# Workload Characterization of Biometric Applications on Pentium 4 Microarchitecture

Chang-Burm Cho[1], Asmita V. Chande[2], Yue Li[3] and Tao Li[4]

*Intelligent Design of Efficient Architecture Lab (IDEAL)*
*Department of Electrical and Computer Engineering*
*University of Florida*

E-mails:[1] choreno@.ufl.edu, [2] asmita11@.ufl.edu, [3] yli@ecel.ufl.edu, [4] taoli@ece.ufl.edu

## Abstract

*Biometric computing is a technique that uses physiological and behavioral characteristics of persons to identify and authenticate individuals. Due to the increasing demand on security, privacy and anti-terrorism, biometric applications represent the rapidly growing computing workloads. However, very few results on the execution characteristics of these applications on the state-of-the-art microprocessor and memory systems have been published so far.*

*This paper proposes a suite of biometric applications and reports the results of a biometric workload characterization effort, focusing on various architecture features. To understand the impacts and implications of biometric workloads on the processor and memory architecture design, we contrast the characteristics of biometric workloads and the widely used SPEC 2000 integer benchmarks.*

*Our experiments show that biometric applications typically show small instruction footprint that can fit in the L1 instruction cache. The loads and stores account for more than 50% of the dynamic instructions. This indicates that biometric applications are data-centric in nature. Although biometric applications work across large-scale datasets to identify matched patterns, the active working sets of these workloads are usually small. As a result, prefetching and large L2 cache effectively handle the data footprints of a majority of the studied benchmarks. Branch misprediction rate is less than 4% on all studied workloads. The IPC of the studied benchmarks ranges from 0.13 to 0.77 indicates that out-of-order superscalar execution is not quite efficient.*

*The developed biometric benchmark suite (BMW) and input data sets are freely available and can be downloaded from http://www.ideal.ece.ufl.edu/BMW.*

## 1. Introduction

Biometric computing is a technique that uses physiological and behavioral characteristics (e.g. face, fingerprint, iris, signature, voice, or gait) of persons to identify and authenticate individuals. Due to the increasing demand on security, privacy and anti-terrorism, biometric applications are being deployed in many areas, such as passport authentication, airport and border control, electronic banking, financial transactions, law enforcement, health and social services. Unlike traditional methodologies (e.g. smart cards, encryption keys and digital signatures), biometric identifiers are inextricably linked to persons themselves and therefore can not be forgotten, counterfeited, or stolen. The trend of biometric growth is reported to be on the rise and biometric industry revenue is expected to grow as high as $4,639 million by 2008 [1]. Clearly, computing systems that can deliver high-performance on the representative biometric applications play an important role on the further growth of biometric computing market.

Despite the widespread usage of biometric applications, their execution characteristics on the state-of-the-art microprocessor and memory systems are largely unknown. In order to ensure good hardware performance on biometric applications, designers need to use the key benchmarks from this application domain for the performance measurement and evaluation. Therefore, there is a clear need for representative biometric workloads and detailed workload characterization of these applications. This paper deals with the collection of benchmarks to characterize the various architectural aspects of different biometric applications, such as handwriting, fingerprint, face, voice and gait recognition, and evaluation of the effectiveness of numerous architectural features, such as trace cache, out-of-order and speculative execution, branch prediction and memory system behavior. The goal of this paper is to understand the workload characteristics of important biometric techniques and to provide computer architects and the implementers of biometric software with detailed execution characteristics of the workloads, which may be useful in the hardware/software design trade-offs for the cost-effective biometric computing platforms.

Depending on the context, a biometric system can be either a verification (authentication) system or an identification system. Verification (*Am I whom I claim I am?*) involves confirming or denying a person's claimed identity. In identification, one has to establish a person's identity (*Who am I?*). Each one of these approaches has its own complexities and could probably be solved best by a certain biometric system. As of today, most embedded processors focus on applications used for authentication. For example, Texas Instruments TMS320C600 and TMS320C500 processors are used for the FADT (Fingerprint Authentication Development Tool) [2]. Atmel has launched biometric systems based on Atmel's AT91RM9200 ARM9 micro controller [3]. It has been designed for applications such as locks, and time and

0-7803-9461-5/05/$20.00 ©2005 IEEE

attendance systems or other secure authentication applications. Identification systems are highly computation intensive and the use of general purpose processors is more common for these systems. This paper focuses on characterizing benchmarks for complex identification systems unlike the authentication systems catered by today's embedded processors. Hence we selected a general purpose processor like the Pentium 4 as our baseline architecture due to its state-of-the-art design and popularity.

The paper makes the following contributions:

- First, it proposes a suite of representative biometric applications that could be used to evaluate the design of future processor and memory architecture on this emerging application domain.
- Second, it provides a detailed quantitative workload characterization of important biometric applications. We use hardware performance counters to measure a wide range of architectural features of a Pentium 4 based machine running various biometric applications. We study the basic workload characteristics and examine the efficiency of caching, out-of-order execution, branch prediction and speculative execution. Our analysis is specifically oriented towards the microprocessor and memory architecture efficiency for biometric applications, where very few results have been published so far.

The rest of this paper is organized as follows: Section 2 presents the proposed biometric benchmark suite - BMW. Section 3 describes the experimental methodology used in this study. Section 4 presents the detailed workload characterization results and discusses their architectural implications. In Section 5, we conclude the paper and outline our future work.

## 2. The BMW (BioMetric Workload) Suite

To characterize the architectural aspects of biometric applications, various sets of benchmarks are collected. Currently, the proposed biometric benchmark suite (BMW) contains five applications (i.e., handwriting, fingerprint, face, voice and gait recognition) which cover a variety of the major biometric technique (see Table 1 for benchmark description and S100 input data sets). The datasets we used were collected from several popular databases released by NIST (National Institute of Standards and Technology). We provide three input data sets – S1, S10 and S100 for each benchmark. This section describes the selected applications.

### 2.1 Handwriting

We use *hsfsys2*, a form-based optical character recognition (OCR) system developed by the National Institute of Standards and Technology (NIST) [4, 5]. The *hsfsys2* performs various tasks such as form registration/removal, field isolation/segmentation, character normalization, feature extraction, character classification, and dictionary based post-processing. Each handwriting form contains 34 fields including the digit fields, lower case field, upper case field, and the constitution box. As the first phase of handwriting recognition, the forms are registered or aligned so that fields in the image correspond with the prototypical template of fields. To extract the featured vectors from characters, all characters are represented by 1024 binary pixel values. The Karhunen Loève (KL) [6] transform is applied to these binary pixel vectors to reduce dimensionality, suppress the noise, and produce optimally compact features for classification. The *hsfsys2* system uses Multi-Layer Perceptron [7] method for character recognition. To classify a character, the appropriate eigenvectors and MLP weight matrices are first loaded. Using the eigenvectors, the normalized image is transformed into a feature vector. The feature vector is then presented to the MLP network. The result is assigned classification along with a confidence value. After spell-correct processing, *hsfsys2* system yields the final output as assigned class for each field and its associated confidence as determined by MLP classifier.

