11/15/96	5194
acrord32.exe		Adobe Acrobat Reader 3.0 executable	16:59:11 06/16/97	2265
mfc40.dll		Microsoft Visual C++ 4.0 DLL	01:53:24 02/28/96	901
netscape.exe		Netscape Navigator Gold 3.01 executable	15:42:53 10/22/96	3093
photoshp.exe		Adobe Photoshop 4.0 executable	09:23:00 10/29/96	3560
pnui3250.dll		Support library for RealPlayer (32-bit) 5.0	22:54:00 11/22/97	590
winhlp32.exe		Windows NT 4.0 help utility	14:17:01 07/17/96	303
winzip32.exe		WinZip compression utility 6.2	17:25:35 10/12/96	846

Table 1. Summary of benchmark modules
Time/Date Stamp is taken from the PE file header.

combined usage of the module by this group of users. We use a series of statistical analyses to examine the similarity between individual profiles and the change in similarity when the profiles grow. We also use the profiles to guide the FX!32 program translation/ optimization and compare the performance of translated programs.

2.3 Statistical analysis

Here we introduce the key methods of our statistical analysis and introduce some terminology that is used in this paper.

An FX!32 profile contains *execution counts* (the number of times a procedure is called) for all procedures that were used during the profile generation. When we compare a group of profiles, we focus on the set of procedures included in each profile, regardless of the procedures' execution counts. This parameter is important for the profile-guided code translation in FX!32 as well as most code layout optimizations. By considering only the set of procedures, we also simplify the comparison and avoid unfair methods of weighing the execution counts. We consider the procedure execution counts only in the part of our analysis that examines the correlation between the number of users who use a procedure and the execution count of the procedure.

We compare the sets of procedures used by individual users by examining their combined profile. If a procedure is included in the combined profile, it has been used by at least one user. For such a procedure, we define its *usage count* as the number of users who have used it. We categorize the procedures in the combined profile according to their usage counts. If a procedure is used by only one user, we call it a *unique procedure*. If it is used by all users, we call it a *common procedure*. Any other procedure is called a *subgroup procedure*.

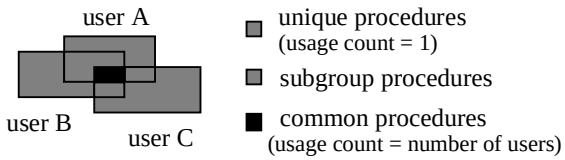

Figure 1. An example of unique, subgroup and common procedures

The usage count distribution of all procedures in the combined profile reflects the similarity between individual profiles. If all users use the same set of procedures, all procedures in the combined profile will be common procedures. If each user uses a different set of procedures, all procedures in the combined profile will be unique procedures. If the sets of procedures used by the individuals are not all the same nor all different, we will see a distribution of unique, subgroup and common procedures in the combined profile. The higher the percentage of common procedures and the lower the percentage of unique procedures, the more similar the individual profiles.

3. Results

In this section, we present a series of statistical analyses of the collected profiles as well as optimization results

Benchmark Module	Number of Users	Number of Procedures in Profile				Smallest %	Largest %	Average %
		Combined	Smallest	Largest	Average			
acrord32.exe	4	5050	4012	4790	4435	79.4%	94.9%	87.8%
excel.exe (97)	4	8514	5885	6821	6351	69.1%	80.1%	74.6%
mfc40.dll	7	2558	1260	1932	1539	49.3%	75.5%	60.2%
mso95.dll (95)	6	2630	1693	2115	1927	64.4%	80.4%	73.3%
mso97.dll (97)	8	9994	5631	8298	6870	56.3%	83.0%	68.7%
netscape.exe	4	7938	4849	7004	5852	61.1%	88.2%	73.7%
outllib.dll (97)	5	16330	10541	13113	11904	64.6%	80.3%	72.9%
photoshp.exe	5	10502	6981	8845	7807	66.5%	84.2%	74.3%
pnui3250.dll	4	1443	1055	1299	1181	73.1%	90.0%	81.8%
powerpnt.exe (97)	5	15014	8905	12895	10504	59.3%	85.9%	70.0%
winhlp32.exe	12	762	543	727	602	71.3%	95.4%	79.0%
winword.exe (95)	5	7317	4600	6222	5398	62.9%	85.0%	73.8%
winword.exe (97)	6	10113	6226	7988	6972	61.6%	79.0%	68.9%
winzip32.exe	5	1125	597	893	737	53.1%	79.4%	65.5%
					Average	63.7%	84.4%	73.2%

