

Performance of Natural I/O Applications

Stevan Vlaovic and Rich Uhlig

Abstract

In this paper we investigate the properties of Natural I/O applications and compare them to five SPEC95 integer benchmarks. Natural I/O is a type of input into the system; unlike the standard keyboard or some digitized form, the input can be composed of a user's handwriting, speech, gestures, etc. We selected representative applications from the field of speech recognition (*Dragon Systems NaturallySpeaking*), handwriting recognition (*ParaScript Demo*), natural language processing (*PowerTranslator 6.0*), and optical character recognition (*OmniPagePro*). Our characterization environment consists of a fully functional x86 simulator that models all aspects of the hardware required for complete PC system simulation and a released build of Windows NT 4.0 that provides the operating system environment. In addition to the four natural I/O applications, five integer SPEC95 benchmarks for comparison: *go*, *gcc*, *perl*, *compress*, and *vortex*. A wide range of experiments were performed on these applications: memory performance, TLB performance, and branch prediction performance. Although as a rule, the two classes of applications, natural I/O and SPEC95, cannot be directly compared, we can draw some general conclusions. As a group, the natural I/O applications outperformed the SPEC95 benchmarks when it comes to overall branch predictor performance. The performance impact of memory hierarchy on the speech recognition is especially important. Even for large sizes of unified L2 caches, the resulting mpi for all of the speech recognition workloads is problematic.

1.0 Introduction

Lately, there is widespread interest in performance evaluation of computer systems. This has come with the increasing complexity and associated cost of development. It is beneficial to be able to project the performance of given applications on new, as yet unimplemented architectures.

There are certain applications that are used to develop systems, and there are those applications that will actually be used on the target architecture. The most common developmental applications are the SPEC benchmarks. These are widely available, are derived from "real" applications and stress some aspects of the hardware. The target applications are difficult to discern, but computer manufacturers usually have a handful of target workloads that they are interested in studying. In this study, we will be looking at some "natural I/O" applications.

1.1 Natural I/O

Natural I/O refers to input that can be composed of a user's handwriting, speech, gestures, and other types of input. These types of applications have not as yet entered the mainstream, as they are generally either performance or accuracy limited.

We selected representative applications from the field of speech recognition, handwriting recognition, natural language processing, and optical character recognition. The choice of applications represents a good cross section of the available applications.

1.2 Accuracy versus Speed

All of these applications have certain characteristics in common: their algorithms must make accuracy and speed tradeoffs. To increase the accuracy, it is almost always to the detriment of speed. The converse is generally also true. Memory requirements can be leveraged to ameliorate the effects of accuracy and speed, but in practice the amount of memory required to make a significant impact is unrealistic.

In section 5, we extend our basic experiments to see the impact of accuracy, in addition to making the input data set more difficult.

1.3 Contributions

This work reveals the architectural impact of applications whose interaction with the hardware has not been studied previously. These applications are of interest as they represent part of the future of computing. Understanding the

architectural requirements of natural I/O applications will make more cost effective implementations possible.

1.4 Related Work

Our work is similar to D. Lee's work on Win32 applications[5]. In his work, binary instrumentation was used to do architectural studies of popular Win32 applications. The instruction set breakdown, the control flow characteristics, cache behavior, translation lookaside buffer (TLB) behavior, and branch predictability are studied and compared to five of the integer SPEC95 benchmarks. The purpose of their study was to test their selected desktop applications and compare the applications to standard Unix derived SPEC95 benchmarks.

While our applications can be characterized as Win32 applications, this would not be the defining characteristic. Additionally, our traces contain all associated code, from kernel drivers up to the application.

2.0 Methodology

Our approach uses commercial applications as workloads for our experiments. To this end, we have chosen a certain class of applications, classified as "natural I/O".

To study these types of applications, there are a number of different approaches that can be used. Firstly, these applications can be run on an existing machine with hardware probes to extract the instruction and data addresses at the memory bus. The problem with this approach is that using today's machines, the buffers on these hardware monitors tends to fill up rather quickly, requiring a way to stall the machine under study. A second approach is to use binary instrumentation to insert code at well defined points in the binaries. The instrumentation code can be placed anywhere in the binary, allowing the user to track certain events of interest, addresses of loads and stores, as well as branch information. The limitations of this type of approach are that it changes the addresses seen by the CPU, as well as the time dilation associated with external events.

Our approach is to use a full functional simulator thus enabling us to capture both user and operating system effects. This method also maintains the order of addresses seen by the CPU.

2.1 PC Simulator

We have developed a PC simulator that models not just the CPU, but the entire platform in enough detail to support the execution of a complete operating system and the applications that run on it. Currently, we are using out-of-the-box Windows NT 4.0 (Build 1381) as the operating system for our Virtual PC. This simulator runs as a user-level process on a standard PC, modeling the platform components completely in software.

Since a standard Windows NT can execute on the functional simulator, we can study commercial applications completely. This approach allows access to all operating system events, in addition to the standard instruction, data, and branch traces.

The limitation of this method is that the simulator is functional, and not cycle accurate. This poses problems for detailed timing analysis, but VPC can be used as a front end for more detailed simulations.

2.2 Branch Predictor

For our branch prediction studies we used the branch prediction mechanism used in SimpleScalar 3.0[1]. This predictor is intended to be used to model existing branch predictors, while providing a wide variety of different tuning parameters.

This particular predictor is capable of modeling GAg, GAp, PAg, PAp, gshare, and combinational, in addition to the standard perfect, bimodal, taken, and not-taken predictors []. The taken and not-taken predictors simply assume either always taken or not taken, respectively. The sizes of all the tables can also be specified, as can the history width for the two level predictors. The branch prediction mechanism is discussed in more detail in Section 4.3. The predictors that are used in this study are the combined bimodal (2-bit) and a gshare predictor.

This work focuses on non-timing related experiments, since all of our experiments are based on traces. With these traces we can do workload, memory, and branch prediction studies. The actual experiments are discussed in Section 4.

3.0 Application Characteristics

In this section we highlight the applications chosen for our study. In addition to the four natural I/O applications, there will be five integer SPEC95 benchmarks for comparison: *go*, *gcc*, *perl*, *compress*, and *vortex*. The SPEC95 benchmarks are run in the exact same environment (Windows NT) as the natural I/O workloads. We will compare and contrast these two sets of applications and their characteristics from a high level. This will include size of executable, length of trace, and branch classification breakdown. This will highlight the main differences between natural I/O applications and SPEC95, and hopefully provide clues to architectural results found in later sections.

3.1 Natural I/O Applications

There are four natural I/O applications that we have chosen. These represent a variety of different software packages that have a certain level of maturity. The fields are handwriting recognition, speech recognition, natural language processing, and optical character recognition.

3.2 Handwriting Recognition

Handwriting recognition has recently come to the forefront of computer software development with the innovation of hand held devices. Generally, pen based input was preferable to standard keyboard type entry, given the physical constraints.

There are many steps associated with handwriting recognition. Statistical classifiers, neural networks, structural classifiers, and graph approaches are common methods used for handwriting recognition[9,10,11]. In general, the algorithm is as follows:

- preprocessing
- segmentation
- normalization
- feature extraction
- stroke classification
- letter classification
- word classification

In the preprocessing step, the data is low pass filtered and differentiated, to remove noise and/or stray marks. In the segmentation portion, the input image is skeletonized, and

the image skeleton is interpreted as a graph. Nodes in the graph are the following:

- breaking points of a pen trace
- pen trace self crossing points
- points of strong curvature
- inflection points

Edges in the graph are sections in between two nodes. Normalization involves evening out the slant and size of the graph. Stroke classification breaks down the graph into its smallest components; then they can be matched against a character “dictionary” in the letter classification phase. Finally, word classification puts these recognized letters together to form words which are then matched with a word dictionary. The complexity is based on the number of strokes in the stroke dictionary, the average number of letter hypotheses per stroke, and the size of the word dictionary.

Our representative application was an address recognizing demo from *ParaScript LLC*. *ParaScript* is the company that supplies the technology to Mitek and NCR for their check processing systems. Since their product came complete with a run-time demo, with sample inputs, it was straightforward to collect the sample traces.

3.3 Speech Recognition

With faster processors, speech recognition has been gradually gaining a foothold as a desirable, functional application. Speech recognition software can either be speaker dependent or independent. Speaker dependent systems are much easier to develop and are more accurate, but not as flexible as speaker independent systems. A speaker independent system is developed to operate for any speaker of a particular type (e.g. American English). These systems are the most difficult to develop, the most expensive, and accuracy is lower than speaker dependent systems. A continuous speech recognition system does not require pauses at the end of each word, whereas an isolated word system does. The continuous type generally recognizes groups of words (and sounds) rather than just using one word for identification.

The basic algorithm generally starts with digital sampling of the input speech pattern. Typical representations of signals for speech recognition are sampled at rates between 8

and 16 kHz with 8 to 16 bits of resolution. This digital signal may then be subjected to conditioning. For instance, bandpass filtering can be used for attenuating the parts of the spectrum that are corrupted with noise. After the signal has been suitably conditioned, it may be used as an input to an algorithm for parameter extraction (i.e. how to represent the signal in numerical form).

A basic parameter is the frequency representation of a signal in various time frames. This representation is equivalent to a spectrogram in numerical form. Typically, this is computed using the fast Fourier transform (FFT). However, the most popular parameter, linear predictive coding (LPC) coefficients, are an efficient way of representing the speech signal. In LPC, a parametric representation of speech is created by using past values of the signal to create future values. This formation of parameters is referred to as acoustic signal processing.

The next stage is recognition of phonemes, groups of phonemes and words. A phoneme is a speech sound that has only one meaning, regardless of how it is pronounced. The most common method of recognizing phonemes is Hidden Markov Modeling (HMMs).

HMMs are a useful method for modeling both the stationary and transient properties of a signal. They are especially good for speech since some speech sounds are sustained, such as vowels, while others are ephemeral, such as stop constants, and the transitions between them are short periods of rapid change. The basic structure of an HMM is a set of states with transitions between each state. Like an ordinary Markov chain, for each transition from a given state a probability of taking that transition is assigned, with the sum of all transitions from a state equaling one. At each state, a symbol is outputted, which is also determined probabilistically. Therefore, each state has a probability distribution of the possible output symbols. These models are referred to as "hidden" since the sequence of states is not directly observable: it can only be probabilistically deduced from the sequence of output symbols.

Essentially, this approach uses a general pattern matching scheme that requires large amounts of training data to produce good word models. It then matches spoken words against known word models, selecting the one with the highest probability of a match. A good introduction to HMMs is contained in [7,4], whereas more advanced work can be found in [3].

The speech recognition package that we used for our study is *Dragon Systems Naturally Speaking Preferred Edition*. According to the manual, this package is capable of recognizing 160 words per minute, 98% accuracy, and has an active (in main memory) vocabulary of 42,000 words. The total vocabulary size is 230,000 words. *NaturallySpeaking* had the desirable trait of being able to process a .wav file, such that we didn't have to provide the sound card capability for our simulation platform. For our input set, we trained the system, which requires about 30 minutes, then recorded a voice, and transcribed it to *Dragon NaturallySpeaking*.

3.4 Natural Language Translation

Natural language translation has become more useful with the increased globalization of the economy, as well as the rapid expansion of the World Wide Web. Most natural language processing systems employ a two level morphology, (i.e. defined as the system of word-forming elements and processes in a language). This helps in generic definition of different rules for case, number, gender, umlaut, and different morphological paradigms. The nature of language transformation is defined by the following:

- two-level rule set
- set of affixes and their allowed combinations
- the lexicon

The development and run time performance depends on the characteristics of the language being described and the number of base forms in the lexicon; in another words, the constrained versions of the above. For example, the English language is relatively simple, and an optimization may be to remove the need for a two level rule set.

PowerTranslator 6.0 was our choice as a representative for a natural language processing system. This package had the capability of translating to and from four different languages, with an advertised rate of 17 words per minute on a Pentium processor. For our input, we translated a page of text from English to Spanish.

3.5 Optical Character Recognition

Although optical character recognition (OCR) has been around for some time, it still provided interesting workload characteristics. The five step process that goes into a OCR product are the following:

- identification of text and images
- character recognition
- word identification/recognition
- correction
- formatting of the output

The first step separates into blocks what the software will try to recognize and what it will not. The second step usually involves feature extraction, where it analyzes a shape and compares it against a set of rules for each character font combination. Comparing the character strings found in the previous step to dictionaries helps to identify words. The correction stage allows the user to manually change the unrecognized words from the previous stage.

There are a number of different accuracy versus speed tradeoffs with OCR: scan resolution, hardware variables, detection algorithms, size of font/word dictionaries, and type and format of the original are just a few.

OmniPagePro 8.0 was chosen as our representative OCR application. For our application, we used a sample `.tif` file as our input. This image was of reasonable quality, but only had a few different type fonts, with no imbedded graphics.

Below is a table that contains the applications and their characteristics.

TABLE 1.

Application	Instructions Executed (millions)	Size of Executable (KB)
gcc	440	1199
go	512	317
perl	344	115
compress	644	342
vortex	315	536
ocr	487	749
speech	434	1227
translate	451	2749
handwriting	329	192

4.0 Experiments

4.1 TLB

The translation lookaside buffer, or TLB, can have a significant impact on the performance of the processor. The TLB's function is to keep recent virtual to physical page mappings in a buffer on chip. If a significant amount of translations miss in the TLB, then this could have an adverse affect on performance.

In this study, we look at separate instruction and data TLBs, as well as a unified TLB. We varied the number of entries from 16 to 256, and the associativity from four way to fully associative. Our metric for performance is the number of misses per instruction (mpi). Looking at Figure 1, for a 16 entry fully-associative data TLB, *go* has, on average, 3.1 data TLB misses for every 100 instructions.