### 2.2 Face

We choose Colorado State University's Face Identification Evaluation System [8] as a face recognition application. It provides standard face recognition algorithms and statistical methods for comparing face recognition algorithms. The system includes standardized image pre-processing software and four distinct face recognition algorithms. Preprocessing consists of five steps in converting an original PGM FERET image to a normalized image. These steps include integer to float conversion, geometric normalization, masking, histogram equalization and pixel normalization. Using principle components analysis (PCA) [9] algorithm, feature vectors are formed by concatenating the pixel values from the images. Linear discriminant analysis (LDA) [10] is then applied to form a subspace that is linearly separable between classes. Bayesian intrapersonal/extrapersonal classifier estimates the statistical properties of two subspaces based on a maximum a posteriori (MAP) and maximum likelihood (ML) classifier [11]: one for difference images that belong to the intrapersonal class which originates from two photos of the same subject and another for difference images that belong to the extrapersonal class which originates from two photos of different subjects. During the testing phase, the classifier takes each image of unknown class membership and uses the estimates of the probability distributions as a means of identification.

### 2.3 Fingerprint

We use NIST Fingerprint Image Software 2 (NFIS2) as our biometric fingerprint benchmark. *Nfis2* has been

adopted by the Federal Bureau of Investigation (FBI) and Department of Homeland Security (DHS). The *nfis2* software is organized into seven major packages [12]. Among them, we choose the PCASYS application which is a neural network based fingerprint pattern classification system.

The PCASYS performs various tasks such as segmentation, image enhancement, ridge-valley orientation detection, feature set transform, probabilistic and multiple-layer neural network classification. The segmentor first reads the input fingerprint image (8-bit grayscale, 512×480 pixels) and cuts a rectangular region of the input image. The image enhancement routine enhances the segmented fingerprint image by snips out a sequence of squares each of size 32×32 pixels, with the snipping positions spaced 16 pixels apart in each dimension to produce overlapping. Ridge-valley orientation detector finds the local orientation of the ridges and valleys of the finger surface image, and produces an array of regional averages of these orientations. Feature set transformation performs a linear transform to the orientation array. The Probabilistic Neural Network (PNN) algorithm classifies an incoming feature vector. Additionally, the pseudo-ridge tracing step takes a grid of ridge orientations of the incoming fingerprint and traces pseudo-ridges, which are trajectories that approximately follow the flow of the ridges. As the final processing step, PCASYS takes the outputs of the neural network classifier and the auxiliary pseudo-ridge tracer, and makes the decision as to what class, and confidence, to assign to the fingerprint.

**Table 1. Benchmark Description**

| Software Package | Program | Description | Input Data Set (S100) | # Retired Instructions (Billions) |
|---|---|---|---|---|
| Hand Writing | *hsfsys2* | Use Multi-Layer Perceptron (MLP) classification to identify handwriting | NIST Special Database 19(SD19) contains the full page binary image of 3,699 Handwriting Sample Form(HSF) and 814,255 segmented handprinted digit and alphabetic characters form those form | 1,147 |
| Face | *csuFaceIdEval.5.0* | The standard statistical methods used for comparing face recognition algorithms | The original Facial Recognition Technology (FERET) Database (released in 2001) consists of 14,051 grayscale images | 534 |
| Finger Print | *nfis2* | Neural network based fingerprint pattern classification system (PCASYS) | A set of 2,700 WSQ compressed grayscale fingerprint images | 254 |
| Voice | *Sphinx3-0.5* | Voice recognition system with an acoustic trainer, text recognition decoder and phoneme recognition decoder | CMU Microphone array database and census database(AN4) | 324 |
| Gait | *GaitBaseline V1.7* | Identification of people from gait | The USF-NIST data set consists of 1,870 sequences from 122 subjects spanning 5 covariates | 484 |

## 2.4 Voice

The voice recognition system we use is Sphinx-3 [13, 14], which is one of the series of Sphinx system developed by CMU. It includes both an acoustic trainer and various decoders, i.e., text recognition, phoneme recognition, N-best list generation, etc. A microphone converts the acoustic vibrations into an analog signal. This analog signal is then filtered to eliminate the frequency components of the signal. The filtered signal is then digitized using a sampling and quantization phase. The digitized waveform is then partitioned into fixed-duration time-slices, frames. In the encoding phase, audio signals are compressed and yield a stream of feature vectors. Each signal transformed from time domain to the frequency domain with fixed loop bounds. The Hidden Markov Model, HMM [15] is used as the primary algorithm to recognize the voice. An HMM is a graph of states with arcs weighted by transition probabilities between the states, and recognition is performed by determining the most probable path through the HMM graph corresponding to a given input sequence of frames. Sphinx-3 performs recognition with the help of a dictionary broken down into four probabilistic models, i.e., phone model, acoustic model, language model and pronunciation dictionary model. At runtime, Sphinx-3 finds the highest probability path in a given input sequence of frames using a beam search. As a result of that, beam search yields a set of candidate frames and then Sphinx-3 selects the candidate with the highest probability and sequentially retraces its path to recreate the constructed sentence.

## 2.5 Gait

Gait recognition systems use the video-sequence footage of a walking person to measure several different movements of each articulate joint. It is input intensive and

computationally expensive. Gait recognition algorithm [16] consists of four-parts, namely, define bounding box, silhouette extraction, gait period detection and similarity computation.

As the first part, gait algorithm semi-automatically defines bounding boxes around the moving person in each frame of a sequence. Prior to extracting the silhouette, a background model of the scene is built. In the first pass through a sequence, the software computes the mean and the covariances of the RGB values at each pixel location. Based on the Mahalanobis distance [17, 18], pixels are classified into foreground or background. Gait period detection is estimated by such methods that count the number of foreground pixels in the silhouette in each frame over time and compute the median of the distances between minima. Using those methods, two kinds of estimated gait cycles are generated and the average of these two median would be the final estimated gait period. The output from the gait recognition algorithm is a complete set of similarity scores between all gallery and probe gait sequences. Similarity scores are computed by spatial-temporal correlation.