Table 2. Summary of profiles

Combined	The number of procedures in the combined profile
Smallest, Largest, Average	The smallest, largest and average number of procedures in an individual profile
Smallest%, Largest%, Average%	"Smallest", "Largest" and "Average" each divided by "Combined"

For every module, each user has used a percentage of the procedures in the combined profile, and *Smallest*%, *Largest*% and *Average*% are the minimum, maximum and average value of this percentage among the group of users.

for two benchmark modules. In our statistical analyses, we examine the similarity between individual profiles and the change in similarity over time.

3.1 Summary of profiles

For every benchmark module, Table 2 lists the number of procedures in the combined profile and the smallest, largest and average number of procedures in an individual profile[4]. Each individual profile reflects one person's usage of the benchmark module over multiple runs, while the combined profile reflects the combined usage by all users of the module. Therefore, each individual profile includes a percentage of the procedures in the combined profile. In Table 2, the lowest *Smallest%* is 49.3% for `mfc40.dll`, which means that one user of `mfc40.dll` has used only 49.3% of all procedures used by the seven users. The highest *Largest%* occurs for `winhlp32.exe`, where one user has used 95.4% of all procedures used by the 12 users. For any module, *Largest%* is 100% if and only if one user has used all procedures used by other users. This does not occur in Table 2, implying that people user the programs in different ways. The average percentage of procedures in the combined profile used by each individual is about 73%.

3.2 Similarity between individual profiles

As discussed in Section 2.3, the usage count distribution of all procedures in the combined profile reflects the similarity between individual profiles. Figure 2 shows the percentage distribution of unique, subgroup and common procedures in the combined profiles for all benchmark modules.

For these benchmark modules, the percentage of common procedures in the combined profile ranges between 38.1% and 76.9%, with an average of 52.1%. The percentage of unique procedures ranges between 7.0% and 23.6%, with an average of 16.4%. In other words, typically about half of the procedures ever used are used by all users, while a small yet significant percentage is used by only one of the users.

Among 14 benchmark modules, `acrord32.exe`, `pnui3250.dll` and `winhlp32.exe` have the highest percentage of common procedures in their

[4] As mentioned in Section 2.1, the term "procedure" refers to an "FX!32 procedure" which may be the combination of several original procedures. The total number of FX!32 procedures in a module is difficult to determine.