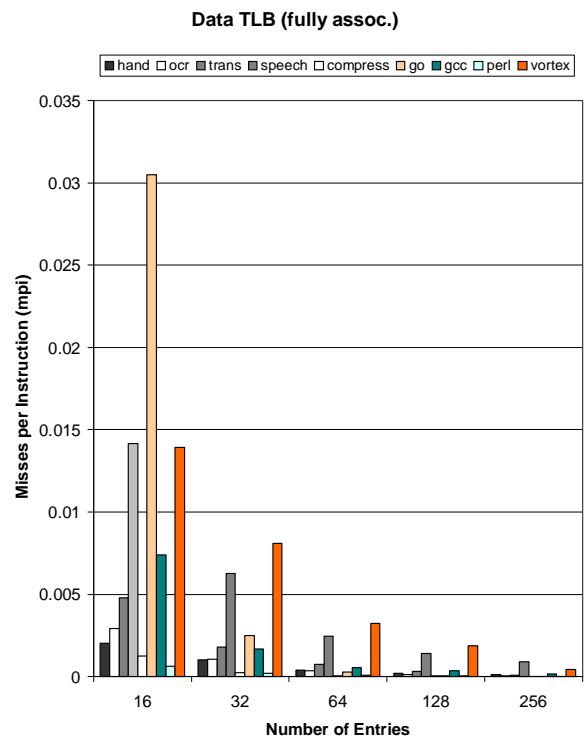


FIGURE 1. Fully associative data TLB

Looking at Figure 1 for the data TLB, the natural I/O and SPEC95 benchmarks behave similarly. The speech recognition application is less sensitive to increasing size and

associativity. The reason for this is discussed later in Section 4.4.

For small instruction TLB sizes, the natural language translation workload performs worse than all of the other applications, about 4 to 5 times worse for a 16 entry instruction TLB. For 64 entries and above, however, most of the instruction page table translations fit into the TLB, and the performance is comparable to the other workloads, as seen in Figure 2. *PowerTranslator 6.0* does have the largest executable size, at 2.7 MB, more than twice as large as the next largest application.

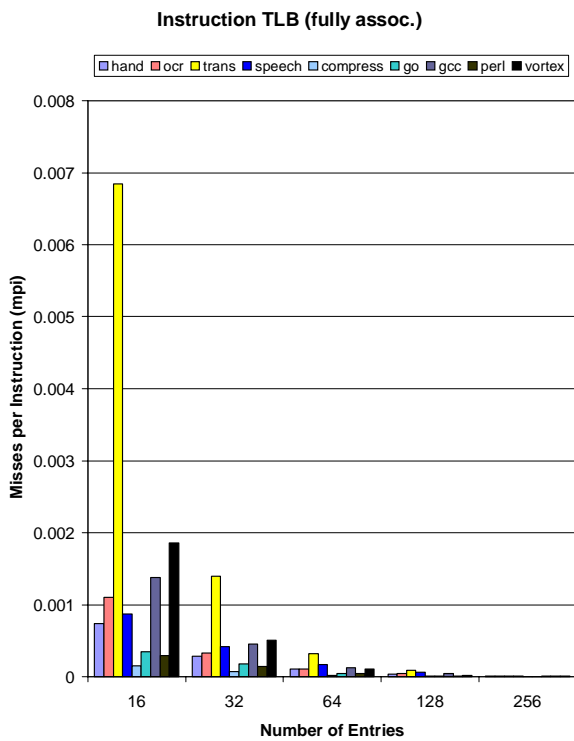


FIGURE 2. Fully associative instruction TLB

The performance of the natural I/O applications tends to be slightly worse on smaller TLB sizes, but this effect is diminished as the TLB sizes become larger. As an example of a current processor, the Alpha 21264 has separate 128 entry instruction and data TLBs which should minimize TLB performance penalties on these applications.

4.2 Cache

For our experiments, we have a split first-level cache and a unified second level cache, with a 32 byte line size for all caches. For the first-level cache, we varied the size from 4 KB to 64 KB, with associativity ranging from direct mapped to eight-way set associative. The L2 cache is varied from 128 KB to 1MB, with associativity ranging from direct mapped to eight-way set associative. The L2 cache experiments are conducted with separate 4-way 16 KB instruction and data caches. The performance metric is the number of misses per instruction.

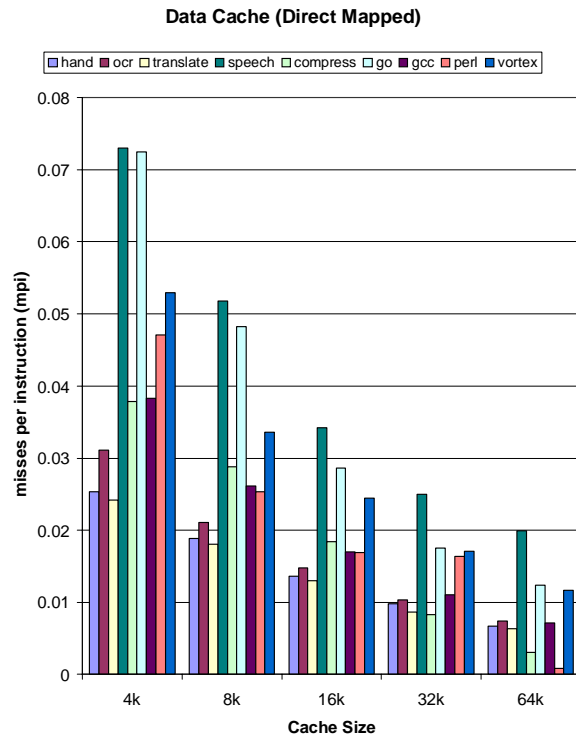


FIGURE 3. Direct mapped L1 data cache

For small L1 data cache sizes, the natural I/O applications are indistinguishable from the selected SPEC95 benchmarks, as seen in Figure 3. The speech recognition application stands out as the most demanding with respect to size and associativity. Even for a 64KB direct mapped cache, the speech recognition workload has an average of 2 misses per every 100 instructions. The reason for this is the size of its dictionary that is kept in memory. In Section 5.1 we fully train the speech recognition application,

hence giving it a larger user dictionary size, in order to test the impact of increased dictionary size on architectural performance.

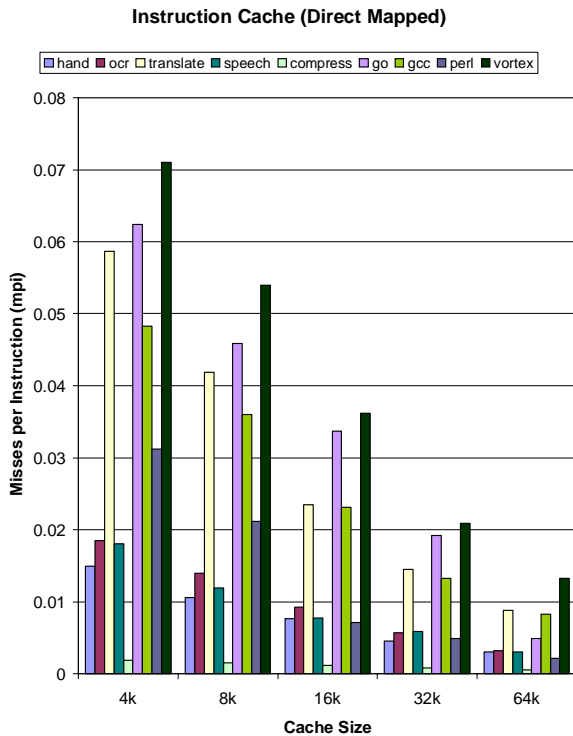


FIGURE 4. Direct mapped L1 instruction cache

Figure 4 shows the instruction cache performance of the SPEC95 benchmarks and the natural I/O applications. The natural language translation application performs the most poorly of the natural I/O applications, while vortex is the poorest performing SPEC95 benchmark. In fact, vortex is the poorest performing of all the applications tested. The two level morphology is to blame for the poor performing instruction cache performance of our natural language translation program. Since the algorithm has to go through two transformations to get the correct translation, the size of the working set is larger than a straightforward pattern matching scheme. But as the L1 instruction cache gets larger, the natural language translation program no longer stands out.

As seen in Figure 5, the results for the unified L2 seem to reveal some more interesting insights, especially in regards to the speech recognition application. The speech

recognition program misses a 1 MB direct mapped unified L2 cache on an average 8 times out of every 1000 instructions.

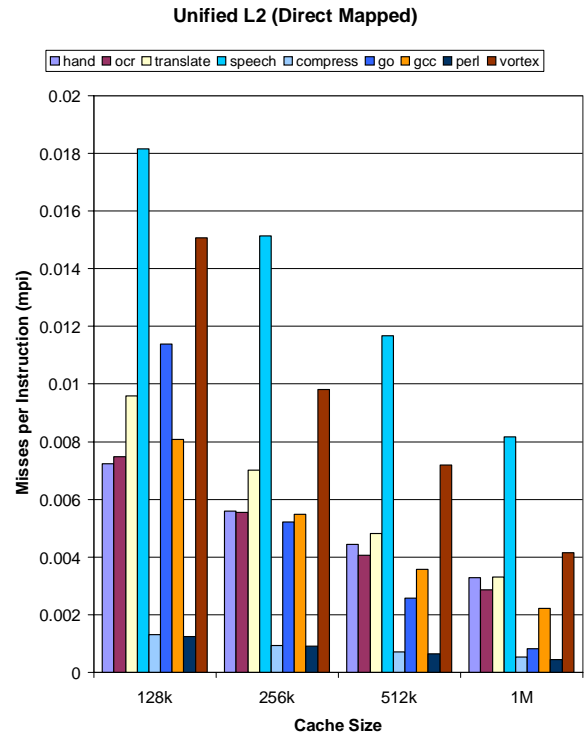


FIGURE 5. Direct mapped L2 unified cache

In fact, most of the natural I/O applications perform worse than the SPEC95 benchmarks especially at larger sizes. The natural I/O application suite has a 200% larger mpi on average than the SPEC95 benchmarks.

4.3 Branch Prediction

In this section we show the results for some preliminary

TABLE 2. Branch Breakdown

App.	% CB	% jmp	% ind. jmp	% dir. call	% ind call	% ret
gcc	79.1(24.7)	4.5	3.3	6.2	0.4	6.6
go	84.0(36.4)	6.3	2.5	4.1	0.6	4.7
perl	66.7(24.6)	11.0	6.9	7.4	0.3	7.7
comp	78.0(54.0)	13.7	0.0	4.0	0.0	4.1
vortex	80.3(10.2)	3.1	0.4	7.5	0.6	8.0
ocr	86.8(67.6)	6.2	0.4	2.3	1.0	3.3
speech	69.0(47.1)	8.0	2.8	7.1	3.0	10.1
translate	49.2(52.1)	11.8	1.1	15.0	3.9	19.0
hand	69.3(50.3)	9.9	0.6	3.1	7.0	10.1

branch prediction experiments. We examine the impact of these natural I/O applications on some common branch prediction schemes, and then compare the results to the SPEC95 benchmarks.

Table 2 gives a breakdown of the branch types. The first column gives the percent of all control instructions that are conditional branches (%CB). The number in parenthesis in the same column is the percent of these conditional branches that are actually taken. The next column gives the percent of all control instructions that are unconditional branches (jumps). The rest of the columns follow in a similar manner. In general, the natural I/O applications have a smaller percentage of control instructions that are conditional branches, and a higher percentage of calls, both direct and indirect.

The first predictor is a simple bimodal predictor that uses a two-bit saturating counter per branch[1]. The second predictor is a gshare predictor that uses a global shared history patterns xor'ed with the PC to get an index into a table of two-bit saturating counters. For the bimodal predictor we vary size from 512 entries up to 32 K entries, while for the gshare predictor the size is varied from 1 K to 32 K entries.

The numbers in Table 3 show the branch prediction performance of the natural I/O applications compared against the

TABLE 3. 1024 entry Bimodal predictor (512 entry BTB)

Application	% correct direction	BTB hit %	Addr. %	num. branches (million)
gcc	85.0	48.3	32.9	30.8
go	83.1	59.6	42.3	28.7
perl	89.0	38.1	43.7	23.0
compress	81.9	77.5	68.5	31.0
vortex	93.2	20.8	22.5	21.0
ocr	91.8	87.9	71.8	28.8
speech	88.8	72.9	58.8	17.0
translate	91.5	71.2	71.2	31.2
handwriting	91.3	75.2	63.5	24.1

SPEC95 benchmarks. This table shows results for a 1K bimodal predictor with a 4-way 512 entry branch target buffer (BTB). The first column contains the percentage of time this predictor predicts the correct direction of the branch. The BTB hit rate is the next column. The percent of the time that branch predictor comes up with the correct address on a control flow change is contained in the third column. The last column is the number of control flow instructions in each trace.

In general, it seems as though the natural I/O applications have better predictability than the SPEC95 benchmarks. On average, the first three performance columns signify that the natural I/O applications should perform better than SPEC95. Some of these results may differ slightly from those reported in the literature, since our traces contain both application and operating system code; hence, there cannot be a direct comparison to these other studies.