Table 1 summarizes the collected benchmarks as well as their default input data sets (S100). We provide three different input data sets (i.e. S1, S10, and S100) for each benchmark. The S10 data sets were created by scaling down the S100 data sets with a factor 10. Similarly, the S1 data sets were made by scaling down the S10 data sets with a factor 10.

# 3. Experimental Setup

This section describes our experimental infrastructure, including the configuration of our machine, the microarchitecture features of the Pentium 4 processor and the methodology for collecting and analyzing hardware counter data. We choose Pentium 4 as our baseline architecture due to its state-of-the-art design and popularity. The use of performance counters allows us to examine the characteristics of entire program execution because these programs are fairly long-running.

## 3.1 Machine Configuration

Our system consists of a 3.0GHz Pentium 4 (Prescott) processor, populated with 1GB main memory (Samsung DDR2-SDRAM, 400MHz). The machine runs Red Hat Linux 9.0 kernel version 2.4.26. The system is configured with an 80GB Seagate 7200.7 SATA hard disk that stores the biometric datasets. The benchmarks have been compiled using the gcc compiler for Linux. All biometric benchmarks are executed to completion. To contrast the execution characteristics of biometrics workloads and the well known benchmark suite, we run all SPEC 2000 integer benchmarks with their reference input datasets. Due to the space limitation, we only present the average statistics of the SPEC benchmarks. For some architecture characteristics, we also show the best and the worst scenarios in the SPEC integer suite.

## 3.2 Pentium 4 Microarchitecture

This section briefly describes the key features of the Intel Pentium 4 microarchitecture and provides the technical background to understand the results we present in Section 4.

The front end of the Pentium 4 micro-architecture fetches and decodes instructions. Its builds the decoded instruction into sequences of μops called traces, which are stored in the execution trace cache. The Pentium 4 trace cache can hold approximate 12,000 μops. The Pentium 4 processors have two areas where branch predictions are performed - in the front end of the pipeline, and at the execution trace cache (the trace cache uses branch prediction when it builds a trace). Front-end BTB (Branch Target Buffer, 4K entries) is accessed on a trace cache miss and smaller Trace-cache BTB (2K entries) is used to detect next trace line. The trace cache BTB, together with the front-end BTB, uses a highly advanced branch prediction algorithm. Static branch prediction will occur at decode time if the front-end BTB has no dynamic branch prediction data for a particular branch. Dynamic branch prediction accuracy is also enhanced by adding an indirect branch predictor. The out-of-order execution engine consists of the allocation, renaming, and scheduling functions. The processor can issue multiple μops per cycle to the next pipeline stage. To exploit the instruction level parallelism (ILP) in the programs, the Pentium 4 microarchitecture provides a very large window of instructions from which the execution units can choose. The Pentium 4 processor has an 8-way, 16KB L1 data cache and an 8-way, 1MB, write-back L2 unified cache with 128 bytes/cache line.

## 3.3 Pentium 4 Performance Counters

We used the Pentium 4 hardware counters to measure architectural events [19]. The Pentium 4 performance counting hardware includes 18 hardware counters that can count 18 different events simultaneously in parallel with pipeline execution. The 18 Counter Configuration Control Registers (CCCRs), each associated with a unique counter, configure the counters for specific counting schemes such as event filtering and interrupt generation. The 45 Event Selection Control Registers (ESCRs) specify the hardware events to be counted and some additional Model Specific Registers (MSRs) for special mechanisms like replay tagging. These counters collect various statistics including the number and type of retired instructions, mispredicted branches, cache misses etc. We used a total of 59 event types for the data presented in this paper.

# 4. Experimental Results

This section presents a detailed characterization of the Pentium 4 processor running the proposed biometric

benchmark suite. We examine benchmark basic characteristics, cache, TLB and memory system behavior, and superscalar execution.

### 4.1 Benchmark Basic Characteristics

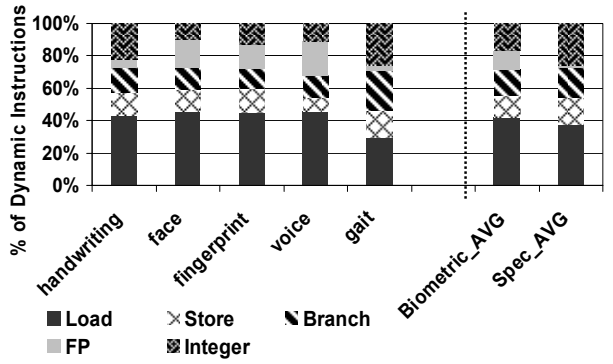


**Figure 1. Dynamic Instruction Mix**

Figure 1 shows the dynamic instruction mix of the biometric applications. On the average, the load, store, branch, floating point, and integer instructions account for 41.9%, 13.5%, 16.1%, 12.1% and 16.4 % of the dynamic instructions respectively. On benchmarks *handwriting*, *face, fingerprint* and *voice*, loads comprise above 40% the dynamic instructions. These applications tend to find or generate information by working on large data sets. The proportion of store instructions ranges from 8.6% on *voice* to 16.51% on *gait*. The studied biometric workloads show large variation in floating point instruction distribution from 3.14% on *gait* to 21.15% on *voice*. And also in Figure 1 and other figures, we add average SPEC 2000 integer benchmark data to compare the biometric benchmark data.

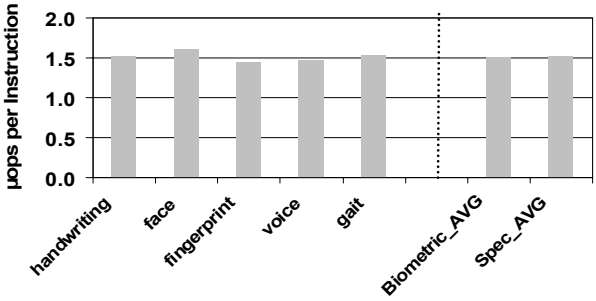


**Figure 2. µops per Instruction**

In order to improve the efficiency of the superscalar execution and the parallelism of the program, each x86 instruction is translated into one or more µops inside the Pentium 4 processor. Typically, a simple instruction is translated into around one to three µops. Figure 2 shows the average number of µops executed per instruction for each of the biometric benchmark. The range is from 1.44 to 1.61, with an average around 1.51. Compared with SPEC benchmarks, biometric applications show similar number in µops per instruction.