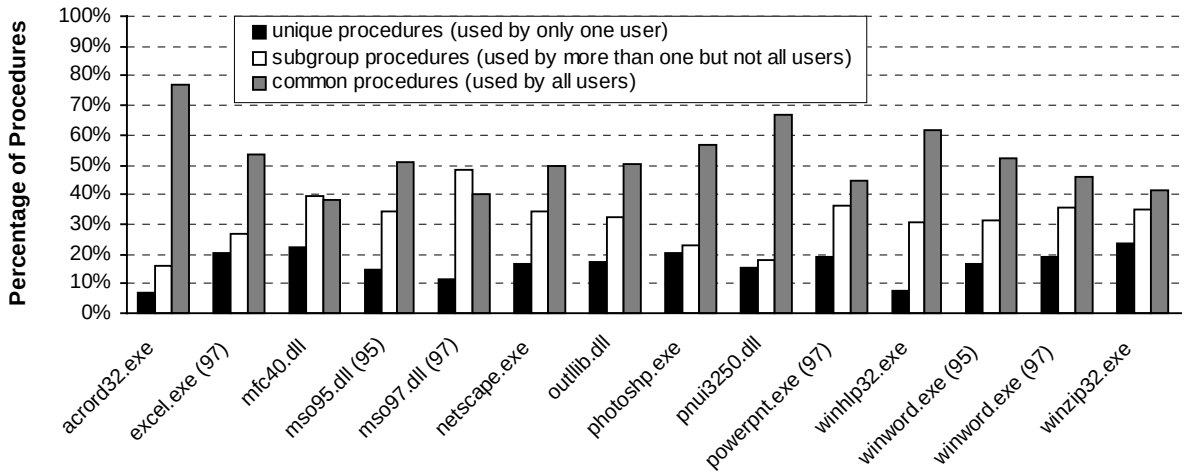

Figure 2. Usage count distribution for all benchmark modules

The Y-axis is the percentage of procedures in the combined profile that fit into a given category. The number of users and the total number of procedures in the combined profile for each benchmark module can be found in Table 2.

combined profiles, and `acrord32.exe` and `winhlp32.exe` also have the lowest percentage of unique procedures. This indicates that each of these three modules shows relatively consistent usage pattern across its users. We notice that these three modules provide less variety of functionality than most other benchmark modules. For example, `acrord32.exe` was mostly used to simply view and print documents downloaded from the Internet. We may also see from Table 1 and Table 2 that `pnui3250.dll` and `winhlp32.exe` are two of the smallest benchmark modules in terms of the file size and the number of procedures. The above two factors may explain the relatively high similarity among each of these three modules' group of individual profiles.

For a more detailed examination, we calculate the distribution of procedure usage counts within each individual profile. Table 3 shows the results for `winword.exe` (95).

Profiled User	# of Proc.	Number of Proc. by Usage Count		
		1 (*unique*)	2-4 (*subgroup*)	5 (*common*)
Bashful	4600	55	722	3823
Doc	4990	69	1098	3823
Grumpy	5332	210	1299	3823
Sneezy	5846	312	1711	3823
Happy	6222	562	1837	3823
Combined	7317	1208	2286	3823

Table 3. Procedure distribution among five users of `winword.exe` (95)

# of Proc.: number of procedures in the profile

We have replaced all user names with pseudonyms. For each user, unique procedures are those used by this user but none of the other four. Common procedures are those used by all five users.

We see that every individual profile has its share of unique procedures and subgroup procedures. The common procedures constitute between 61.4% and 83.1% of the procedures in an individual profile, while the percentage of unique procedures in an individual profile ranges from 1.2% to 9.0%. In terms of the procedures included, none of the profiles is a subset or superset of another profile. We have observed similar phenomena for other benchmark modules. For several benchmark modules, one or more relatively small individual profiles have no unique procedures, but they still contain subgroup procedures. `winhlp32.exe` and `netscape301.exe` are the only two benchmark modules where one person uses a subset of the procedures another person uses.

We also examine whether there are highly similar usage patterns among small groups of users. To evaluate this, we use pair-wise comparison between all individual profiles for a module to see whether some of them have significantly higher similarity among themselves than with other profiles. For each pair of profiles, we calculate the percentage of procedures included in both among all procedures included in either of them. This percentage measures the similarity between two profiles. Table 4 lists the results for `winword.exe` (95). All numbers in the table fall between 66.6% and 77.2%, indicating that similarity between each pair of users is on a close level. In the analysis for other

	Bashful	Doc	Grumpy	Sneezy	Happy
Bashful	--	77.2%	69.6%	69.1%	66.6%
Doc	77.2%	--	73.5%	69.7%	72.5%
Grumpy	69.6%	73.5%	--	76.4%	71.6%
Sneezy	69.1%	69.7%	76.4%	--	76.0%
Happy	66.6%	72.5%	71.6%	76.0%	--

Table 4. Pair-wise comparison between five users of `winword.exe` (95)

The number for each pair of users is the percentage of procedures included in both users' profiles among all procedures included in either of them.

benchmark modules, we have seen a few cases of relatively higher similarity between two or three individuals, but we do not think they are sufficient to conclude that there is particularly high similarity among small groups of users.

Results in this subsection imply that users use applications in different ways, supporting the theory that user-specific profiling is important for effective optimization.

3.3 Correlation between procedure usage count and execution count

In this subsection, we examine whether there is correlation between a procedure's usage count and its execution count. In many profile-based optimizations, priority is given to the most frequently executed code. In this case, procedures with higher execution counts are more important to the optimization than those with lower counts.