For a 32 K entry gshare predictor, we have the results as shown in Table 4. This table also illustrates the BTB sensi-

TABLE 4. 32 K entry Gshare predictor (2 K entry BTB)

Application	% correct direction	Addr. %	BTB hit %
gcc	91.3	35.8	57.7
go	90.4	43.4	70.6
perl	93.7	43.8	41.2
compress	89.7	64.2	77.7
vortex	98.1	26.7	26.9

TABLE 4. 32 K entry Gshare predictor (2 K entry BTB)

Application	% correct direction	Addr. %	BTB hit %
ocr	95.3	71.1	89.9
speech	94.6	62.3	76.9
translate	96.0	74.1	78.7
handwriting	94.7	64.1	78.8

tivity. The differences in the two workloads becomes even more pronounced with the better predictor. With the exception of vortex, all of the natural I/O applications have better direction prediction percentages. Similarly, with the exception of compress, all of the natural I/O applications have better BTB hit percentages and correspondingly better address prediction percentages.

4.4 Miss Classification

These experiments involved breaking down the cache misses into classifications. The classification breaks down misses into the standard three, compulsory, conflict, and capacity[], but splits up conflict misses into mapping and replacement[12]. Compulsory misses are cold start misses that would miss no matter what type of cache; capacity misses are those that are due to the actual size of the cache; mapping are those that occur due to mapping to the same set; and replacement misses are those that are due to a non-optimal replacement policy. This breakdown gives complete coverage of misses, while at the same time allowing for an explanation for each type of miss, helping to identify the root cause (and possible solution).

In Figure 6 the graph depicting this breakdown for instruction misses is highlighted. The left y-axis represents the percentage breakdown and applies to the stacked bar graphs. The *lru_mpi* line is the number of misses per instruction for a 16 KB, four way, 32 byte line size cache and it uses the right y-axis. The higher the mpi, the more important the miss breakdown. The two applications with the highest mpi, *go* and *speech* recognition, have markedly different miss breakdowns. *Speech* recognition has more capacity misses with nearly half of all misses being attributed to capacity of the cache. Another interesting facet of Figure 6 is the large amount of conflict misses contained

in the *perl* benchmark. Almost 60% of the misses in *perl* are conflict misses.

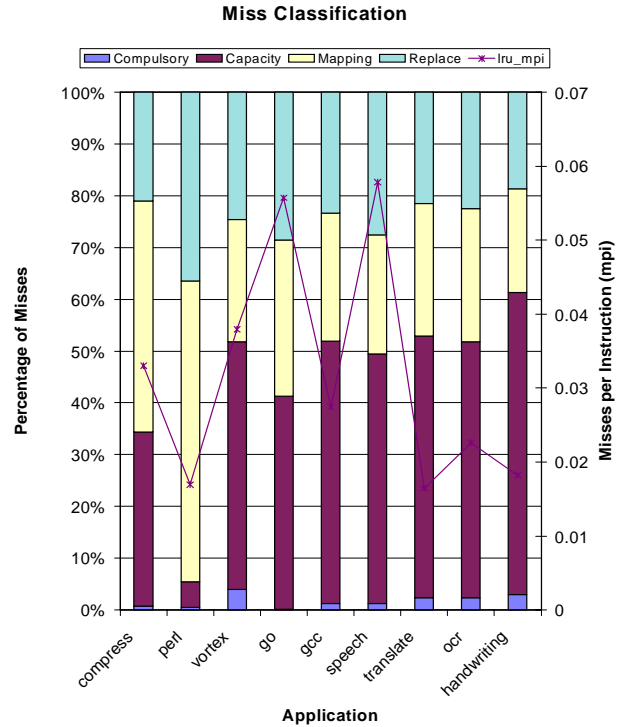


FIGURE 6. Miss Classification breakdown

In this experiment we also get a memory footprint for each application. The size of the memory footprint is the mea-

TABLE 5. Code and Data Footprint

Applications	Code (MB)	Data (MB)
gcc	1.38	4.58
go	0.89	1.55
perl	0.79	0.98
compress	0.72	5.43
vortex	0.95	14.7
ocr	1.64	8.19
speech	0.98	9.83
translate	1.18	5.48
handwriting	1.66	6.91

sure of all the cache lines touched in an infinite cache, with a cache line size of 32 bytes. In general, the SPEC95

benchmarks have smaller data and code footprints. Since all of the natural I/O applications require a search, it is reasonable to assume a larger memory footprint. However, the total cache lines touched is of less importance than the time distribution of the references to these cache lines. Vortex has a large memory footprint (14.7 MB), when compared to go (1.55 MB), yet go has a higher mpi, as illustrated in Figure 6.

Table 6 displays the reference stream in greater detail. The first column contains the percentage of all code references that are to the kernel space. The next column, % r, contains the fraction of the data set that is read, while % w has the fraction of the set written. The reason that these do not total one hundred percent is a memory location can be both written to and read from. The next two columns have the fraction of the working set that is reused by reads and writes, respectively, while the last column holds data that is only written and never read from.

The one application that stands out is speech recognition. It has the highest percentage of reads, at 90.7%, and by far the lowest fraction of the set reused by writes, at 43.2%. This is more than 20% lower than the next lowest application. Speech recognition also has the smallest fraction of data that is written and never read, at 9.32%. Clearly, speech recognition is a data intensive application, with an emphasis on processing the given data.

TABLE 6. Data Reference Breakdown

Application	% sys call	% r	% w	% re/r	% re/w	w/o
gcc	3.78	82.8	80.9	79.2	77.6	17.2
go	3.49	66.5	68.8	63.0	64.9	33.5
perl	4.72	71.7	64.6	66.6	64.6	28.5
compress	2.16	16.5	92.3	15.6	91.4	83.5
vortex	5.85	37.6	84.5	36.3	81.7	62.4
ocr	8.97	70.2	80.3	68.1	78.8	29.8
speech	5.82	90.7	44.9	84.1	43.2	9.32
translate	6.01	80.0	68.4	76.5	62.9	20.0
handwriting	5.78	81.5	70.3	79.7	70.3	18.5

5.0 Extended Experiments

In this section, we extend the experiments by either making the data set more complex or making the dictionary more complex. The extensions were performed on the speech recognition application, the optical character recognition application, and the natural language translation program.

5.1 Speech Recognition

The speech recognition application used for this study, *Dragon Systems NaturallySpeaking Preferred*, uses “User Speech Files” in order to facilitate speech recognition. The previous run of the speech recognition used only the basic training model of 30 minutes (one passage). *NaturallySpeaking Preferred* can be trained with 10 passages, with 3 short and 3 long. (The base test used one of the long passages.) In the users’ manual, one of the features is that “with regular use and training you can achieve a high level of recognition accuracy”. The extended test for speech recognition involved the complete training of *NaturallySpeaking Preferred* and then running the same input .wav file as in the previous experiment.

Dragon NaturallySpeaking also allows the user to trade off accuracy for speed. The previous speech experiments involved maximum speed; a new experiment was devised using maximum accuracy. This test was performed in conjunction with a fully trained system.

5.1.1 Memory Hierarchy

The results for the L1 data cache are shown in Figure 7. One would expect the fully trained system to perform worse than the base system. This is, however, not the case. In fact, although both perform similarly for the L1 data cache, the fully trained system actually performs slightly better. In fact, the accurate speech recognition system actually performs the best for smaller L1 data sizes. However, when the size reaches 32KB and larger, all of the systems perform similarly. The nature of HMMs dictates that

the dictionary has better word models, realizing a higher match probability more quickly.

(both systems) and more thorough search (accurate speech system) is being performed.[13]

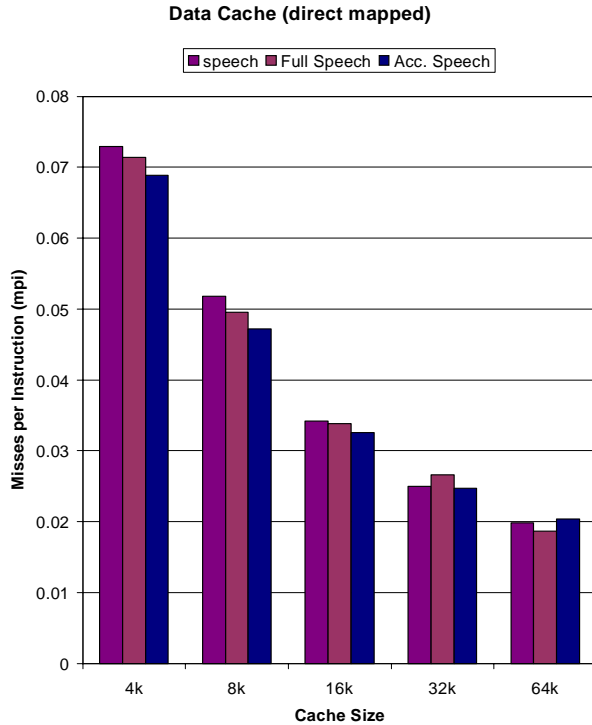


FIGURE 7. Speech direct mapped L1 data cache

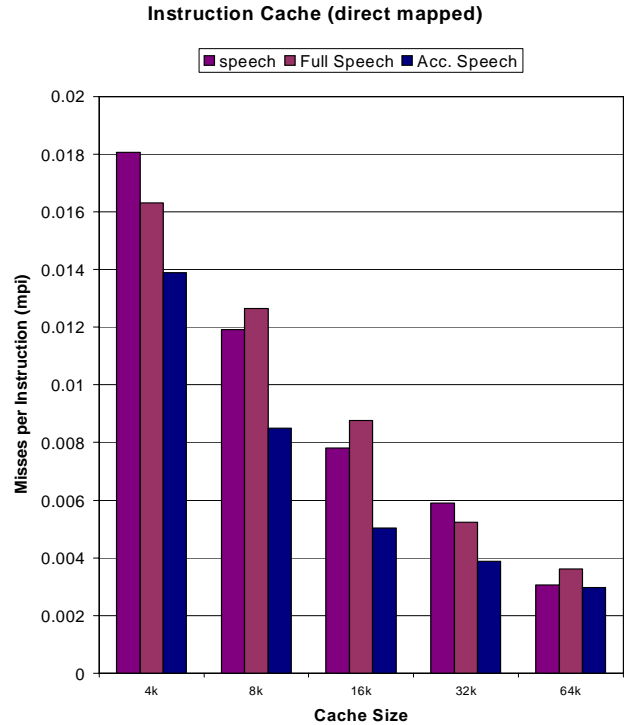


FIGURE 8. Speech direct mapped L1 instruction cache

As illustrated in Figure 8, the results for the L1 instruction cache differ more significantly. Although the base system performs worse for smaller cache sizes, the fully-trained system is less sensitive to cache size and associativity. These results are shown in Section 5.1.3. The fully-trained speech system and the accurate speech system perform better for smaller cache sizes. The fully trained system has a better performing HMM model, thereby increasing instruction reuse. Additionally, the application is spending more time in the search process, as a larger dictionary

Results for the unified L2 cache are unexpected and do not quite follow from the previous results as seen in Figure 9. For small cache sizes the base system performs slightly worse than the fully trained and accurate ones, but for larger sizes, the difference between the base and fully trained system is negligible. The performance of the accurate speech system is significantly worse (38.5% for a 1M direct mapped cache) for larger L2 cache sizes, though. This illustrates the performance accuracy tradeoff that we mentioned earlier.

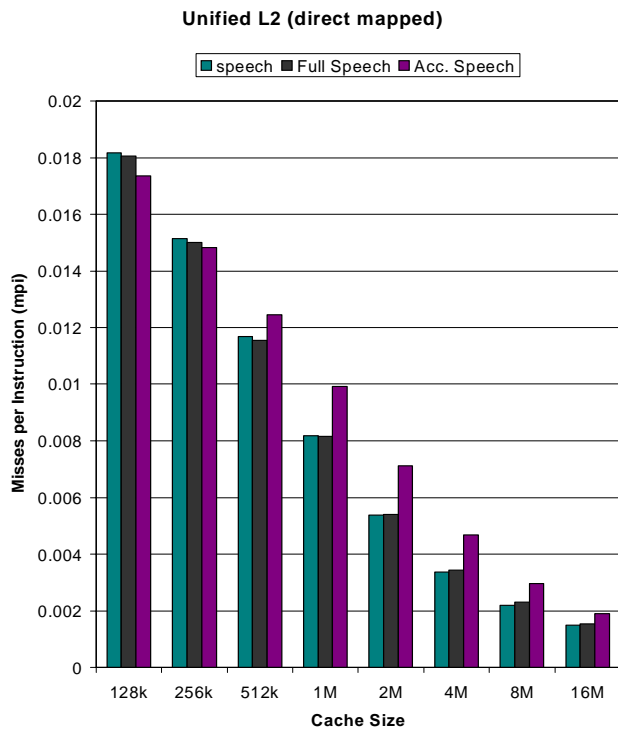


FIGURE 9. Speech direct mapped L2 unified cache

5.1.2 TLB

For all data TLB sizes except 16 entries, the performance of the fully trained and base system are nearly identical as shown in Figure 10. The accurate speech recognition system has a much higher mpi, especially for TLB sizes less than 64 entries. Since the HMM uses a probabilistic modeling process and a more accurate system it requires a higher probability to match, it is natural to expect a higher data TLB miss rate. For a corresponding phoneme match, the data reference stream is actually longer without the added benefit of locality.