### 4.2 IPC and µPC

The instructions-per-cycle (IPC) metric indicates how efficiently a microprocessor performs its functions. Using the Pentium 4 events that count the number of cycles, number of instructions retired and number of µops retired during the measurement period, we computed the IPC and µops per cycle (µPC) metrics for the biometric workloads (as shown in Figure 3). The biometric application IPC ranges from 0.13 to 0.77, with an average around 0.47. A lower IPC can be caused by an increase in cache misses, branch mispredictions, or pipeline stalls in the CPU. The IPCs are low on benchmarks *face* and *voice* due to the excessive data cache misses. On these two benchmarks, large data structures (e.g. HMM) are first created and then intensively accessed during the computation. The memory references to these data structures cause high miss rates due to their poor locality. Interestingly, the benchmark *gait* shows low IPC despite of its good cache and branch prediction performance.

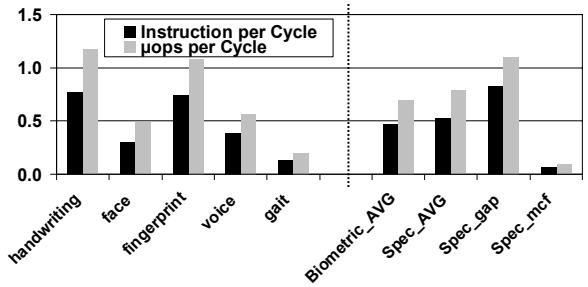


**Figure 3. IPC vs. µPC**

We believe that the low IPC stems from the limited ILP inherent to this benchmark. The µPC ranges from 0.2 to 1.17, with an average around 0.7. Only two benchmarks (*handwriting* and *fingerprint*) achieve more than one µops per cycle. Figure 3 shows that compared with SPEC benchmarks, biometric applications show slightly lower IPC and µPC. The greatest IPC (*fingerprint)* that the processor can yield on biometric applications is similar to that on the SPEC benchmarks (*gap*). The observed lowest IPC (*gait*) is still better than the worst case (*mcf*) of SPEC benchmarks.

### 4.3 Trace Cache

As the front end, the Pentium 4 trace cache sends up to 3 µops per cycle directly to the out-of-order execution engine, without the need for them to pass through the decoding logic. Only when there is a trace cache miss does the front end fetches x86 instructions from the L2 cache. There are some exceedingly long x86 instructions (e.g., the string manipulation instructions) that decode into hundreds of µops. For these long instructions, the Pentium 4 processor fetches µops from a special µops ROM that stores the canned µops sequence. Figure 4 shows the proportion of the µops fetched from the L2 cache, the trace cache, and the µops ROM respectively. As can be seen, a dominant fraction (97.22%) of the µops is supplied by the trace

cache. The L2 cache contributes to less than 0.17% of the µops on most of the benchmarks. This indicates that the Pentium 4 trace cache is highly efficient in providing the µops to the rest of the pipeline.

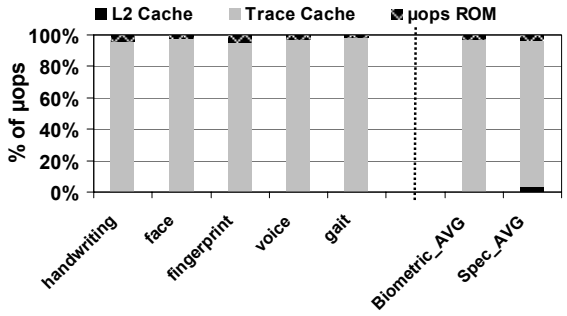


**Figure 4. Source of the µops**

The trace cache operates in two modes: deliver mode and build mode. The deliver mode is the mode in which the trace cache is feeding stored traces to the execution logic to be executed. This is the mode that the trace cache normally runs in. When there is a trace cache miss, the trace cache goes into build mode. In this mode, the front end fetches x86 instructions from the L2 cache, translates into µops, builds a trace segment with it, and loads that segment into the trace cache to be executed. Figure 5 shows the percentage of non-sleep cycles that the trace cache is delivering µops from the trace cache, vs. decoding and building traces. As can be seen, the utilization of the trace cache is extremely high except on the benchmark *fingerprint*. The trace cache BTB yields high misprediction rate on the benchmark *fingerprint*. After branch outcomes are resolved, the speculatively built traces have to be squashed and rebuilt again. Figure 5 shows that the fraction of trace cache deliver mode on biometric application is higher than that on SPEC benchmarks.

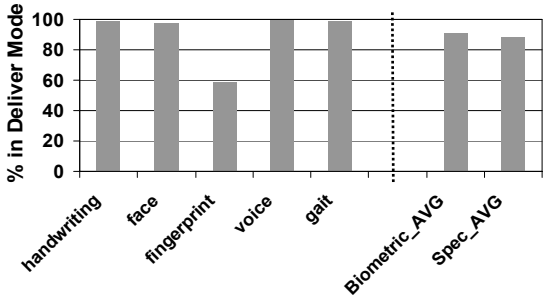


**Figure 5. Trace Cache: % of Cycles in Deliver Mode (fraction of all non-sleep cycles that the trace cache is delivering µops from vs. decoding and building traces)**

## 4.4 Cache Misses

Figure 6 presents the counts of cache misses per 1000 instruction retired. We see that instruction related cache misses are nearly fully satisfied by the trace cache. In most cases the trace cache misses are so small that they don't even shown on the scale used in Figure 6. Data cache miss ratios are higher because the data footprint is much larger than the instruction footprint. For example, the benchmarks *face* and *voice* can cause more than 32 L1 data cache misses on every thousand instructions executed. On the average, the studied biometrics applications generate 16.69 L1 cache misses per thousand retired instructions. Note that this number is likely to increase as (1) the size of the biometric databases grows and (2) the analysis methods get more complicated. Figure 6 shows that in the worst case (*mcf*), SPEC benchmarks yield a cache miss rate 6.5 times higher than the worst case of biometric workloads. On the average, the L1 D-cache miss rate on SPEC benchmarks is three times higher than that on biometric workloads.

