MITRE

GPU-Accelerated Fingerprint Matching

Scott Bai, Joseph P. Marques, Matthew T. McMahon, and Steven H. Barry, PhD
{sbai, jmarques, mcmahon, sbarry}@mitre.org
The MITRE Corporation, McLean, Virginia

Introduction

Fingerprint matching algorithms compare images to determine whether they represent impressions taken from the same finger or from different fingers. For an input (a *probe*) that may consist of up to 10 fingerprint images, a matching algorithm finds the best match to an identity in a database of identities (the *gallery*); each identity is associated with a set of fingerprint images. As galleries increase in size to tens of millions of identities, matching systems face larger computational workloads, and are required to meet stringent speed and accuracy requirements. As part of an effort to improve speed throughout the matching process, we investigated the use of GPUs to accelerate the FingerCode algorithm described by Jain, et al. (1999).

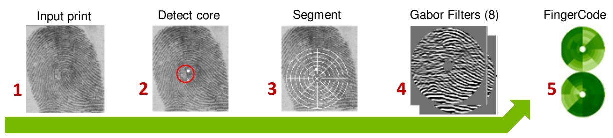


Figure 1: Generation of a FingerCode vector from the source image (Jain, et al., 1999).

FingerCodes

A FingerCode is a 640 byte feature vector extracted from a fingerprint image (see **Fig. 1** above). Two FingerCode vectors can be compared to compute the similarity or degree of match between the corresponding fingerprints.

$$score(p, g_i) = \sqrt{\sum_{j=1}^{fcsize} (p_j - g_{i,j})^2}$$

- Match score is the Euclidean distance between two FingerCode vectors (see formula above). The algorithm is
  - Very parallel, many independent operations due to size of gallery
  - Ideal for GPU acceleration but not as accurate as slower methods (e.g. minutiae matching)
  - Sensitive to fingerprint quality; requires accurate core detection, or some other reference point that is consistent across different impressions of the same finger
- Two fingerprint images may have different orientations; thus, we rotate one of them through 16 possible quantized orientations when comparing them (i.e. rotate through 360˚, in 22.5˚ increments), and keep only the best of the 16 possible scores

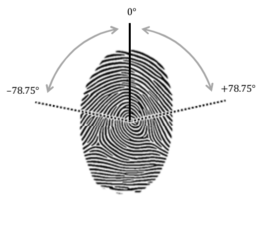


Figure 2: The GPU matcher computes scores for the probe compared to each candidate in the gallery, at seven orientations over a range of ±78.75˚

GPU Approach

- Change vector from 640 bytes to 512 (a power of 2)
  - Simplifies parallel sum implementation for Euclidean distance operation
- Match on only 7 orientations instead of all 16
  - Allows for ±78.75˚ discrepancy between the orientation of the probe vector vs. the gallery vector
  - Avoids testing for rarely seen extreme orientations, reducing false positive rate
- Compute match scores for one probe identity (10 FingerCodes) against all identities resident on the GPU using a single kernel call

Optimizations

- Unroll parallel sum loop to reduce logic overhead
  - See "Optimizing Parallel Reduction in CUDA", NVIDIA whitepaper by Mark Harris, included with NVIDIA CUDA SDK sample projects
- Compare against all 7 orientations concurrently
  - Requires more shared memory and registers per block but increases per-block speed by making better use of threads within each block (fewer idle threads in last iterations of parallel sum)
- Use only 64 threads per block to perform parallel sum on 512 elements
  - See "Better Performance at Lower Occupancy", GTC2010 presentation by Vasily Volkov
  - Improves instruction-level parallelism with multiple independent math operations per thread
  - Uses more registers per thread (79 on GT200)

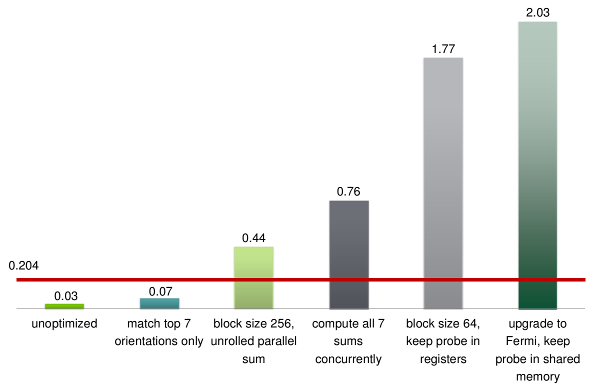


Figure 3: Matching speed in millions of identity-to-identity comparisons per second for a single GPU storing a gallery of 500,000 identities, showing results of successive optimizations from left to right. Red horizontal line indicates baseline speed of a CPU implementation for comparison. All results from a Tesla C1060 except for the rightmost column, which was measured on a Tesla C2050.