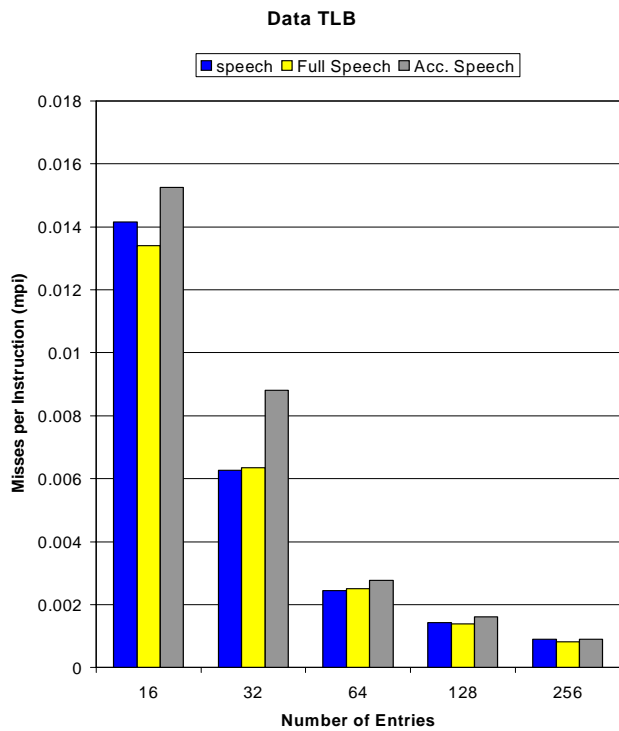


FIGURE 10. Speech fully associative data TLB

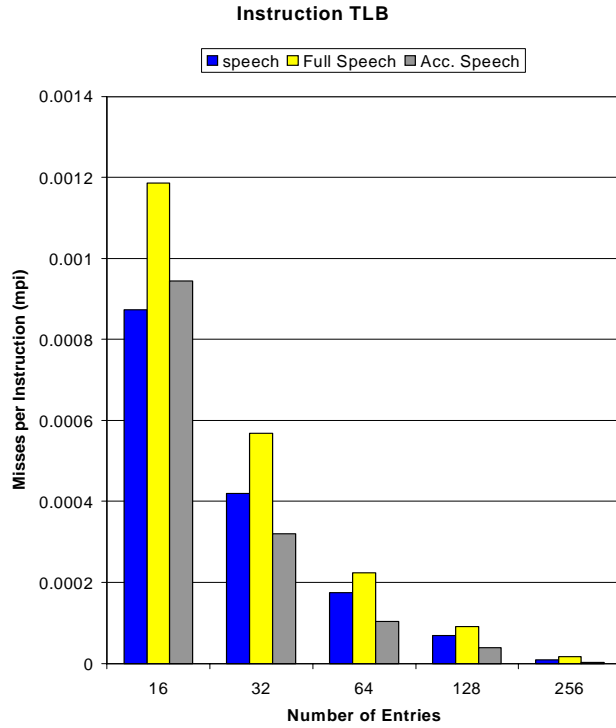


FIGURE 11. Speech fully associative instruction TLB

The results for the instruction TLB bear out the conclusions drawn in the previous section. For a 32 entry TLB, the full speech application performs about 38% worse than the base case. This supports the larger working set size put forth in the previous section. The accurate speech system has similar performance to the base system. For all but 16 entry instruction TLBs, the accurate speech recognition system outperforms the two other systems. This behavior may stem from control loop for the search - the more accurate system spends more time in the control loop searching for the most accurate match.

5.1.3 Miss Classification

As seen in Figure 12, the percent breakdown of misses is similar across all three of these applications. The fully trained and base system have nearly identical cache miss breakdowns. The accurate speech recognition system has a

slightly higher percentage of capacity misses and cold start misses, with less misses being attributed to conflict.

TABLE 7. Code and Data Footprint

Applications	Data (MB)	Code (MB)
base	9.83	0.98
full dictionary	11.0	1.13
accurate	9.83	0.46

Table 7 above is similar to the one described in Section 4.4. It is natural that the both memory footprints increase from the base to the full dictionary system, but it decreases from the full dictionary to the accurate system. The algorithm generating the same amount of references, yet fewer of the phonemes are being recognized per reference. Far fewer phoneme's are being recognized by the accurate system than are being recognized by the other two systems.

TABLE 8. Data Reference Breakdown

Application	% sys call	% r	% w	% re/r	% re/w	w/o
base	5.82	90.7	44.9	84.1	43.2	9.32
full dict.	5.59	92.1	45.3	85.8	43.6	7.87
accurate	4.17	89.2	44.0	81.6	42.0	10.8

As would be expected, the percentage of operating system references will continue to decrease from the full dictionary to the accurate system. This is once again due to the longer time it takes to recognize each phoneme. Overall,

there is less data reuse when going from from less accurate to more accurate.

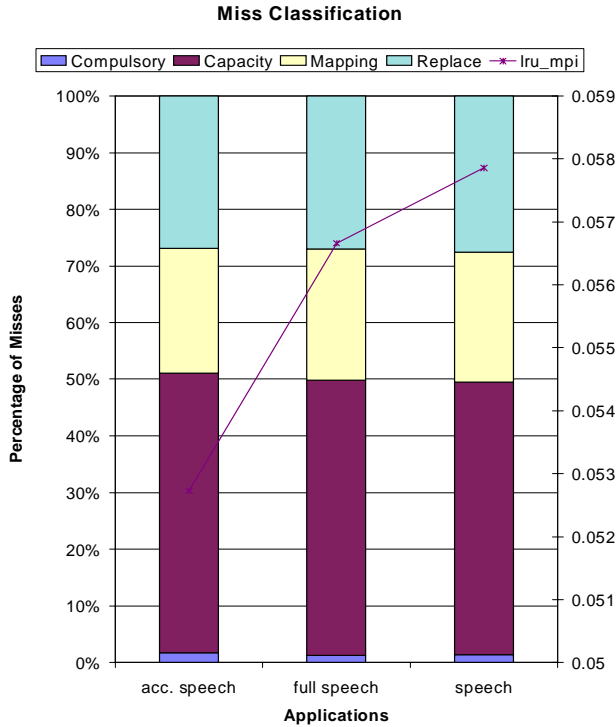


FIGURE 12. Speech miss classification breakdown

5.1.4 Branch Prediction

Table 9 shows the branch breakdown for the base speech recognition, the fully trained speech recognition, and the accurate speech recognition applications. Fully training

TABLE 9. Branch Breakdown

App.	% CB	% jmp	% ind. jmp	% dir. call	% ind call	% ret
base	69.0(47.1)	8.0	2.8	7.1	3.0	10.1
full dict.	67.9(53.9)	6.7	3.3	8.4	2.7	11.0
accurate	71.5(52.6)	6.7	2.2	6.9	2.9	9.8

the system tends to marginally reduce the number of jumps while increasing the number of calls. The better performing HMMs of the fully trained system allow for the recognition of more phonemes in a given period of time. By making the system more accurate, the number of

calls is reduced significantly. The increase in probability

TABLE 10. 1024 entry Bimodal predictor (512 entry BTB)

Application	% correct direction	BTB hit %	Addr. %	num. branches (million)
speech	88.8	72.9	58.8	17.0
full dict.	87.9	77.9	64.1	19.1
accurate	89.2	79.8	63.5	22.1

required to make a match using HMMs is responsible. The increase in direction predictability is due to increasing number of calls and conditional branches for the fully trained and accurate systems, respectively, as shown in Table 9. Table 11 illustrates the effect of a better predictor

TABLE 11. 32 K entry Gshare predictor (2 K entry BTB)

Application	% correct direction	Addr. %	BTB hit %
speech	94.6	62.3	76.9
full dict.	94.9	67.4	82.2
accurate	94.5	65.6	83.0

on all three systems. By moving to a 32 K-entry Gshare predictor with a 2 K-entry BTB, the performance difference between all three systems becomes almost indiscernible.

5.2 Optical Character Recognition

In this experiment, we chose a much harder document to scan than in the original study. For instance, the original study used a document that didn't have any pictures, whereas this study did have pictures interspersed with text. A summary of the differences are listed in Table 12.

TABLE 12. Difference in Input File for OCR

	easy	hard
pictures	no	yes
total fonts	1	5
special chars	2	3
total char sizes	2	8

TABLE 12. Difference in Input File for OCR

	easy	hard
total chars	2114	967
rejected chars	0	2
total words	377	205
suspect words	0	26
recognition time	2s	11s
words/min.	11310	1118

The easy image is the sample dispersed with *OmniPagePro 8.0*, while the hard image is the table of contents of a popular automotive magazine.

5.2.1 Memory Hierarchy

The performance of the L1 data cache for the hard OCR is much worse than that of the easy one as shown in Figure 13. For a 16KB four way set associative data cache, the complex OCR performs about 50% worse. As cache sizes get larger, the difference isn't as dramatic, but the complex OCR still has a poorer mpi. The reason for the poorer performance is that the complex OCR is harder to partition,

and it has to search many more font “dictionaries” in order to find a match.

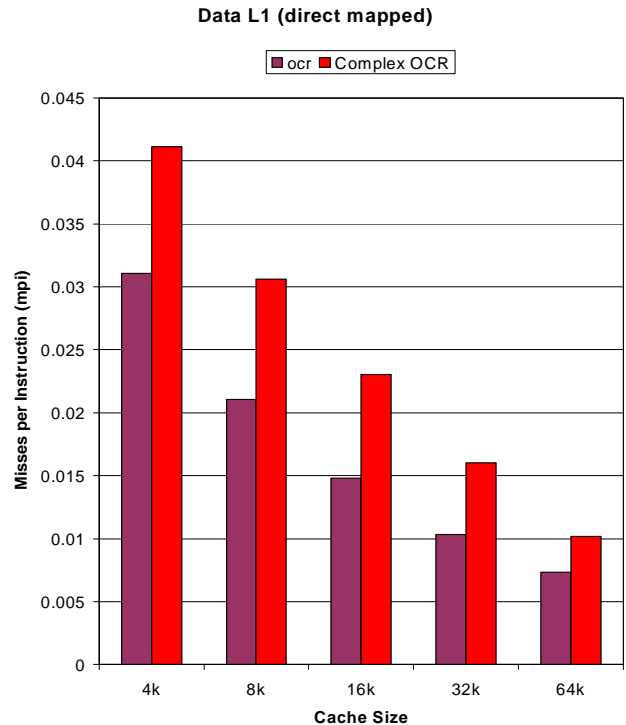


FIGURE 13. OCR direct mapped L1 data cache

The results for the L1 instruction cache illustrated in Figure 14 follow the explanation given in the previous section. Given that the complex OCR system has to spend more time partitioning images and text as well as more exhaustively search font “dictionaries”, it would seem that the instruction stream would have a higher locality. This is in fact the exact result that we see. The complex OCR system has about a 30% lower mpi than the base system for a 4 KB direct mapped cache. The difference then increases for 8KB and 16KB caches, but as the cache size

approaches 64 KB, the difference tapers off. This is where the increased L1 instruction size makes up for the locality.

complex OCR's instruction stream makes it perform slightly better on the larger L2 sizes. this effect is shown in

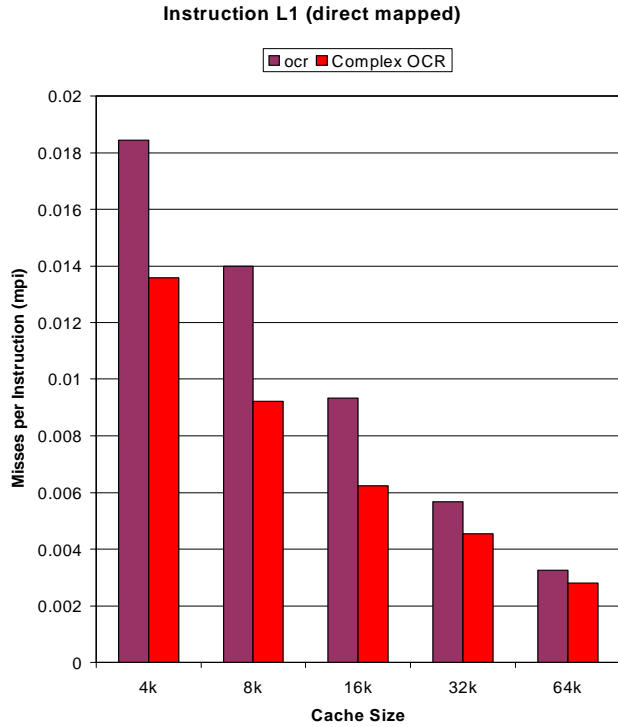


FIGURE 14. OCR direct mapped L1 instruction cache

For small L2 sizes the poor data locality makes the complex OCR perform worse; increasing the size of the L2 diminishes this effect and the increased locality of the

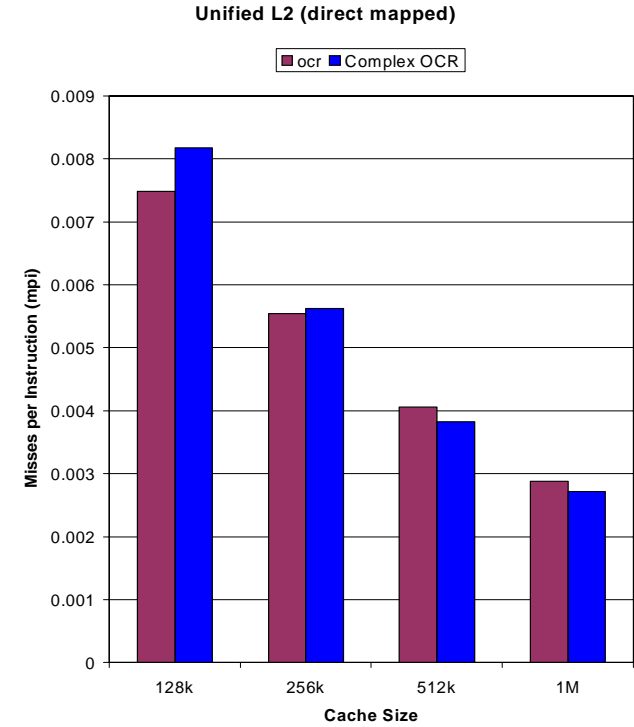


FIGURE 15. OCR direct mapped L2 unified cache

Figure 15.