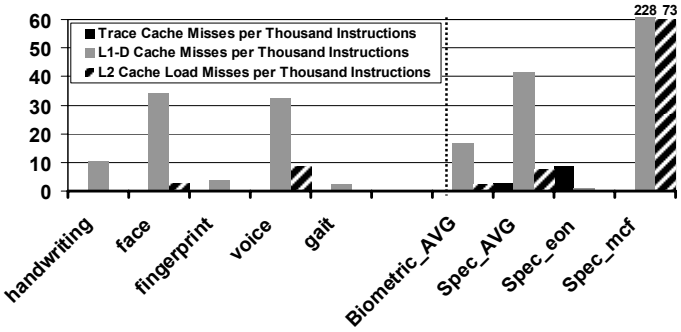


**Figure 6. Cache Misses per Thousand Instructions**

We found that the L1 data cache misses on 3 out of the 5 studied benchmarks can be nearly fully satisfied by the L2 cache. The Pentium 4 processors use automatic hardware prefetch to bring cache lines into the unified L2 cache based on prior reference patterns. Prefetching is beneficial because many accesses to the biometric databases are sequential, and thus, predictable. With prefetching, the L2 cache can handle the working sets of most of the studied benchmarks. Interestingly, the benchmarks (*face* and *voice*) with the highest L1 data cache misses also have the highest L2 misses, implying their poor data locality. Due to the irregular memory access patterns and poor locality, data accesses on these two benchmarks are difficult to absorb, even for the 1MB, 8-way set associative L2 cache with prefetching, which suggests that either larger L2 caches or better prefetching scheme could be beneficial for them. Figures 3 and 6 show a fairly strong correlation between the L2 misses and IPC, indicates that the L2 miss latency is more difficult to be completely overlapped by out-of-order execution.

The Pentium 4 L2 cache uses write-back policy. Cache lines may be in one of four sates: modified (M), exclusive (E), shared (S) or invalid (I). The Pentium 4 counters allow us to monitor the MESI state of an L2 cache line on an access to the L2 cache. Accesses to invalid lines correspond to cache misses, while accesses to lines in other states correspond to hits to an L2 line found in that state, before any modifications due to that access are made.

Figure 7 shows the percentages of L2 accesses broken down by MESI states. Note that these references result in misses in the L1 cache. As expected, we see that a large fraction of accesses that hit in the L2 cache are to exclusive and modified cache lines. The exclusive state is heavily utilized for loads in the Pentium 4 processor. The high percentage of load hits to modified lines indicates that the processor reads data in the same line as it has recently written.

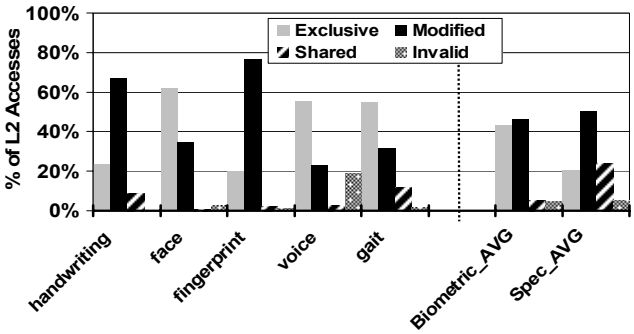


**Figure 7. MESI Sates of L2 Line on L2 Accesses**

## 4.5 TLB Misses

The Pentium 4 processor uses separate TLB (Translation Lookaside Buffer) to translate the virtual address into physical address for instruction and data accesses. Figure 8 presents the ITLB and DTLB miss rates across the studied benchmarks. The ITLB miss rates are well below 2.0% on most benchmarks. Benchmark *gait* yields higher ITLB miss rates than others. This is due to the fact that *gait* has a very large percentage of branches in its instruction mix (as seen from Figure.1). Figure 8 also shows that most of the DTLB accesses can be handled very well by the Pentium 4 processor. Nevertheless, benchmarks *voice,* which exhibits poor data locality, yields high (2.67%) DTLB miss rates. Figure 8 also presents the best (*eon*) and the worst (*mcf*) cases of DTLB miss rates on SPEC benchmarks. As can be seen, compared with SPEC benchmarks, biometrics applications show better DTLB but worse ITLB performance.

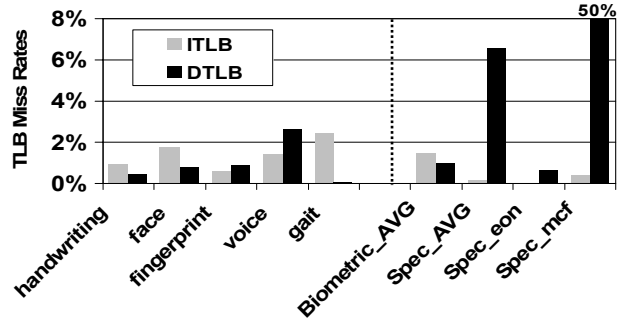


**Figure 8. TLB Miss Rates**

We further characterize the DTLB misses on load and store operations. Figure 9 demonstrates that the DTLB miss rate of *voice* on load operation is higher than that on store operation. Inversely, DTLB miss rate of *fingerprint* on store operations is higher than that on load operation.

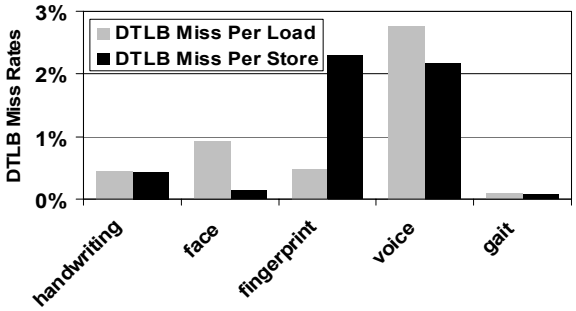


**Figure 9. DTLB Miss Rates on Load and Store**

## 4.6 Load Replays

Given that a large percentage of dynamic instructions are loads and stores, we investigate the efficiently of the processor in handling the load and store operations next.

In an out-of-order-execution processor, stores are not allowed to be committed to permanent machine state (the L1 data cache, etc.) until after the store has retired. With the very deep pipeline of the Pentium 4 processor it takes many clock cycles for a store to make it to retirement. Often loads must use the result of one of these pending stores. The Pentium 4 processor uses a store-to-load forwarding technique to enable certain memory load operations (loads from an address whose data has just been modified by a preceding store operation) to complete without waiting for the data to be written to the cache.

Memory Order Buffer (MOB) acts as a separate schedule and dispatch engine for data loads and stores and also temporarily holds the state of outstanding loads and stores from dispatch until completion. There are size and alignment restrictions for store-to-load forwarding cases to succeed, and when a restriction is not observed, the memory load operation stalls. Later, the load operation replays in the MOB. The MOB load replays event, indicates that store-to-load forwarding restrictions are not being observed.