The procedure execution counts in FX!32 profiles do not always match the traditional definition of procedure execution count. In FX!32, control transfers within the translated Alpha code are not captured in the profile. Since a user may perform program translation from time to time, a procedure's execution count in the profile may be less than the number of times it has been called. However, experience shows that usually only up to a few percents of the counts will differ by more than one order of magnitude. For Figure 3, we divide execution counts into levels that each covers at least two orders of magnitude. This figure shows the usage count distribution of all procedures in the combined profile for `winword.exe` (95), broken down by their average execution counts.

Among 3842 procedures with average execution counts below 100, about 30% are unique procedures and another 30% are common procedures. Among 724 procedures with average execution counts above 10000, over 90% are used by all or all but one users. These numbers show that frequently executed procedures are more likely to appear in all or most individual profiles than those executed less frequently. In other words, the part of code that users execute the most is similar, despite the differences in their overall profiles. The statistics for other benchmark modules also support this conclusion.

For optimizations that give priority to frequently executed code, this conclusion on similarity suggests that despite the differences among users, we may find a representative training workload that exercises the procedures frequently used by most users. On the other hand, such a workload may not cover enough procedures for any one user. For `winword.exe` (95), 2535 procedures have been used by four or five users with average execution counts above 100, while the smallest individual profile includes 4600 procedures and the combined profile includes 7317 procedures. Further analysis in the context of a specific optimization will determine the tradeoff between using such a user-independent training workload and using user-specific profiling.

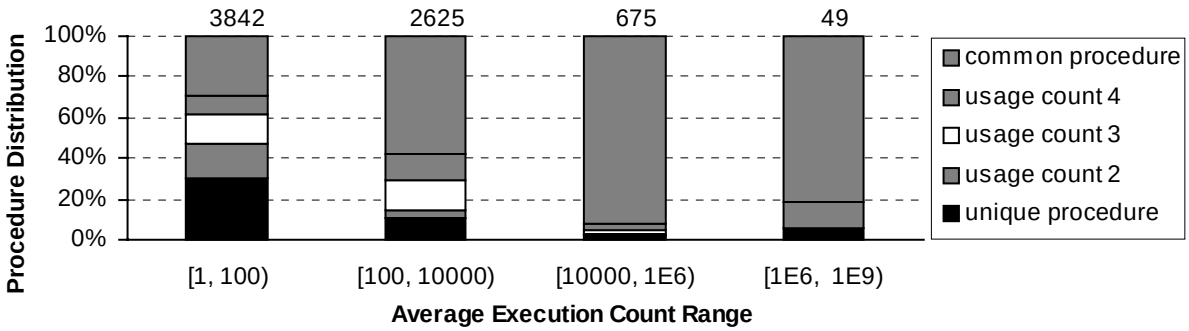

Figure 3. Procedure distribution: usage count vs. execution count: `winword.exe` (95), five users

Average Execution Count: procedure execution count averaged over users who have use this procedure
(Average Execution Count = execution count in combined profile / usage count)
number at the top of each bar: the total number of procedures in this range

3.4 Change in similarity when profiles grow

When the individual profiles grow larger with more use of the program, one might speculate that their similarity increases as they all approach one common limit, the set of all procedures in the program. Table 5 examines the change in similarity between five `winword.exe` (95) profiles when some of them grow larger.

We took snapshots of the five individual profiles at six different times during a month. Each time at least one profile had grown since the last time. During the first five snapshots, the percentage of common procedures in the combined profile increased and the percentage of unique procedures decreased. In these cases, the similarity between individual profiles increased when some of them grew larger. However, the slow change in the similarity suggests that the individual profiles might never "converge"; that is, profiles from different users may never reach a certain high level of similarity, reflecting their different usage patterns. In fact, in the last snapshot where *Sneezy*'s profile grew, the similarity between individual profiles slightly decreased due to the new procedures *Sneezy* had started to use. In summary, the users' accumulated usage patterns may become more similar with more use of the program, but some differences persist.