5.2.2 TLB

For smaller data TLB sizes, the more complex OCR performs much more poorly than the base OCR system. However, this trend tends to dissipate as the TLB size gets larger. This result is seen in Figure 16 and can be attrib-

uted to the more complex partitioning and searching font dictionaries.

that the complex OCR system exhibits higher locality yet has a larger instruction working set.

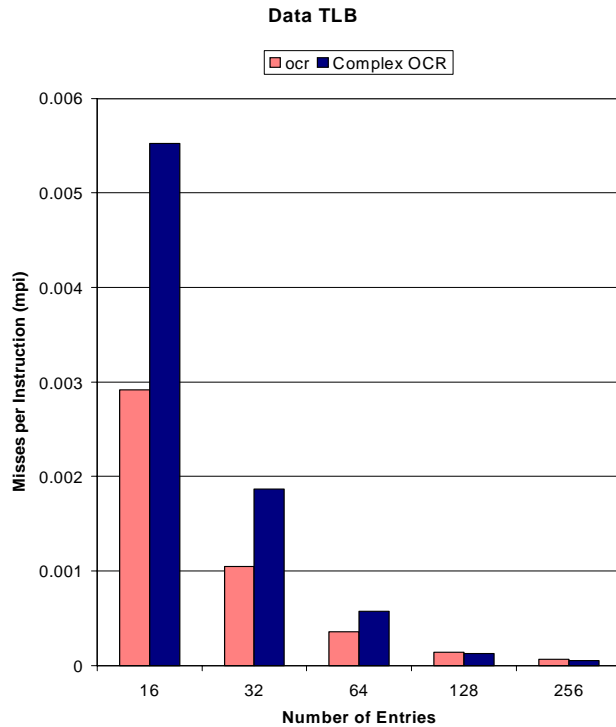


FIGURE 16. OCR fully associative data TLB

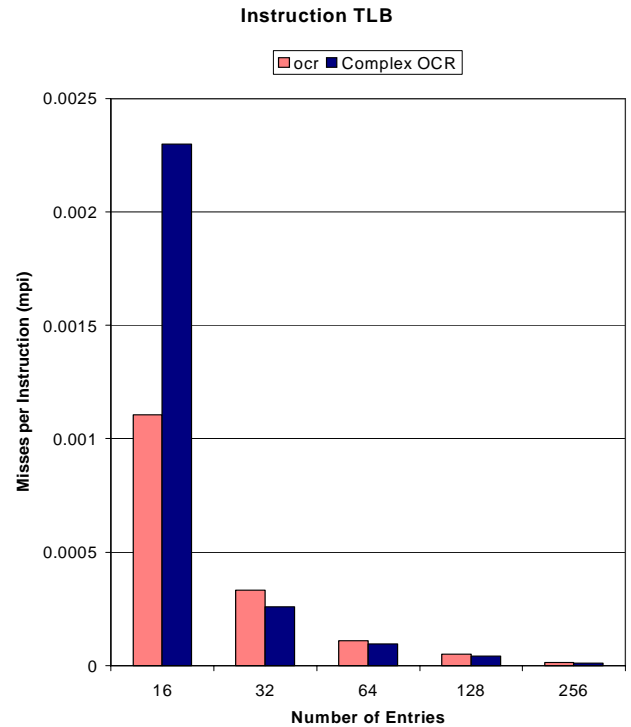


FIGURE 17. OCR fully associative instruction TLB

With regards to the instruction TLB, the complex OCR system seems to perform poorly only for the 16 entry case as illustrated in Figure 17. Given this result, combined with the results for the L1 instruction cache, it can be seen

5.2.3 Miss Classification

The miss classification breakdown for ocr and the complex ocr are quite different, as seen in Figure 18. Most importantly, the mpi is much worse for the complex ocr system, and an increase in the percent of capacity misses is the main reason. The percent of conflict misses decreases for the more complex system, but the overall number of conflict misses remains the same.

TABLE 13. Code and Data Footprint

Application	Data (MB)	Code (MB)
ocr	5.48	1.64
complex ocr	5.72	1.07

Table 13 gives a breakdown of the both data and code memory footprints. Interestingly, the complex ocr application has a larger code footprint, most likely attributed to

the application spending more time in earlier parts of the algorithm, such as segmentation.

TABLE 14. Data Reference Breakdown

Application	% sys call	% r	% w	% re/ r	% re/ w	w/o
ocr	8.97	70.2	80.3	68.1	78.8	29.8
complex ocr	8.63	82.0	70.2	79.6	68.9	17.9

Table 14 more completely illustrates the impact of trying to recognize a more complex input set. The application spends more time reading and correspondingly less time writing.

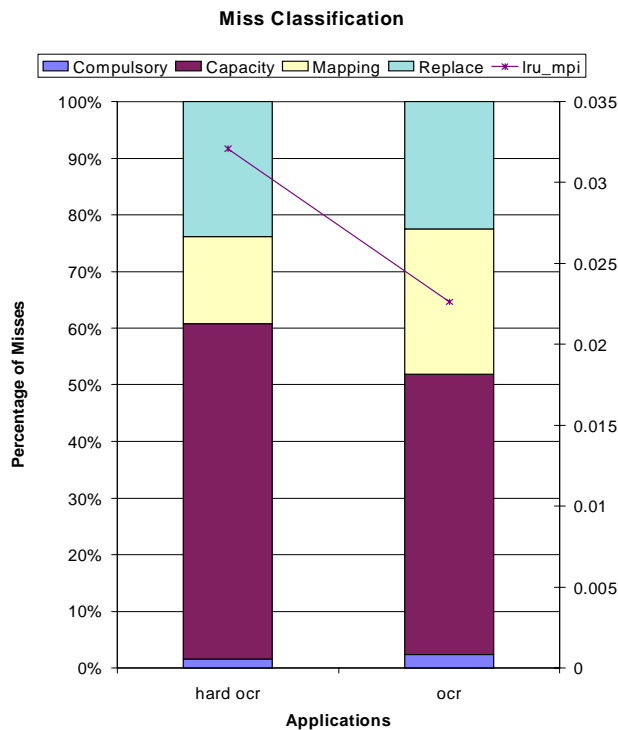


FIGURE 18. OCR miss classification breakdown

5.2.4 Branch Prediction

The control flow breakdown is highlighted in Table 15.

TABLE 15. Branch Breakdown

App.	% CB	% jmp	% ind. jmp	% dir. call	% ind call	% ret
ocr	86.8(67.6)	6.2	0.4	2.3	1.0	3.3
complex ocr	87.1(64.3)	8.0	0.2	1.9	0.5	2.4

There are less calls and returns in the complex ocr application, with a corresponding increase in jumps. The appli-

TABLE 16. 1024 entry Bimodal predictor (512 entry BTB)

Application	% correct direction	BTB hit %	Addr. %	num. branches (million)
ocr	91.8	87.9	71.8	28.8
complex ocr	90.1	91.7	69.4	33.0

action is spending more time in each call. The results from Table 15 can explain why an increase in BTB hit rate can still mean a decrease in the predictability of the addresses. Because the base OCR system has more calls, more of the target addresses can be obtained from the call-return stack; hence a higher address predictability. Table 17 shows the improvement with a 32 K-entry Gshare predictor, with a 2048 entry BTB. This table illustrates that while both

TABLE 17. 32 K entry Gshare predictor (2 K entry BTB)

Application	% correct direction	Addr. %	BTB hit %
ocr	95.3	71.1	89.9
complex ocr	94.5	68.8	92.6

applications improve, there is still a gap in performance.

5.3 Natural Language Translation

The natural language translation program that is used in this study is Globalink *Power Translator 6.0* which has the capability of adding subject specific dictionaries. For this extended study, we added the subject dictionary COM, for computer science terminology. *Power Translator 6.0* also has the capability of changing the order in which the libraries are searched. For our new study, we have two libraries,

the general one, GEN, and the subject specific one, COM. We then changed the search order to have *Power Translator* search COM before GEN. The input to this system was this paper.

5.3.1 Memory Hierarchy

Looking at the results for the L1 data cache in Figure 19, it seems that searching the specific computer science dictionary reduces the search space for the general dictionary. The specific dictionary acts as a cache, exploiting the “locality” of this paper. The most frequently looked up

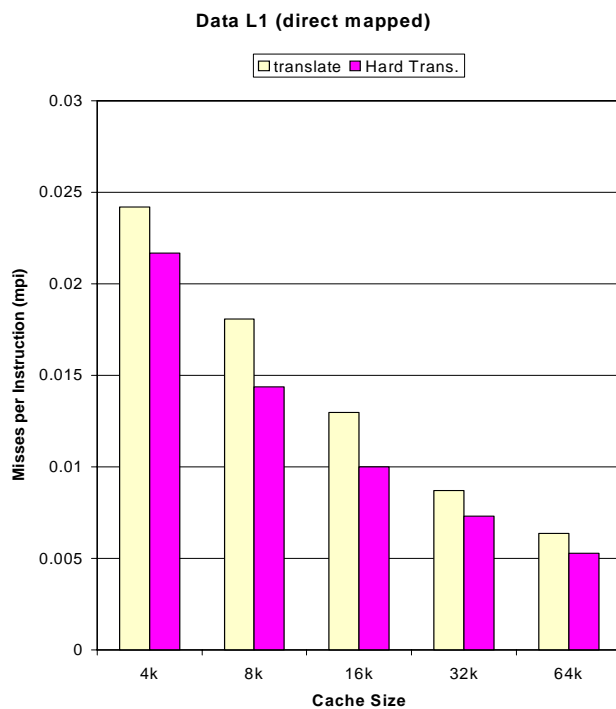


FIGURE 19. Translate direct mapped L1 data cache

words are filtered out by the COM dictionary.

The application spends more time in the specific computer science dictionary, the control loop ends up being “tighter” than in the ordinary translation. The result is slightly better instruction L1 performance as seen in Figure 20. Similar

results can be seen for the unified L2 cache as shown in Figure 21.

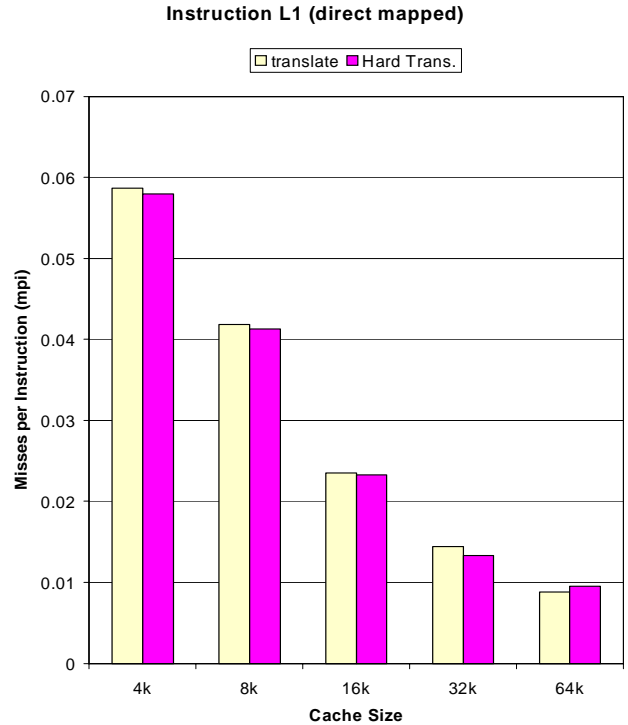


FIGURE 20. Translate direct mapped L1 instruction cache

The analogy to memory hierarchy for dictionary ordering is reasonably accurate. By assigning a preference to the smaller, more accurate dictionary we are directing the translation search to start with dictionary that is likely to

contain the desired translation. This increased hit rate translates into a better performing memory hierarchy.

the tighter control loop provided by the ordered dictionaries.

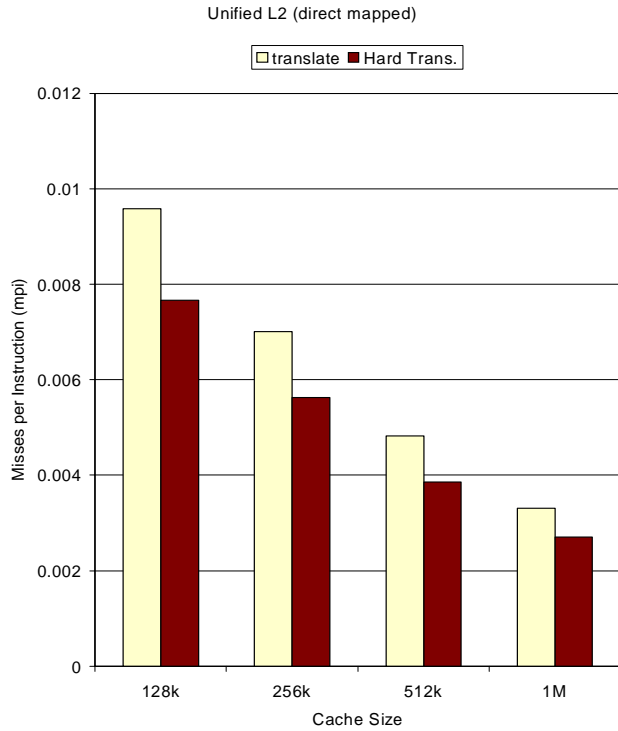


FIGURE 21. Translate direct mapped L2 unified cache

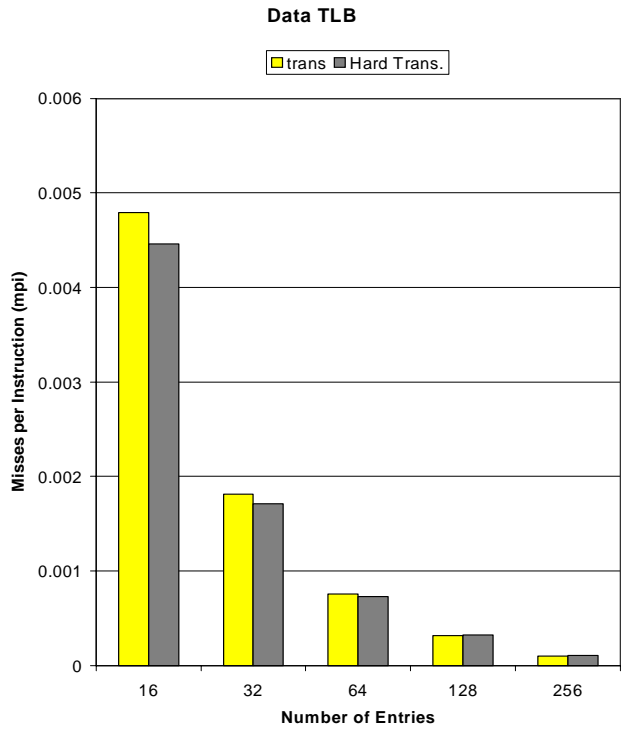


FIGURE 22. Translate fully associative data TLB

5.3.2 TLB

Although initially the base system’s performance is worse than the “hard” translation, the base system reacts more favorably to increasing TLB size. Since the “hard” translation actually contains more data (i.e. is a more linguistically complex document), the results in Figure 22 are to be expected.