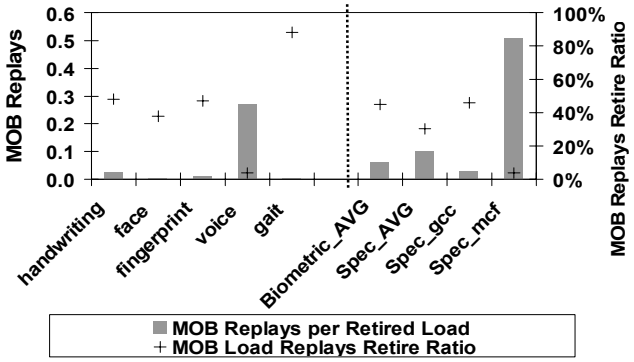


**Figure 10. Load Replays**

Figure 10 shows the average number of MOB replays per retired load. One can see that the numbers are small for most of the benchmarks. There is significant store-to-load forwarding confliction on the benchmarks *voice* due to its irregular memory access patterns. We also plot the percentage of the MOB replayed loads that reach the retirement status and find the numbers vary significantly across different benchmarks. For example, *gait* shows higher MOB replayed load retire ratios than the others. This is because the inherent ILP in benchmark *gait* is low.

### 4.7 Branches and Branch Prediction

Figure 11 presents the fraction of branches that belongs to conditional branches, indirect branches, calls and returns. Conditional branches, ranging from 47.03% (*gait*) to 92.90% (*fingerprint*) of the dynamic branches, dominate the control flow transfers in the biometric applications. Indirect branches account for more than 22.69% of the dynamic branches on benchmarks *gait*. On the average, conditional branches, indirect branches, call, and return contribute to the 78.99%, 8.65%, 6.19% and 6.18% of the total dynamic branches. Figure 11 shows that compared with the SPEC benchmarks, biometric applications show higher ratio of conditional branches in their control flow transfer instructions.

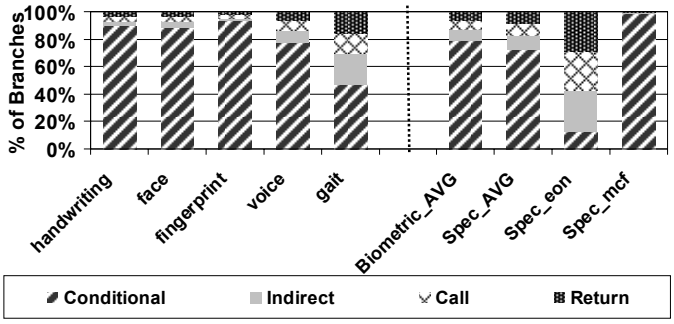


**Figure 11. Dynamic Branch Mix**

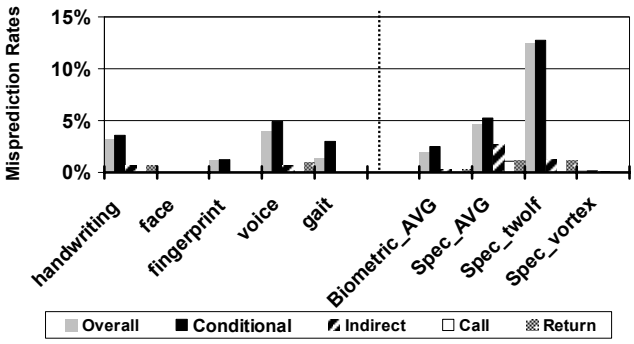


**Figure 12. Branch Misprediction Rates**

Figure 12 shows the branch misprediction rates on the biometric applications. Due to the advanced branch prediction schemes, the overall branch misprediction rates are very low, except *handwriting*(3.23%) and *voice*(4%). Indirect branch misprediction rates are also low on the studied benchmarks. The calls and returns can be predicted accurately by the 16 entries return address stack. The accuracy of conditional branch prediction largely determinates the overall branch prediction performance. Figure 12 shows that SPEC benchmarks yields higher misprediction rates than biometric benchmarks.

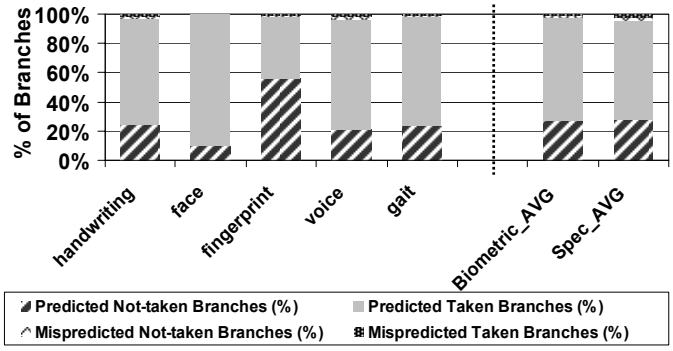


**Figure 13. Dynamic Branch Direction**

Figure 13 presents the branch directions (i.e., taken or not-taken) for both correctly predicted and incorrectly predicted conditional branches. As can be seen, a significant portion of the conditional branches are taken branches, implying there are abundant loop structures within the studied biometric applications. Strongly taken or strongly not-taken (i.e. strongly-biased) branches can be easily predicted by the Pentium 4 branch predictors. Even if the branch information can not be found in its dynamic branch predictors, the Pentium 4 can still predict the strongly-biased branches accurately using its static branch prediction scheme. The mispredicted branches are weakly-biased in nature by showing the equal distribution on the taken and non-taken directions. Previous studies [20] show that the weakly-biased branches, when intermingling with the strongly-biased branches, can increase branch aliasing and degrade the prediction accuracy. It will be interesting to further investigate the branch aliasing caused by the weakly-biased branches.

### 4.8 Speculative Execution

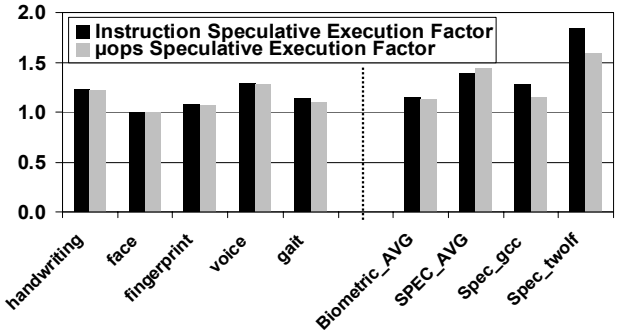


**Figure 14. Speculation Factor**

To reach high performance, the Pentium 4 machine fetches and executes instructions along the predicted path until the branch is resolved. In case there is a branch misprediction, the speculatively executed instructions along the mispredicted path are flushed. The speculative execution factor or the ratio of the total number of instructions decoded to the total number of instructions retired quantitatively captures how aggressively the processor executes the speculated instructions.