3.5 Optimization performance

This subsection examines the impact of differences in profiles on the performance of programs optimized using the profiles. This impact is dependent on how the profile information is used during the optimization. Different optimizations may have varying sensitivity to differences in profiles. Even with the same optimization, the performance impact may vary for different programs and different workloads.

In our experiments, the FX!32 translator/optimizer uses the profile to determine the set of code to translate and to guide common compiler optimizations, such as procedure layout, procedure inlining and dead code elimination, on the translated code. We used different training profiles, both individual and combined, to translate/optimize the same module. For each profile, we measured the performance of the application using the translated code. Based on the results we discuss the effectiveness of using profiles from another user or a group of users to translate/optimize a program.

One difficulty in our experiments was measuring the performance of an interactive application. To achieve repeatability, we chose to measure the execution time of a standard, script-driven workload. We consequently assumed that our "test user" performed this same workload for both training and testing. In reality, a user's usage pattern of a program may change over time, causing the testing workload to be different from the training workload. The impact of this factor on program optimization is not investigated in this paper.

We conducted our experiments on benchmark modules `winword.exe` (95) and `powerpnt.exe` (97).

3.5.1 Microsoft Word benchmark

For testing on `winword.exe` (95) from Microsoft Word 7.0 for Windows 95, we used the workload from SYSmark32 for Windows NT version 1.0 distributed by BAPCo [BAPCo]. We included two individual profiles used in the statistical analysis from users *Grumpy* and *Happy*, plus two relatively small individual profiles *Dopey* and *Sleepy* to examine the issue of "under-training". Figure 4 shows the results for these four individual profiles and various combined profiles. All performance measurements were done on a 500MHz Alpha computer with 64MB of main memory. Word 7.0 was the only application running on the system.

Without any translation/optimization, the program is executed entirely through emulation, which is slow (459 seconds). The *Minimal* profile, generated by starting up `winword.exe` and exiting immediately, is practically the smallest user profile possible and a subset of any real user profile. Its test result (292 seconds) indicates a lower bound of optimization benefit one should expect from using any profile.

	Number of Procedures Used By						Procedure Distribution by Usage Count		
Date	Bashful	Doc	Grumpy	Sneezy	Happy	*Combined*	1 (*unique*)	2-4 (*subgroup*)	5 (*common*)
10/10/97	4600	4091	4691	5648	6222	7191	20.4%	33.5%	46.2%
10/15/97	4600	*4465	4691	5648	6222	7213	19.5%	33.2%	47.3%
10/22/97	4600	4465	*4947	5648	6222	7239	18.2%	34.0%	47.8%
10/29/97	4600	*4834	*5332	5648	6222	7283	17.0%	31.6%	51.8%
11/03/97	4600	*4990	5332	5648	6222	7288	16.3%	31.5%	52.3%
11/10/97	4600	4990	5332	*5846	6222	7317	16.5%	31.2%	52.2%

* A profile that grew

Table 5. Change in similarity when individual profiles grow larger: `winword.exe` (95), five users