For the instruction TLB, the “hard” translation actually performs better than the base system. This is because of

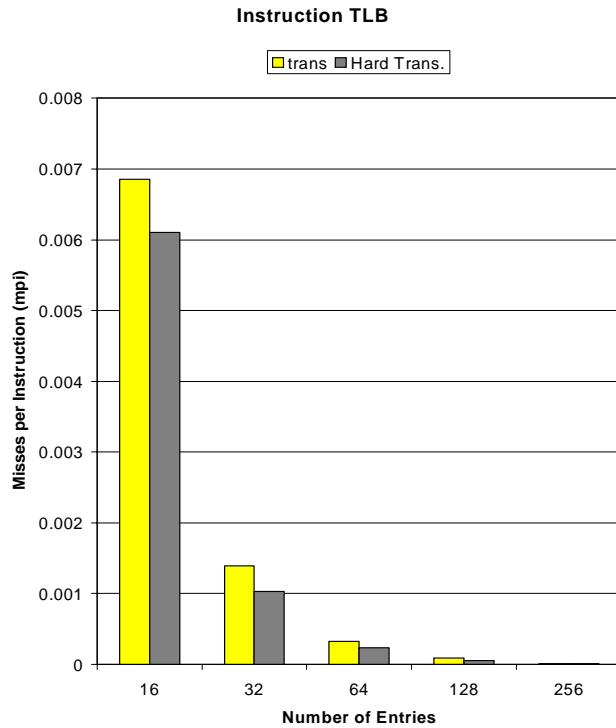


FIGURE 23. Translate fully associative instruction TLB

5.3.3 Miss Classification

Figure 24 below shows the difference in cache miss breakdown between the translation of an “easy” document, and the that of a “hard” document. The reduction in the number of compulsory misses corroborates the filtering effect of the COM dictionary.

TABLE 18. Code and Data Footprint

Application	Data (MB)	Code (MB)
tranlate	5.48	1.18
hard translate	3.70	0.88

Table 18 helps to identify the results obtained so far. There is a 32% decrease in the size of the data working set size, and a 25% decrease in the instruction working set size. The COM-GEN ordering of dictionaries definitely filtered

out most of the word lookups, with most of them hitting the smaller COM dictionary.

TABLE 19. Data Reference Breakdown

Application	% sys call	% r	% w	% re/r	% re/w	w/o
translate	6.01	80.0	68.4	76.5	62.9	20.0
hard trans.	2.87	91.8	63.7	86.5	58.1	8.17

The data reference breakdown, especially the increase in the fraction of read reuse, helps to illustrate the importance of dictionary ordering. There is also less time spent

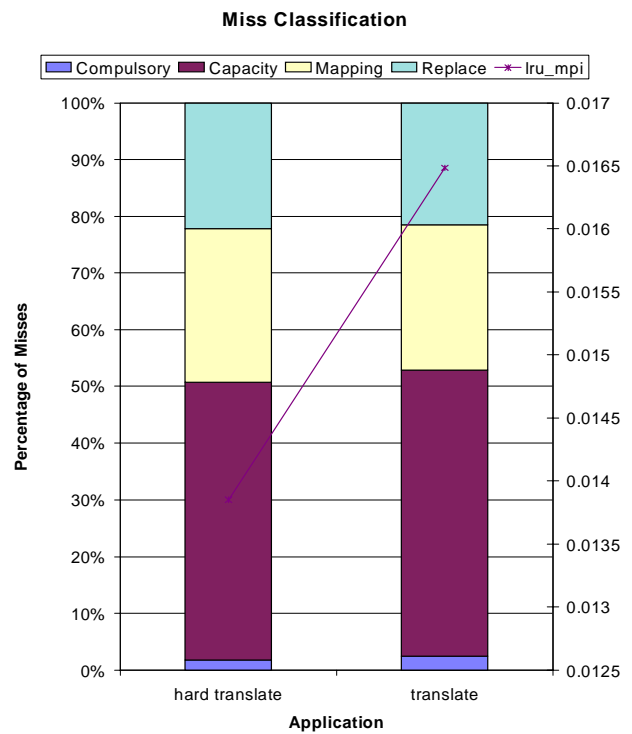


FIGURE 24. Translate miss classification breakdown

in operating system space, which should help out the branch prediction performance.

5.3.4 Branch Prediction

The control flow breakdown is highlighted in Table 20.

TABLE 20. Branch Breakdown

App.	% CB	% jmp	% ind. jmp	% dir. call	% ind call	% ret
translate	49.2(52.1)	11.8	1.1	15.0	3.0	10.1
hard trans.	48.2(45.3)	10.4	0.9	14.8	5.5	20.3

The one feature that stands out is the increase in the number of indirect calls for the hard system. Although the amount of data reuse is higher for the “hard” system, the increase in the number of indirect calls has an adverse effect on the address predictability aspect of branch predictor performance. For the 1 K-entry bimodal predictor,

TABLE 21. 1024 entry Bimodal predictor (512 entry BTB)

Application	% correct direction	BTB hit %	Addr. %	num. branches (million)
translate	91.5	71.2	71.2	31.2
hard trans.	91.6	70.2	68.8	25.6

both applications perform similarly, as shown in Table 21, except for the predictability of target addresses. Table 22 shows the improvement with a 32 K-entry Gshare predictor, with a 2048 entry BTB. Both applications seem to improve at the same rate with regard to the improved predictor.

TABLE 22. 32 K entry Gshare predictor (2 K entry BTB)

Application	% correct direction	Addr. %	BTB hit %
translate	96.0	74.1	78.7
hard trans.	96.5	71.5	76.5

6.0 Summary

In this paper, we have studied the natural I/O applications and compared them with the SPEC95 benchmarks. We performed a wide range of experiments on all of these

applications: memory performance, TLB performance, and branch prediction performance.

By comparing the two types of applications, we discover a significant difference in the branch prediction behavior. As a group, the natural I/O applications outperformed the SPEC95 benchmarks when it comes to overall branch predictor performance. The greater number of calls and taken conditional branches is the reason for the better target address predictability and direction prediction, respectively. The performance improvement is magnified with the better predictor and larger BTB.

In this framework, we used the term “hard” or “complex” to denote a supposedly more difficult task or input set. For instance, the hard translation versus the base translation, or the complex OCR versus the base OCR. Although intuitively it would appear that the “hard” or “complex” system would perform more poorly with respect to architectural structures (e.g. higher miss ratios), this is not always the case. Although complex OCR did exhibit this behavior, the hard translation system did not. In fact, even the complex OCR system exhibited lower miss ratios than its base counterpart on the unified L2 study. Some of the results are non-intuitive because of the complex interactions between data input set and the control algorithm.

The performance impact of speech recognition on the memory hierarchy is especially important. Even for large sizes of unified L2 caches, the resulting mpi for all of the speech recognition workloads is problematic. This makes the solution of building larger caches for speech recognition systems impractical. An alternative to large L2 caches has to be used to ameliorate the effect of speech recognition’s data reference pattern.

7.0 References

[1] T.M. Austin, D. Burger, S. Bennett. “Evaluating Future Microprocessors: The SimpleScalar Tool Set”, University of Wisconsin Technical Report. July 1996.

[2] D. Carter. “Rapid Development of Morphological Descriptions for Full Language Processing Systems”, *Proceedings of the 7th European ACL*, Dublin, Ireland, March 1995, pp. 202-209.

-
- [3] X.D. Huang, Y. Ariki, M.A. Jack. *Hidden Markov models for speech recognition*; Edinburgh: Edinburgh University Press, c1990.
- [4] C.H. Lee, F.K. Soong and K.K. Paliwal (Eds.), *Automatic Speech and Speaker Recognition: Advanced Topics*, Kluwer, Boston, 1996.
- [5] D. C. Lee, P. J. Crowley, J-L Baer, T. E. Anderson, and B. N. Bershad, "Execution Characteristics of Desktop Applications on Windows NT", *Proceedings of the 25th International Symposium on Computer Architecture*, pp. 27-38.
- [6] S. E. Levinson, L. R. Rabiner and M. M. Sondhi. *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition*, in Bell Syst. Tech. Jnl. v62(4), pp1035--1074, April 1983
- [7] L. Rabiner & J. Biing-Hwang. *Fundamentals of Speech Recognition*; Englewood Cliffs NJ: PTR Prentice Hall (Signal Processing Series), c1993.
- [8] S. Impedovo, L. Ottaviano, and S. Occhinegro, "Optical Character Recognition-a Survey", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 5, no. 1-2, pp. 1-24, 1991.
- [9] L. Schomaker, E. Hoenkamp, & M. Mayberry. "Towards collaborative agents for automatic on-line handwriting recognition." *Proceedings of the Third European Workshop on Handwriting Analysis and Recognition*, 14-15 July, 1998, London: The Institution of Electrical Engineers, Digest Number 1998/440, pp. 13/1-13/6.
- [10] L. Schomaker, G. Abbink, & S. Selen. "Writer and Writing-Style Classification in the Recognition of Online Handwriting." *Proceedings of the European Workshop on Handwriting Analysis and Recognition: A European Perspective*, 12-13 July, 1994, London: The Institution of Electrical Engineers, Digest Number 1994/123.
- [11] L. Schomaker. "User-interface aspects in recognizing connected-cursive handwriting." *Proceedings of the IEE Colloquium on Handwriting and Pen-based input*, 11 March, London: The Institution of Electrical Engineers, Digest Number 1994/065.
- [12] R. A. Sugummar. *Multi-Configuration Simulation Algorithms for the Evaluation of Computer Architecture Designs* Rabin A. Sugummar Ph.D. Thesis (Technical Report CSE-TR-173-93), University of Michigan, August 93.
- [13] *Dragon NaturallySpeaking Preferred: Getting Started*. Dragon Systems Inc. Version 3.0, June 1998.

1.0 Appendix

Table 1: Data Cache

	hand	ocr	trans	Speech	Com-press	Go	Gcc	Perl	Vortex	Full Speech	Complex OCR	Hard Trans.	Acc. Speech
4k direct	0.0253	0.0311	0.0242	0.0729	0.0378	0.0725	0.0383	0.0470	0.0530	0.0714	0.0411	0.0217	0.0689
4k 2way	0.0188	0.0230	0.0168	0.0581	0.0331	0.0559	0.0276	0.0172	0.0382	0.0570	0.0326	0.0141	0.0530
4k 4way	0.0176	0.0206	0.0142	0.0552	0.0329	0.0474	0.0242	0.0077	0.0332	0.0540	0.0302	0.0116	0.0501
4k 8way	0.0172	0.0188	0.0135	0.0515	0.0309	0.0430	0.0231	0.0080	0.0312	0.0501	0.0291	0.0109	0.0464
8k direct	0.0189	0.0211	0.0180	0.0518	0.0287	0.0482	0.0261	0.0253	0.0336	0.0496	0.0306	0.0144	0.0472
8k 2way	0.0137	0.0165	0.0114	0.0374	0.0255	0.0304	0.0181	0.0076	0.0274	0.0364	0.0258	0.0093	0.0356
8k 4way	0.0124	0.0142	0.0101	0.0334	0.0243	0.0242	0.0160	0.0013	0.0230	0.0328	0.0225	0.0082	0.0318
8k 8way	0.0122	0.0132	0.0096	0.0320	0.0242	0.0215	0.0152	0.0011	0.0216	0.0317	0.0201	0.0077	0.0304
16k direct	0.0136	0.0148	0.0130	0.0342	0.0184	0.0286	0.0170	0.0169	0.0245	0.0338	0.0230	0.0100	0.0326
16k 2way	0.0101	0.0111	0.0085	0.0258	0.0148	0.0176	0.0116	0.0011	0.0183	0.0256	0.0178	0.0067	0.0255
16k 4way	0.0091	0.0102	0.0074	0.0234	0.0140	0.0131	0.0101	0.0008	0.0165	0.0231	0.0165	0.0060	0.0229
16k 8way	0.0087	0.0095	0.0071	0.0225	0.0133	0.0122	0.0096	0.0008	0.0154	0.0221	0.0157	0.0058	0.0220
32k direct	0.0098	0.0103	0.0087	0.0250	0.0082	0.0175	0.0110	0.0164	0.0171	0.0266	0.0160	0.0073	0.0247
32k 2way	0.0076	0.0078	0.0065	0.0198	0.0065	0.0096	0.0071	0.0007	0.0119	0.0192	0.0129	0.0052	0.0196
32k 4way	0.0070	0.0072	0.0060	0.0183	0.0047	0.0074	0.0058	0.0006	0.0104	0.0179	0.0128	0.0048	0.0180
32k 8way	0.0069	0.0069	0.0058	0.0180	0.0043	0.0068	0.0054	0.0005	0.0095	0.0173	0.0127	0.0047	0.0174
64k direct	0.0067	0.0073	0.0064	0.0199	0.0030	0.0124	0.0071	0.0008	0.0116	0.0187	0.0102	0.0053	0.0204
64k 2way	0.0054	0.0055	0.0051	0.0165	0.0013	0.0047	0.0043	0.0004	0.0071	0.0157	0.0075	0.0041	0.0160
64k 4way	0.0051	0.0050	0.0047	0.0156	0.0009	0.0031	0.0037	0.0003	0.0057	0.0146	0.0063	0.0038	0.0150
64k 8way	0.0050	0.0049	0.0046	0.0154	0.0007	0.0027	0.0035	0.0003	0.0050	0.0143	0.0057	0.0037	0.0147