Figure 14 shows the speculative execution factors for instruction and µops on the biometrics applications. On the average, the processor decodes 15% more instructions than it retires. Higher speculation factors are observed on benchmarks *handwriting* and *voice*. Note that there is a fairly strong correlation between the branch prediction accuracy and the speculative execution factor on these programs. Due to the use of deeply pipelined design to reach high operation clock frequency, the accuracy of branch prediction plays an important role on Pentium 4 pipeline performance. Biometric benchmarks with higher mispredicted branches per instruction have higher speculated instructions, indicating these applications can further benefit from more accurate branch prediction. Figure 14 shows that on the average, the speculative factor on SPEC benchmarks is higher than biometrics workloads.

### 4.9 Phase Behavior

Recent computer architecture research has shown that program execution exhibits phase behavior, and these behaviors can be seen even on the largest of scales [21]. Program phases can be exploited to design adaptive microarchitecture, guide feedback compiler optimization and reduce simulation time. To reveal the phase behavior of biometric applications, we sampled performance counters at a time interval of 0.1 second. Figure 15 shows the sampled IPC during program execution. As can be seen, the studied biometric applications show heterogeneous phase behavior. For example, benchmark *gait* shows periodic spikes where program execution yields high IPC. Although benchmark *face* shows irregular behavior during initialization stage, its phase behavior is highly predictable for the majority of program execution. Benchmarks *handwriting*, *fingerprint* and *voice* exhibit irregular and unpredictable phase behavior during the entire program execution.

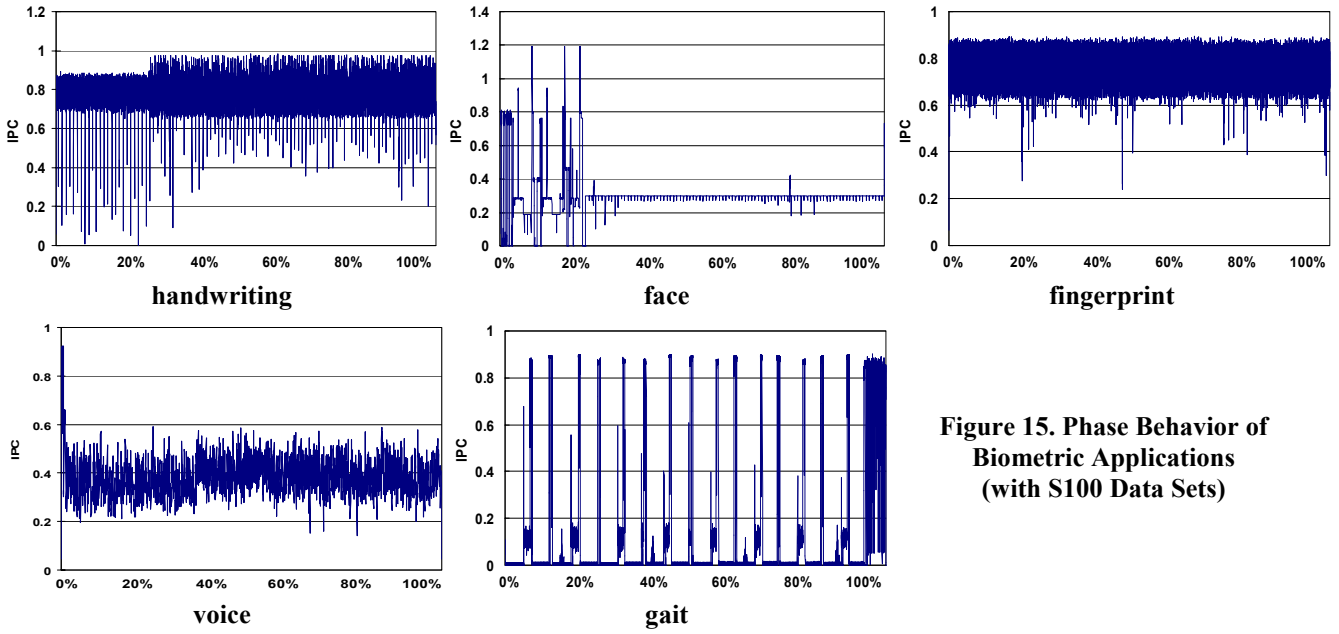


**Figure 15. Phase Behavior of Biometric Applications (with S100 Data Sets)**

### 4.10 Performance Variability under Different Input Data Sets

To understand the implication of using different data sets to evaluate the performance of biometric applications, we run each benchmark with S1 and S10 data sets. Table 2 summarizes performance variability of biometric workloads under different input data sets. For the purpose of comparison, we also include the results of S100 data sets. As we expect, the scales of input data sets can have significant impact on the miss rates of L1 data cache and L2 cache. For instance, on benchmark *face*, the L1 data cache misses per thousand instructions increase from 19 to 34 when the input data set is scaled from S10 to S100. In some cases, small data sets perform worse than large data sets due to cold start effect.

**Table 2. Performance Variability under Different Input Data Sets**

| Metrics | | Hand writing | Face | Finger print | Voice | Gait |
|---|---|---|---|---|---|---|
| IPC | S1 | 0.75 | 0.38 | 0.75 | 0.46 | 0.18 |
| | S10 | 0.77 | 0.31 | 0.75 | 0.38 | 0.28 |
| | S100 | 0.77 | 0.3 | 0.75 | 0.38 | 0.13 |
| Trace Cache Misses per 1000 Instructions | S1 | 0.23 | 0.08 | 0.04 | 0.51 | 0.5 |
| | S10 | 0.22 | 0.11 | 0.04 | 0.13 | 0.07 |
| | S100 | 0.13 | 0.07 | 0.03 | 0.12 | 0.05 |
| L1-D Cache Misses per 1000 Instructions | S1 | 12.58 | 19.48 | 3.62 | 23.4 | 0.71 |
| | S10 | 12.75 | 19.86 | 3.66 | 31.37 | 0.72 |
| | S100 | 10.37 | 34.3 | 3.78 | 32.56 | 2.43 |
| L2 Cache Load Misses per 1000 Instructions | S1 | 0.53 | 2.11 | 0.08 | 5.39 | 0.08 |
| | S10 | 0.52 | 2.01 | 0.08 | 8.74 | 0.07 |
| | S100 | 0.31 | 2.77 | 0.08 | 8.48 | 0.11 |
| I-TLB Miss Rate | S1 | 1.18% | 2.2% | 0.2% | 0.51% | 2.28% |
| | S10 | 1.16% | 1.67% | 0.21% | 1.03% | 3.31% |
| | S100 | 0.96% | 1.79% | 0.62% | 1.4% | 2.47% |
| D-TLB Miss Rate | S1 | 0.82% | 2.97% | 0.92% | 2.65% | 0.03% |
| | S10 | 0.83% | 2.39% | 0.92% | 2.88% | 0.05% |
| | S100 | 0.45% | 0.78% | 0.93% | 2.67% | 0.09% |
| Trace Cache: % of Cycles in Deliver Mode | S1 | 86.25% | 74.15% | 97.26% | 92.98% | 23.1% |
| | S10 | 87.82% | 75.55% | 97.19% | 96.26% | 34.93% |
| | S100 | 98.81% | 97.18% | 58.58% | 99.64% | 99.03% |
| Overall Branch Misprediction Rates | S1 | 1.81% | 0.2% | 1.19% | 2.87% | 0.19% |
| | S10 | 1.8% | 0.11% | 1.20% | 3.69% | 0.07% |
| | S100 | 3.23% | 0.04% | 1.17% | 4.00% | 1.41% |
| Conditional Branch Misprediction Rate | S1 | 1.91% | 0.19% | 1.26% | 3.54% | 0.52% |
| | S10 | 1.90% | 0.21% | 1.27% | 4.54% | 0.21% |
| | S100 | 3.55% | 0.04% | 1.23% | 4.89% | 2.99% |
| Instruction Speculative Execution Factor | S1 | 1.14 | 1 | 1.08 | 1.25 | 1.01 |
| | S10 | 1.14 | 1 | 1.08 | 1.28 | 1.01 |
| | S100 | 1.23 | 1 | 1.07 | 1.29 | 1.14 |