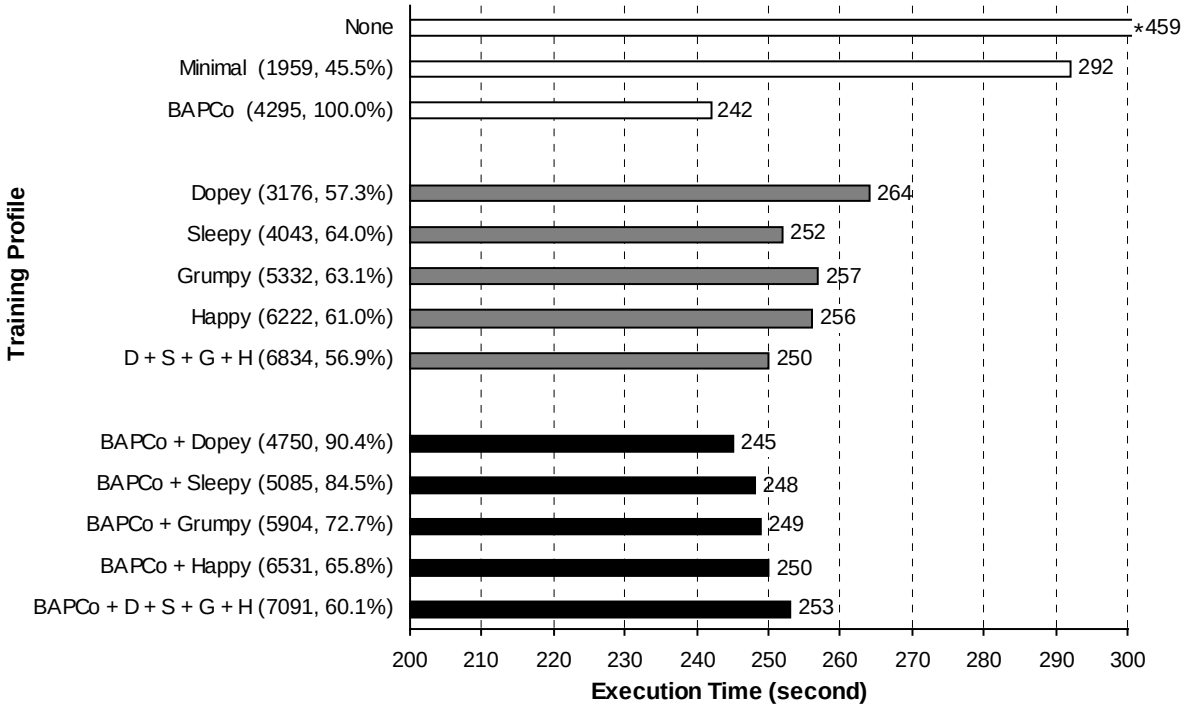

Figure 4. Optimization results for `winword.exe` (95)

Training Profile	The profile used to translate/optimize the program
Execution Time	The execution time of the BAPCo workload using the translated code. Average of three warm runs. Standard deviation is within 2 seconds for all numbers except 6 seconds for "None"
(number, %)	The number of procedures in the training profile, the similarity between the training profile and the BAPCo profile (calculated as the percentage of procedures included in both profiles among all procedures included in either of them)
None	No profile, and therefore no translation/optimization
Minimal profile	The profile generated by starting up `winword.exe` and then exiting immediately
BAPCo profile	The profile generated from one run of the testing workload from BAPCo
+	The combining of profiles
D, S, G, H	Dopey, Sleepy, Grumpy, Happy

We draw several conclusions from Figure 4:

1. We achieve the best performance (242 seconds) when we translate the program by using the test user profile. When using a profile from another user or a group, we see performance that is 1–9% worse (245-264 seconds) but still much better than using the *Minimal* profile.

2. The optimization benefit has a rough trend of increasing with the similarity between the training profile and the testing profile. However, this relation is not monotonic.

3. In the graph, the black bars correspond to profiles from groups that include the test user, while the gray bars correspond to profiles from other users and groups that do not include the test user. With the exception of *BAPCO+Dopey+Sleepy+Grumpy+Happy*, "black bar profiles" provide more effective optimization than "gray bar profiles." This suggests that a group's combined profile is more effective for optimization if the group includes the test user than if not.

4. Among combined profiles that include the test user profile (the black bars), the larger the profile, the less the optimization benefit. This suggests that a combined profile may become less effective for a user in the group when the group is large. A possible explanation is that extra procedures in the translated code increase memory system load and cause sub-optimal code layout. This may also explain why *BAPCO+Dopey+Sleepy+Grumpy+Happy* provides less effective optimization than *Dopey+Sleepy+Grumpy+Happy*.

For this benchmark, user-specific profiling has measurable impact on optimization performance.

3.5.2 Microsoft PowerPoint benchmark

For `powerpnt.exe` (97) from Microsoft PowerPoint 97, we used an automated testing script designed at Digital Equipment Corporation. Similar with BAPCo workloads, it uses Microsoft Visual Test to drive the application. This script was originally designed to test the functionality of PowerPoint. It included some wait time in between tasks. In this sense, it may be closer to a real user than BAPCo workloads, which mostly consist of continuously executed CPU-intensive tasks. On the other hand, some application response time and background activity may be hidden by the wait time, making the throughput measurement of execution time less sensitive to the code quality. Figure 5 shows the results. All performance measurements were done on a 500MHz Alpha computer with 128MB of main memory. PowerPoint 97 was the only application running on the system.