Table 2: Instruction Cache

	hand	ocr	trans	Speech	Com-press	Go	Gcc	Perl	Vortex	Full Speech	Complex OCR	Hard Trans.	Acc. Speech
4k direct	0.0149	0.0185	0.0586	0.0181	0.0019	0.0624	0.0483	0.0313	0.0710	0.0163	0.0136	0.0579	0.0139
4k 2way	0.0126	0.0163	0.0461	0.0135	0.0018	0.0603	0.0452	0.0210	0.0658	0.0130	0.0129	0.0458	0.0099
4k 4way	0.0120	0.0155	0.0400	0.0114	0.0018	0.0589	0.0435	0.0183	0.0651	0.0110	0.0121	0.0369	0.0078
4k 8way	0.0117	0.0153	0.0392	0.0099	0.0018	0.0596	0.0433	0.0177	0.0656	0.0100	0.0116	0.0354	0.0066
8k direct	0.0106	0.0140	0.0419	0.0119	0.0015	0.0458	0.0360	0.0212	0.0539	0.0126	0.0092	0.0413	0.0085
8k 2way	0.0076	0.0108	0.0269	0.0085	0.0014	0.0421	0.0307	0.0112	0.0468	0.0087	0.0064	0.0245	0.0059
8k 4way	0.0066	0.0099	0.0214	0.0066	0.0014	0.0407	0.0294	0.0035	0.0444	0.0079	0.0055	0.0196	0.0051
8k 8way	0.0061	0.0093	0.0189	0.0057	0.0013	0.0400	0.0293	0.0032	0.0436	0.0074	0.0040	0.0160	0.0042
16k direct	0.0076	0.0093	0.0235	0.0078	0.0012	0.0337	0.0232	0.0072	0.0362	0.0088	0.0062	0.0233	0.0050
16k 2way	0.0047	0.0066	0.0158	0.0046	0.0010	0.0248	0.0172	0.0023	0.0292	0.0059	0.0042	0.0142	0.0036
16k 4way	0.0040	0.0055	0.0115	0.0039	0.0010	0.0210	0.0134	0.0018	0.0270	0.0048	0.0026	0.0091	0.0028
16k 8way	0.0037	0.0052	0.0095	0.0036	0.0009	0.0199	0.0121	0.0018	0.0264	0.0045	0.0023	0.0072	0.0027
32k direct	0.0045	0.0057	0.0145	0.0059	0.0008	0.0192	0.0133	0.0050	0.0209	0.0052	0.0045	0.0133	0.0039
32k 2way	0.0029	0.0035	0.0084	0.0026	0.0006	0.0133	0.0077	0.0014	0.0151	0.0035	0.0027	0.0073	0.0022
32k 4way	0.0025	0.0027	0.0061	0.0021	0.0005	0.0088	0.0058	0.0010	0.0122	0.0027	0.0016	0.0044	0.0014
32k 8way	0.0024	0.0024	0.0054	0.0020	0.0004	0.0079	0.0046	0.0009	0.0096	0.0025	0.0014	0.0039	0.0013
64k direct	0.0030	0.0032	0.0088	0.0031	0.0006	0.0049	0.0083	0.0022	0.0133	0.0036	0.0028	0.0095	0.0030
64k 2way	0.0018	0.0018	0.0045	0.0015	0.0004	0.0040	0.0037	0.0007	0.0052	0.0018	0.0015	0.0039	0.0014
64k 4way	0.0016	0.0014	0.0033	0.0011	0.0003	0.0036	0.0026	0.0005	0.0027	0.0015	0.0009	0.0023	0.0007
64k 8way	0.0015	0.0013	0.0030	0.0010	0.0002	0.0036	0.0023	0.0005	0.0019	0.0014	0.0009	0.0021	0.0006

Table 3: L2 Unified Cache

	hand	ocr	trans	Speech	Com- press	Go	Gcc	Perl	Vortex	Full Speech	Com- plex OCR	Hard Trans.	Acc. Speech
128k direct	0.0072	0.0075	0.0096	0.0182	0.0013	0.0114	0.0081	0.0012	0.0151	0.0181	0.0082	0.0077	0.0174
128k 2way	0.0063	0.0066	0.0084	0.0171	0.0010	0.0085	0.0064	0.0010	0.0105	0.0167	0.0062	0.0066	0.0160
128k 4way	0.0059	0.0060	0.0077	0.0166	0.0008	0.0069	0.0056	0.0008	0.0078	0.0160	0.0052	0.0061	0.0152
128k 8way	0.0057	0.0058	0.0074	0.0163	0.0008	0.0063	0.0052	0.0007	0.0064	0.0156	0.0048	0.0058	0.0149
256k direct	0.0056	0.0055	0.0070	0.0151	0.0009	0.0052	0.0055	0.0009	0.0098	0.0150	0.0056	0.0056	0.0148
256k 2way	0.0048	0.0045	0.0056	0.0143	0.0007	0.0038	0.0040	0.0006	0.0056	0.0138	0.0042	0.0046	0.0137
256k 4way	0.0044	0.0040	0.0048	0.0140	0.0006	0.0023	0.0034	0.0005	0.0041	0.0133	0.0036	0.0040	0.0131
256k 8way	0.0043	0.0037	0.0045	0.0140	0.0005	0.0015	0.0032	0.0005	0.0036	0.0132	0.0034	0.0037	0.0129
512k direct	0.0044	0.0041	0.0048	0.0117	0.0007	0.0026	0.0036	0.0007	0.0072	0.0116	0.0038	0.0039	0.0125
512k 2way	0.0035	0.0031	0.0037	0.0106	0.0005	0.0012	0.0024	0.0004	0.0034	0.0104	0.0027	0.0031	0.0115
512k 4way	0.0032	0.0027	0.0031	0.0102	0.0005	0.0007	0.0021	0.0003	0.0029	0.0098	0.0024	0.0027	0.0111
512k 8way	0.0031	0.0026	0.0029	0.0100	0.0004	0.0005	0.0020	0.0003	0.0027	0.0096	0.0023	0.0025	0.0111
1M direct	0.0033	0.0029	0.0033	0.0082	0.0005	0.0008	0.0022	0.0005	0.0042	0.0082	0.0027	0.0027	0.0099
1M 2way	0.0027	0.0021	0.0023	0.0063	0.0004	0.0004	0.0014	0.0003	0.0027	0.0066	0.0019	0.0020	0.0090
1M 4way	0.0025	0.0019	0.0020	0.0053	0.0004	0.0003	0.0012	0.0002	0.0024	0.0056	0.0017	0.0017	0.0086
1M 8way	0.0024	0.0018	0.0018	0.0047	0.0004	0.0002	0.0011	0.0002	0.0023	0.0050	0.0016	0.0017	0.0084
2M direct				0.0054						0.0054	0.0019	0.0018	0.0071
2M 2way				0.0034						0.0036	0.0013	0.0012	0.0060
2M 4way				0.0024						0.0026	0.0012	0.0009	0.0055
2M 8way				0.0020						0.0021	0.0011	0.0008	0.0052
4M direct				0.0034						0.0034	0.0014	0.0011	0.0047
4M 2way				0.0019						0.0020	0.0009	0.0007	0.0032
4M 4way				0.0014						0.0015	0.0008	0.0005	0.0025
4M 8way				0.0013						0.0013	0.0008	0.0004	0.0020
8M direct				0.0022						0.0023	0.0011	0.0007	0.0030
8M 2way				0.0012						0.0013	0.0007	0.0004	0.0017
8M 4way				0.0010						0.0010	0.0007	0.0003	0.0012
8M 8way				0.0010						0.0010	0.0006	0.0003	0.0011
16M direct				0.0015						0.0015	0.0009	0.0005	0.0019
16M 2way				0.0010						0.0009	0.0006	0.0003	0.0011
16M 4way				0.0009						0.0008	0.0006	0.0003	0.0010
16M 8way				0.0008						0.0008	0.0006	0.0003	0.0010

Table 4: Data TLB

	hand	ocr	trans	speech	compress	go	gcc	perl	vortex	Full Speech	Com- plex OCR	Hard Trans.	Acc. Speech
16e4w	2.41E-03	3.38E-03	6.48E-03	1.76E-02	1.77E-03	3.55E-02	1.14E-02	6.78E-04	1.51E-02	1.68E-02	6.78E-03	6.16E-03	1.91E-02
16e8w	2.09E-03	2.95E-03	5.48E-03	1.50E-02	1.30E-03	3.08E-02	9.48E-03	6.41E-04	1.43E-02	1.42E-02	5.38E-03	5.04E-03	1.67E-02
16e16w	2.04E-03	2.92E-03	4.79E-03	1.42E-02	1.26E-03	3.05E-02	7.38E-03	6.50E-04	1.39E-02	1.34E-02	5.52E-03	4.46E-03	1.53E-02
32e4w	1.24E-03	1.35E-03	2.99E-03	8.28E-03	3.58E-04	6.14E-03	3.29E-03	2.60E-04	8.79E-03	8.40E-03	2.50E-03	2.82E-03	1.01E-02
32e8w	1.08E-03	1.14E-03	2.27E-03	6.98E-03	2.82E-04	4.59E-03	2.26E-03	2.37E-04	8.13E-03	7.05E-03	2.04E-03	2.12E-03	8.79E-03
32e16w	1.02E-03	1.04E-03	2.03E-03	6.47E-03	2.56E-04	3.75E-03	1.87E-03	2.22E-04	7.99E-03	6.58E-03	1.79E-03	1.92E-03	8.28E-03
32e32w	1.01E-03	1.05E-03	1.82E-03	6.26E-03	2.60E-04	2.52E-03	1.70E-03	2.22E-04	8.11E-03	6.36E-03	1.87E-03	1.71E-03	8.81E-03
64e4w	6.76E-04	5.84E-04	1.65E-03	3.81E-03	9.73E-05	1.87E-03	8.00E-04	1.23E-04	4.20E-03	4.05E-03	1.04E-03	1.48E-03	4.42E-03
64e8w	5.29E-04	4.28E-04	1.09E-03	2.69E-03	8.29E-05	3.11E-04	6.42E-04	9.67E-05	3.60E-03	2.73E-03	8.64E-04	1.05E-03	3.15E-03
64e16w	4.40E-04	3.79E-04	8.90E-04	2.53E-03	7.21E-05	2.84E-04	5.67E-04	9.01E-05	3.35E-03	2.56E-03	9.71E-04	8.90E-04	2.92E-03
64e32w	4.25E-04	3.60E-04	8.02E-04	2.47E-03	6.77E-05	2.82E-04	5.46E-04	8.67E-05	3.26E-03	2.51E-03	9.37E-04	7.74E-04	2.80E-03
64e64w	4.22E-04	3.60E-04	7.59E-04	2.45E-03	6.60E-05	2.79E-04	5.41E-04	8.65E-05	3.24E-03	2.49E-03	5.73E-04	7.26E-04	2.78E-03
128e4w	3.34E-04	2.14E-04	7.66E-04	1.81E-03	4.24E-05	8.10E-05	3.79E-04	5.39E-05	2.12E-03	1.90E-03	4.96E-04	7.83E-04	2.02E-03
128e8w	2.81E-04	1.82E-04	6.62E-04	1.55E-03	3.62E-05	6.91E-05	3.62E-04	4.43E-05	1.92E-03	1.50E-03	4.99E-04	7.24E-04	1.71E-03
128e16w	2.32E-04	1.46E-04	4.34E-04	1.47E-03	3.53E-05	6.54E-05	3.51E-04	3.48E-05	1.84E-03	1.42E-03	4.60E-04	5.15E-04	1.64E-03
128e32w	2.25E-04	1.39E-04	3.60E-04	1.45E-03	3.47E-05	6.47E-05	3.47E-04	2.99E-05	1.86E-03	1.39E-03	1.32E-04	3.98E-04	1.62E-03