## 5. Conclusions

Although biometric applications represent a rapidly growing security computing market, their implications on the computer architecture design are still unknown. In order to apply the quantitative approach in computer architecture design and performance evaluation, there is a clear need for representative biometric applications and detailed workload characterization of these applications.

This paper proposes BMW, a group of programs representative of biometric workloads. These programs include popular biometrics identifications used for handwriting, face, fingerprint, voice and gait recognitions. This paper studies the characteristics of biometric workloads and evaluates the effectiveness of numerous architectural features, such as trace cache, out-of-order and speculative execution, branch prediction and memory system behavior.

We find that the instruction footprints of biometric applications are typically small and can fit in the L1 instruction cache. Loads and stores account for 55% of dynamic instructions executed. This indicates that biometric workloads are data-centric. Prefetching and the large L2 cache can efficiently handle the working sets of a majority of the studied benchmarks. Compared with SPEC benchmarks, the studied biometric applications show better performance in terms of data cache, D-TLB miss rates and branch misprediction rate. The IPC of the studied benchmarks ranges from 0.13 to 0.77. This indicates that out-of-order superscalar execution is not quite efficient.

We believe that the development of representative biometric benchmarks and the workload characterization of

these benchmarks will help in understanding the design issues of processor architecture as well as evaluating computer system performance on these emerging workloads. In the future works, we will extend the current biometric benchmark suite with new applications such as iris, palm recognitions. It is also very interesting to study and compare the performance of biometric applications on other processor paradigms such as SMT (Simultaneous Multi-Threading) and CMP (Chip Multi-Processor). Additionally, we will explore the integrated software/hardware techniques to optimize the performance of biometric applications.

## References


[1] Biometrics Market and Industry Report (2004 – 2008), International Biometric Group, 2004.

[2] DSP for Smart Biometric Solutions, White Paper, Texas Instruments, 2003.

[3] http://www.arm.com/iqonline/news/partnernews/6420.html.

[4] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NIST Internal Report 5469 and CD-ROM, Jul, 1994.

[5] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System(R2), NIST Internal Report 5959, Jan, 1997.

[6] P. J. Grother, Karhunen Loève Feature Extraction for Neural Handwritten Character Recognition, NIST Internal Report 4824, April 1992, and in Proceedings of Applications of Artificial Neural Networks III, Vol. 1709, pp. 155-166. SPIE, Orlando, April 1992.

[7] C. L. Wilson, J. L. Blue, O. M. Omidvar, The Effect of Training Dynamics on Neural Network Performance, NIST Internal Report 5696, August 1995.

[8] R. Beveridge, D. Bolme, M. Teixeira and B. Draper, The CSU Face Identification Evaluation System User's Guide, V5.0, Colorado State University, May, 2003.

[9] M. A. Turk and A. P. Pentland, Face Recognition Using Eigenfaces, In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pages 586 – 591, June 1991.

[10] W. Zhao, R. Chellappa, and A. Krishnaswamy, Discriminant Analysis of Principal Components for Face Recognition, In Wechsler, Philips, Bruce, Fogelman-Soulie, and Huang, Editors, Face Recognition: From Theory to Applications, pages 73–85, 1998.

[11] B. Moghaddam, C. Nastar, and A. Pentland, A Bayesian Similarity Measure for Direct Image Matching, ICPR, B:350–358, 1996.

[12] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. McCabe, and S. Janet, User's Guide to NIST Fingerprint Image Software 2,Oct.2004.

[13] K. K. Agaram, S. W. Keckler, and D. C. Burger, A Characterization of Speech Recognition on Modern Computer Systems, In Proceedings of the 4th IEEE Workshop on Workload Characterization, 2001.

[14] http://cmusphinx.sourceforge.net/html/cmusphinx.php.

[15] D. Robert and E. Woodland, A Hidden Markov Model-based Trainable Speech Synthesizer, Computer Speech and Language, (13:223-241), 1999.

[16] S. Sarkar, P. J. Phillips, Z. Liu, I. Robledo, P. Grother, and K. Bowyer, The Human ID Gait Challenge Problem: Data Sets, Performance, and Analysis", IEEE Transactions on Pattern Analysis and Machine Intelligence, Aug., 2003.

[17] P. J. Phillips, S. Sarkar, I. Robledo, P. Grother, and K. Bowyer, Baseline Results for the Challenge Problem of Human ID using Gait Analysis, In Proceedings of the International Conference on Automatic Face and Gesture Recognition, pages 137–142, 2002.

[18] P. J. Phillips, S. Sarkar, I. Robledo, P. Grother, and K. Bowyer, The Gait Identification Challenge Problem: Data Sets and Baseline Algorithm, In Proceedings of the International Conference on Pattern Recognition, pages 385–388, 2002.

[19] B. Sprunt, The Basics of Performance Monitoring Hardware, IEEE Micro, July-August, page 64-71, 2002.

[20] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference, In Proceedings of the International Symposium on Computer Architecture, 1997.

[21] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, Automatically Characterizing Large Scale Program Behavior , In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.