The results show that all the individual and combined profiles are almost equally effective for optimization, with differences on the level of 1%. This implies that for this program and this workload, user-specific profiling does not have significant impact on the performance of FX!32 translation/optimization.

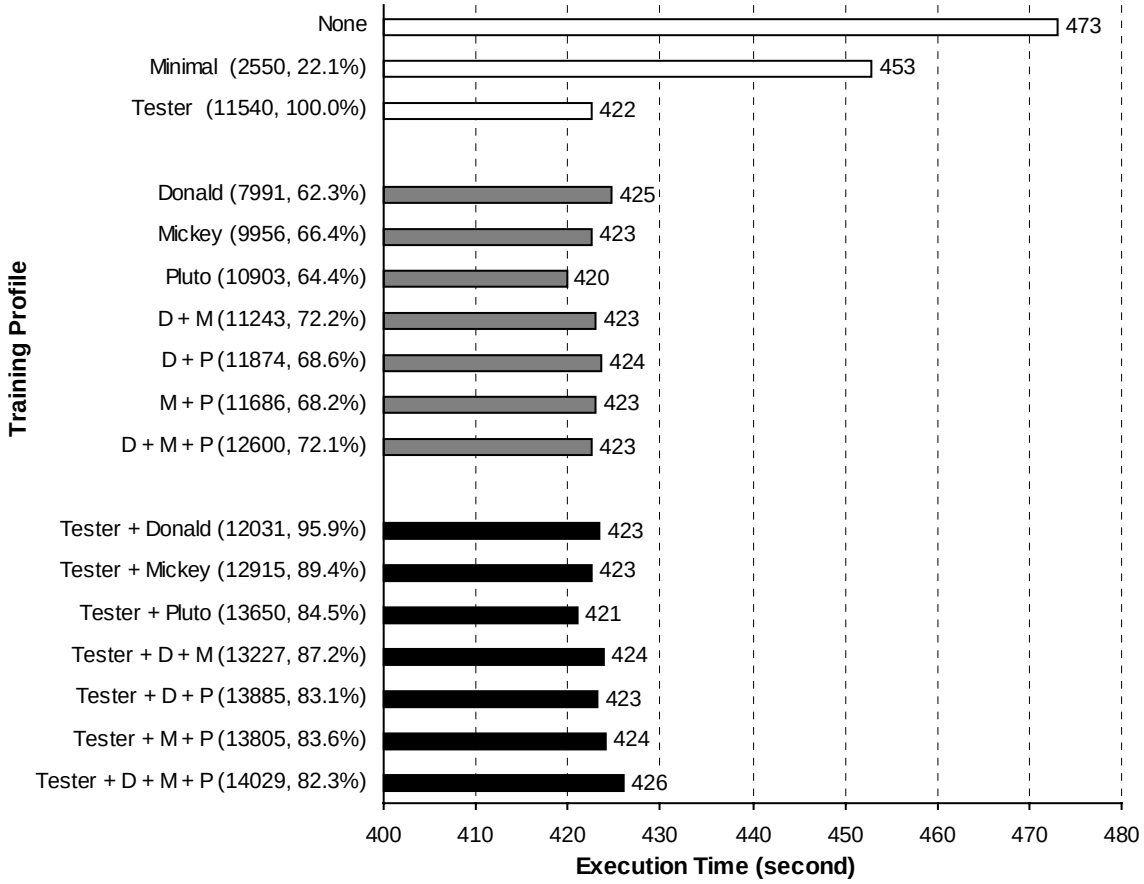

Figure 5. Optimization results for `powerpnt.exe` (97)

Training Profile	The profile used to translate/optimize the program
Execution Time	The execution time of the testing workload using the translated code. Average of three warm runs. Standard deviation is within 2 seconds for all numbers
(number, %)	The number of procedures in the training profile, the similarity between the training profile and the Tester profile (calculated as the percentage of procedures included in both profiles among all procedures included in either of them)
None	No profile, and therefore no translation/optimization
Minimal profile	The profile generated by starting up `powerpnt.exe` and then exiting immediately
Tester profile	The profile generated from one run of the testing script
+	The combining of profiles
D, M, P	Donald, Mickey, Pluto

Results for these two benchmarks indicate that depending on the program and the workload, differences in profiles can have measurable or insignificant impact on optimization performance.