Table 4: Data TLB

128e64w	2.22E-04	1.37E-04	3.30E-04	1.43E-03	3.14E-05	6.45E-05	3.43E-04	2.82E-05	1.87E-03	1.38E-03	1.24E-04	3.45E-04	1.61E-03
128e128w	2.20E-04	1.37E-04	3.13E-04	1.42E-03	3.28E-05	6.44E-05	3.43E-04	2.80E-05	1.88E-03	1.38E-03	1.23E-04	3.25E-04	1.60E-03
256e4w	1.87E-04	9.68E-05	3.23E-04	1.02E-03	2.15E-05	3.18E-05	1.46E-04	2.11E-05	7.97E-04	9.38E-04	8.81E-05	3.99E-04	1.02E-03
256e8w	1.60E-04	7.56E-05	2.69E-04	9.56E-04	1.88E-05	2.45E-05	1.49E-04	1.83E-05	5.93E-04	8.77E-04	6.82E-05	3.67E-04	9.58E-04
256e16w	1.52E-04	7.26E-05	2.48E-04	9.18E-04	1.88E-05	2.17E-05	1.55E-04	1.78E-05	5.42E-04	8.40E-04	6.45E-05	3.49E-04	9.24E-04
256e32w	1.44E-04	6.57E-05	1.45E-04	8.94E-04	1.54E-05	2.00E-05	1.58E-04	1.43E-05	4.85E-04	8.21E-04	5.58E-05	1.97E-04	9.12E-04
256e64w	1.43E-04	6.41E-05	1.17E-04	8.89E-04	1.50E-05	1.90E-05	1.61E-04	1.38E-05	4.64E-04	8.13E-04	5.42E-05	1.36E-04	9.06E-04
256e128w	1.42E-04	6.32E-05	1.04E-04	8.84E-04	1.50E-05	1.90E-05	1.65E-04	1.37E-05	4.56E-04	8.10E-04	5.35E-05	1.14E-04	9.02E-04
256e256w	1.42E-04	6.33E-05	9.97E-05	8.84E-04	1.50E-05	1.89E-05	1.67E-04	1.36E-05	4.56E-04	8.09E-04	5.33E-05	1.07E-04	9.03E-04

Table 5: Instruction TLB

	hand	ocr	trans	speech	compress	go	gcc	perl	vortex	Full Speech	Complex OCR	Hard Trans.	Acc. Speech
16e4w	8.34E-04	1.15E-03	7.94E-03	1.09E-03	1.64E-04	3.85E-04	1.59E-03	3.14E-04	1.90E-03	1.44E-03	2.28E-03	6.86E-03	1.10E-03
16e8w	7.85E-04	1.14E-03	7.24E-03	8.68E-04	1.57E-04	3.52E-04	1.41E-03	2.94E-04	1.83E-03	1.18E-03	2.41E-03	6.34E-03	9.02E-04
16e16w	7.42E-04	1.11E-03	6.85E-03	8.72E-04	1.53E-04	3.53E-04	1.38E-03	3.01E-04	1.86E-03	1.19E-03	2.30E-03	6.10E-03	9.45E-04
32e4w	3.67E-04	4.93E-04	2.61E-03	4.82E-04	8.83E-05	1.93E-04	5.68E-04	1.65E-04	6.59E-04	6.37E-04	7.81E-04	2.22E-03	4.21E-04
32e8w	3.06E-04	3.45E-04	2.24E-03	4.41E-04	7.53E-05	1.77E-04	4.73E-04	1.44E-04	5.45E-04	5.81E-04	4.72E-04	1.91E-03	3.77E-04
32e16w	2.98E-04	3.35E-04	1.40E-03	4.19E-04	7.61E-05	1.78E-04	4.59E-04	1.45E-04	4.99E-04	5.58E-04	2.64E-04	1.03E-03	3.43E-04
32e32w	2.88E-04	3.32E-04	1.39E-03	4.20E-04	7.63E-05	1.79E-04	4.58E-04	1.45E-04	5.13E-04	5.69E-04	2.60E-04	1.03E-03	3.20E-04
64e4w	1.45E-04	1.74E-04	4.93E-04	2.06E-04	3.77E-05	6.89E-05	1.96E-04	7.16E-05	1.90E-04	2.66E-04	2.91E-04	3.86E-04	1.59E-04
64e8w	1.28E-04	1.23E-04	3.63E-04	1.89E-04	2.93E-05	5.77E-05	1.58E-04	5.72E-05	1.36E-04	2.43E-04	1.07E-04	2.56E-04	1.39E-04
64e16w	1.16E-04	1.13E-04	3.42E-04	1.81E-04	2.30E-05	4.73E-05	1.39E-04	4.81E-05	1.15E-04	2.27E-04	9.79E-05	2.42E-04	1.09E-04
64e32w	1.12E-04	1.08E-04	3.29E-04	1.77E-04	2.01E-05	4.28E-05	1.32E-04	4.36E-05	1.11E-04	2.25E-04	9.33E-05	2.33E-04	1.05E-04
64e64w	1.13E-04	1.09E-04	3.23E-04	1.75E-04	1.99E-05	4.21E-05	1.31E-04	4.38E-05	1.10E-04	2.24E-04	9.39E-05	2.32E-04	1.04E-04
128e4w	5.43E-05	6.02E-05	1.34E-04	8.21E-05	9.83E-06	2.05E-05	6.14E-05	2.02E-05	4.27E-05	1.10E-04	5.17E-05	1.08E-04	5.01E-05
128e8w	4.64E-05	5.42E-05	1.08E-04	7.81E-05	8.64E-06	1.69E-05	5.10E-05	1.70E-05	3.04E-05	1.02E-04	4.63E-05	7.76E-05	4.75E-05
128e16w	4.28E-05	5.12E-05	9.55E-05	7.43E-05	8.30E-06	1.59E-05	4.88E-05	1.61E-05	2.88E-05	9.77E-05	4.24E-05	5.92E-05	4.46E-05
128e32w	4.19E-05	4.94E-05	9.43E-05	6.93E-05	7.93E-06	1.50E-05	4.76E-05	1.56E-05	2.73E-05	9.15E-05	4.10E-05	5.42E-05	3.98E-05
128e64w	4.13E-05	4.91E-05	9.50E-05	6.89E-05	7.84E-06	1.49E-05	4.71E-05	1.54E-05	2.61E-05	9.07E-05	4.06E-05	5.30E-05	3.87E-05
128e128w	4.13E-05	4.91E-05	9.39E-05	6.87E-05	7.77E-06	1.48E-05	4.68E-05	1.53E-05	2.59E-05	9.05E-05	4.07E-05	5.41E-05	3.84E-05
256e4w	2.11E-05	2.20E-05	2.60E-05	2.79E-05	3.67E-06	7.72E-06	1.72E-05	8.14E-06	1.39E-05	3.98E-05	2.02E-05	1.85E-05	1.52E-05
256e8w	1.65E-05	1.72E-05	1.55E-05	1.93E-05	3.37E-06	6.63E-06	1.63E-05	7.38E-06	1.10E-05	2.98E-05	1.53E-05	8.08E-06	1.03E-05
256e16w	1.56E-05	1.52E-05	1.15E-05	1.44E-05	3.15E-06	6.08E-06	1.52E-05	7.09E-06	1.02E-05	2.37E-05	1.40E-05	6.59E-06	7.88E-06
256e32w	1.48E-05	1.46E-05	1.08E-05	1.29E-05	2.81E-06	5.84E-06	1.48E-05	6.89E-06	9.78E-06	2.18E-05	1.27E-05	5.17E-06	6.50E-06
256e64w	1.44E-05	1.37E-05	1.06E-05	9.98E-06	2.60E-06	5.79E-06	1.45E-05	6.80E-06	9.79E-06	1.89E-05	1.18E-05	5.01E-06	4.30E-06
256e128w	1.43E-05	1.36E-05	1.06E-05	8.67E-06	2.58E-06	5.78E-06	1.44E-05	6.74E-06	9.79E-06	1.75E-05	1.17E-05	5.00E-06	2.74E-06
256e256w	1.43E-05	1.36E-05	1.06E-05	8.62E-06	2.60E-06	5.78E-06	1.45E-05	6.76E-06	9.77E-06	1.76E-05	1.16E-05	4.96E-06	2.64E-06

Table 6: Unified TLB

	hand	ocr	trans	speech	compress	go	gcc	perl	vortex	Full Speech	Complex OCR	Hard Trans.	Acc. Speech
16e4w	6.12E-03	8.82E-03	2.90E-02	2.95E-02	2.96E-03	5.08E-02	2.36E-02	9.70E-03	3.01E-02	2.81E-02	2.81E-02	2.69E-02	3.08E-02
16e8w	5.23E-03	7.57E-03	2.52E-02	2.46E-02	1.97E-03	4.60E-02	2.04E-02	8.99E-03	2.48E-02	2.33E-02	2.33E-02	2.24E-02	2.58E-02
16e16w	5.13E-03	7.41E-03	2.37E-02	2.28E-02	1.90E-03	4.55E-02	1.76E-02	8.18E-03	2.38E-02	2.13E-02	2.13E-02	2.07E-02	2.39E-02
32e4w	3.05E-03	4.08E-03	1.27E-02	1.46E-02	7.16E-04	1.52E-02	8.87E-03	1.02E-03	1.53E-02	1.49E-02	1.49E-02	1.18E-02	1.70E-02
32e8w	2.73E-03	3.62E-03	1.14E-02	1.31E-02	6.25E-04	1.28E-02	7.34E-03	8.74E-04	1.40E-02	1.36E-02	1.36E-02	1.04E-02	1.56E-02
32e16w	2.63E-03	3.56E-03	1.06E-02	1.26E-02	6.00E-04	1.12E-02	6.71E-03	7.89E-04	1.40E-02	1.27E-02	1.27E-02	9.73E-03	1.48E-02
32e32w	2.61E-03	3.52E-03	9.65E-03	1.20E-02	6.04E-04	9.72E-03	6.36E-03	7.88E-04	1.41E-02	1.22E-02	1.22E-02	8.90E-03	1.42E-02
64e4w	1.61E-03	1.68E-03	4.62E-03	6.45E-03	2.93E-04	4.20E-03	2.70E-03	4.15E-04	7.46E-03	6.72E-03	6.72E-03	3.94E-03	7.94E-03
64e8w	1.27E-03	1.30E-03	3.57E-03	4.91E-03	2.60E-04	9.31E-04	2.19E-03	3.72E-04	6.78E-03	4.93E-03	4.93E-03	3.20E-03	5.99E-03
64e16w	1.20E-03	1.20E-03	3.30E-03	4.10E-03	2.55E-04	8.08E-04	1.81E-03	3.51E-04	6.26E-03	4.47E-03	4.47E-03	2.88E-03	5.07E-03
64e32w	1.12E-03	1.13E-03	3.13E-03	3.93E-03	2.33E-04	7.81E-04	1.65E-03	3.45E-04	6.26E-03	4.27E-03	4.27E-03	2.66E-03	4.77E-03

Table 6: Unified TLB

64e64w	1.11E-03	1.12E-03	2.76E-03	3.82E-03	2.28E-04	7.71E-04	1.62E-03	3.43E-04	6.27E-03	4.23E-03	4.23E-03	2.39E-03	4.75E-03
128e4w	7.35E-04	6.05E-04	1.81E-03	2.78E-03	1.07E-04	3.97E-04	8.25E-04	1.81E-04	3.43E-03	2.88E-03	2.88E-03	1.54E-03	2.98E-03
128e8w	5.94E-04	4.57E-04	1.37E-03	2.26E-03	8.81E-05	2.27E-04	6.99E-04	1.40E-04	3.03E-03	2.33E-03	2.33E-03	1.27E-03	2.41E-03
128e16w	4.97E-04	3.91E-04	1.12E-03	2.13E-03	8.28E-05	2.21E-04	6.64E-04	1.28E-04	2.76E-03	2.23E-03	2.23E-03	1.05E-03	2.31E-03
128e32w	4.82E-04	3.75E-04	1.03E-03	2.11E-03	7.91E-05	2.06E-04	6.48E-04	1.18E-04	2.71E-03	2.21E-03	2.21E-03	9.29E-04	2.23E-03
128e64w	4.68E-04	3.67E-04	1.00E-03	2.10E-03	7.75E-05	2.04E-04	6.38E-04	1.13E-04	2.69E-03	2.20E-03	2.20E-03	8.81E-04	2.21E-03
128e128w	4.56E-04	3.67E-04	9.72E-04	2.10E-03	7.67E-05	2.05E-04	6.37E-04	1.12E-04	2.67E-03	2.18E-03	2.18E-03	8.46E-04	2.20E-03
256e4w	3.59E-04	2.46E-04	7.02E-04	1.42E-03	4.30E-05	8.32E-05	3.53E-04	6.31E-05	1.25E-03	1.38E-03	1.38E-03	6.63E-04	1.40E-03
256e8w	2.73E-04	1.77E-04	5.31E-04	1.28E-03	3.66E-05	6.69E-05	3.44E-04	4.92E-05	9.85E-04	1.25E-03	1.25E-03	5.67E-04	1.28E-03
256e16w	2.57E-04	1.69E-04	4.72E-04	1.23E-03	3.39E-05	6.01E-05	3.44E-04	4.38E-05	9.35E-04	1.19E-03	1.19E-03	5.31E-04	1.23E-03
256e32w	2.45E-04	1.62E-04	3.65E-04	1.19E-03	3.01E-05	5.93E-05	3.46E-04	3.99E-05	8.30E-04	1.17E-03	1.17E-03	3.85E-04	1.21E-03
256e64w	2.40E-04	1.60E-04	3.43E-04	1.18E-03	2.93E-05	5.55E-05	3.49E-04	3.84E-05	7.77E-04	1.16E-03	1.16E-03	3.33E-04	1.20E-03
256e128w	2.39E-04	1.60E-04	3.34E-04	1.18E-03	2.89E-05	5.57E-05	3.49E-04	3.77E-05	7.28E-04	1.15E-03	1.15E-03	3.17E-04	1.19E-03
256e256w	2.38E-04	1.60E-04	3.31E-04	1.18E-03	2.87E-05	5.56E-05	3.48E-04	3.77E-05	7.29E-04	1.15E-03	1.15E-03	3.08E-04	1.19E-03