Single-GPU Performance

With 500,000 synthetic identities (5 million FingerCodes) pre-loaded per GPU, we achieved:

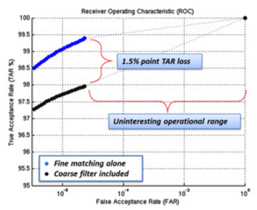
- 2.03 million identity-to-identity comparisons per second on Fermi
- 1.77 million per second on GT200
- Compared to 204,000 comparisons per second on dual quad-core Intel CPUs using OpenMP

FingerCode as Coarse Match Filtering Mechanism

- FingerCodes alone are fast but too inaccurate to be used as a standalone matcher
  - Our version has 59% true acceptance rate (TAR) at 1% false acceptance rate (FAR) on the publically available FVC2004 dataset
- Instead we use FingerCode as a filter in a multi-stage matching system
  - FingerCode algorithm is a main component of a "coarse" match filter
  - The goal of coarse match filter is to eliminate identities unlikely to match the probe
  - After coarse match steps, use more accurate, slower "fine" match algorithm for remaining few percent of gallery that pass the FingerCode matching threshold

Performance Evaluation

- Matcher software resides on a dual quad-core Intel-based computing node connected to a Tesla S1070 containing four GPUs
  - Fine matcher threads and some coarse matcher components execute on CPUs using full gallery stored in main memory
  - FingerCode coarse match step executes on GPUs with gallery in device memory
  - Only one GPU being used currently because CPU memory size cannot accommodate more identities to fill other GPUs with corresponding FingerCodes
- Matcher system was tested using a subset of an anonymized database of 10 million real identities of varying quality, randomly sampled from a larger database
- Note that all accuracy results below were measured using an older but functionally identical FingerCode kernel operating at 760,000 comparisons per second (CPS). The GPUs in the node are capable of 1.77 M CPS (see **Fig. 2**, second column from the right).



	Fine matching alone	Matcher with coarse filter
True acceptance rate at FAR = 10^{-6}	99%	97.75%
Match rate (comparisons per second)	18,000	**625,000**

Figure 4: (Left) Receiver Operating Characteristic (ROC) curve showing extent of TAR loss due to identities being erroneously discarded by coarse filter. (Right) Table showing increased matching speed of a matcher node with corresponding TAR loss at a FAR of 10^{-6}.

Accuracy vs Speed Tradeoff

- Matching rate improves by ~35x with GPU coarse filter
- Use of coarse filter reduces TAR by ~1.25% for same FAR
  - But FingerCode accuracy can be improved with image enhancement/restoration techniques (future work) which may also improve fine match accuracy
  - Image enhancement is one-time cost per image – a relatively low cost when amortized over number of comparisons made against the associated identity

Conclusions

Larger biometric databases require faster matchers. Our GPU approach has demonstrated capability to compare 2 million identities per second. Used as a filtering stage, the FingerCode matcher can substantially improve matcher throughput while avoiding the low accuracy of using FingerCodes alone. This method has shown potential to improve match speeds to near-interactive rates on real-world databases of tens of millions of identities. Furthermore, the high density of processing power provided by the GPUs permits large databases to be searched with fewer compute nodes.

References and Acknowledgements

- A. K. Jain, S. Prabhakar, L. Hong, and S. Pankanti, "FingerCode: A Filterbank for Fingerprint Representation and Matching," 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99) - Volume 2, p. 2187, 1999.
- The U.S. Department of Homeland Security (DHS) sponsored the production of this material under Contract No. HSHQDC-09-D-00001 with The MITRE Corporation. The information contained herein was developed by The MITRE Corporation under its research and development program and DHS funded Contract Nos. TIRNO-99-D-0005 and HSHQDC-09-D-00001 with The MITRE Corporation.