4. Summary

This paper has compared and analyzed FX!32 profiles from different users for a set of Windows NT programs. We discovered that the sets of procedures used by individuals are fairly different. Among all procedures used by a group of users, typically around 50% are used by all users, while 7-24% are used by only one of the users. In most cases, the users have usage patterns different from each other, without anyone using a subset or superset of the procedures another person uses. Frequently executed procedures tend to be used by most individuals. With more use of the program over time, different people's usage patterns may become increasingly similar, but our results suggested that they will never converge. For the FX!32 program translation/optimization, differences in profiles can have impact on optimization performance for some benchmarks. Using profiles from another user or a group may be less effective than using the test user's own profile, but is always effective compared to using no profile or a minimal profile. Overall, we conclude that user-specific profiling is an important factor to consider in profile-based optimization.

Acknowledgement

Many members of the AMT group at Digital Equipment Corporation provided enormous support and important feedback for this project. Special thanks to my advisor at Harvard University, Prof. J. Bradley Chen, for his generous help on improving this paper. Also, thanks to members of the program committee as well as many people at Harvard for their valuable comments.

Availability

The complete set of results for all benchmarks is available in a technical report [WR98] and through the URL http://www.eecs.harvard.edu/~zhwang/NT98/

References

[BAPCo] Business Applications Performance Corporation, http://www.bapco.com/

[CE96] J. B. Chen, Y. Endo, K. Chan, D. Mazieres, A. Dias, M. Seltzer, and M. D. Smith, "The Measured Performance of Personal Computer Operating Systems." In *ACM Transactions on Computer Systems* 14:1, pages 3-40, February 1996.

[CG97] R. Cohn, D. Goodwin, P. G. Lowney, and N. Rubin, "Spike: An Optimizer for Alpha/NT Executables." In *Proceedings of the USENIX Windows NT Workshop*, USENIX Association, pages 17-24, August 1997.

[EW96] Y. Endo, Z. Wang, J. B. Chen, and M. I. Seltzer, "Using Latency to Evaluate Interactive System Performance." In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, USENIX Association, pages 185-199, October 1996.

[FF92] J. Fisher and S. Freudenberger, "Predicting Conditional Branches from Previous Runs of a Program." In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, pages 85-95, October 1992.

[GY96] N. Gloy, C. Young, J. B. Chen, and M. D. Smith, "An Analysis of Dynamic Branch Prediction Schemes on System Workloads." In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, ACM, pages 12-21, May 1996.

[HH97] R. J. Hookway and M. A. Herdeg, "DIGITAL FX!32: Combining Emulation and Binary Translation." In *Digital Technical Journal*, Volume 9, Number 1, Digital Equipment Corporation, pages 3-12, 1997.

[LC98] D. Lee, P. Crowley, J. Baer, T. Anderson, and B. Bershad, "Execution Characteristics of Desktop Applications on Windows NT." To appear in *Proceedings of the 25th International Symposium on Computer Architecture*, IEEE, June 1998.

[PS96] S. Perl and R. Sites, "Studies of Windows NT Performance Using Dynamic Execution Traces." In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, USENIX Association, pages 169-183, October 1996.

[WR98] Z. Wang and N. Rubin, "A Statistical Analysis of User-Specific Profiles." Technical Report TR-09-98, Computer Science Group, Harvard University, July 1998.

[ZW97] X. Zhang, Z. Wang, N. Gloy, J. B. Chen, and M. D. Smith, "System Support for Automated Profiling and Optimization." In *Proceedings of the 16th ACM Symposium of Operating Systems Principles*, ACM, pages 15-26, October 1997.