CUDA Accelerated Iris Template Matching on Graphics Processing Units (GPUs)

Carnegie Mellon CyLab — CONFIDENCE FOR A NETWORKED WORLD

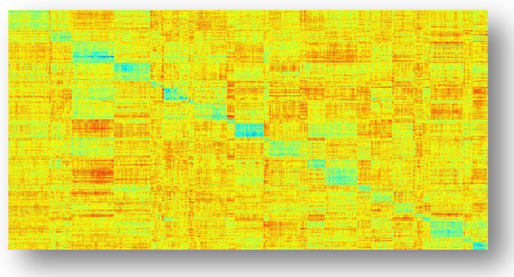
Nicholas A. Vandal
nvandal@ri.cmu.edu

Marios Savvides
msavvid@ri.cmu.edu

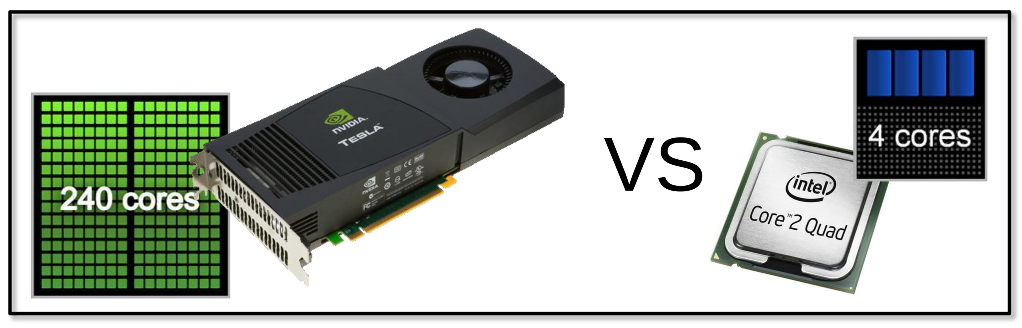
IEEE Fourth International Conference on Biometrics: Theory and Applications and Systems, BTAS 2010

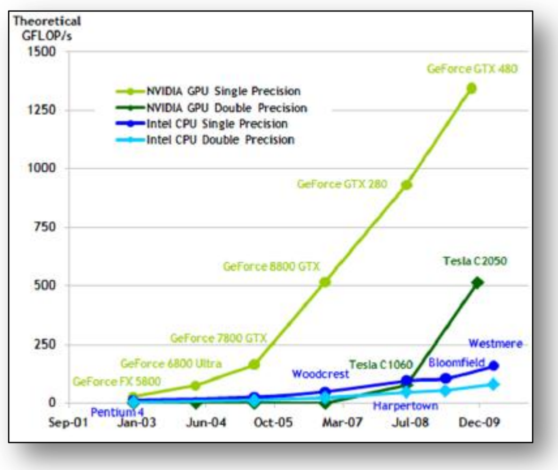
Goals & Motivations

- The iris pattern is considered to be amongst the most reliable for high confidence identification
  - Identification in enormous (national/global) databases
  - Requires 'many-to-many' comparisons; makes even the simplest distance metric computationally expensive
  - Fortunately: "embarrassingly parallel"
  - Distance matrix elements independent



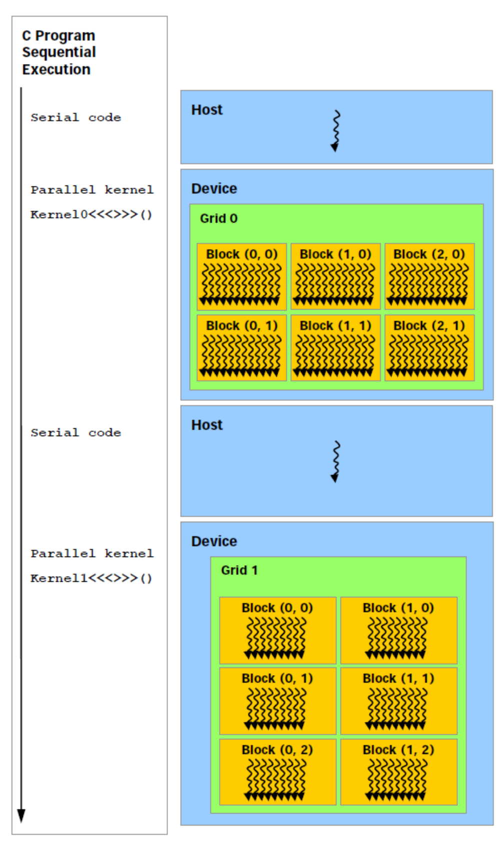
- Standard solution: large CPU-based clusters
  - CPUs designed for general purpose, sequential tasks
  - Non-optimal w.r.t. power, cooling, footprint size & cost
- Better solution: acceleration with off-the-shelf GPUs
  - Market demand for realistic 3D games has evolved the GPU into a highly parallel, multithreaded, many-core processor of tremendous power





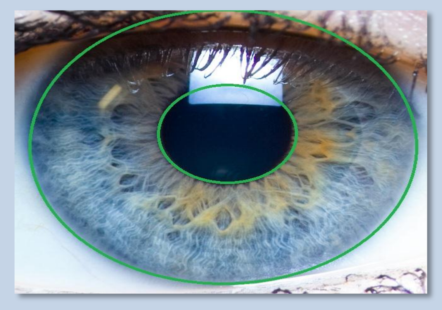
Compute Unified Device Architecture

- Prior to introduction of CUDA (API/architecture) in 2006 General-Purpose Computation on Graphics Hardware (GPGPU) was hard work:
  - Express problems in terms of graphics primitives
- CUDA enables expression of programs in C, C++, Fortran and other high level languages
- Heterogeneous: execute "kernels" on GPU
  - Section of device code that executes in parallel on GPU
  - Unique thread ID, program counter, registers and private local memory
  - Shared global device memory for communication
  - Hierarchy of memory types
  - SIMT model v. SIMD model



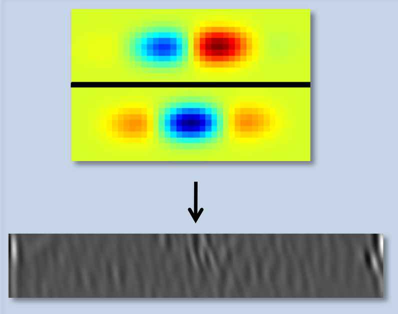
Serial code executes on the host while parallel code executes on the device.

How Iris Recognition Works

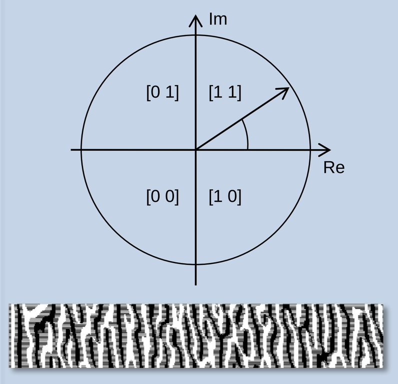


Step 1: Image Capture and Iris Segmentation

Step 2: Normalized Polar Transform of Iris/Mask



Step 3: Convolution with complex valued Gabor to generate filter response

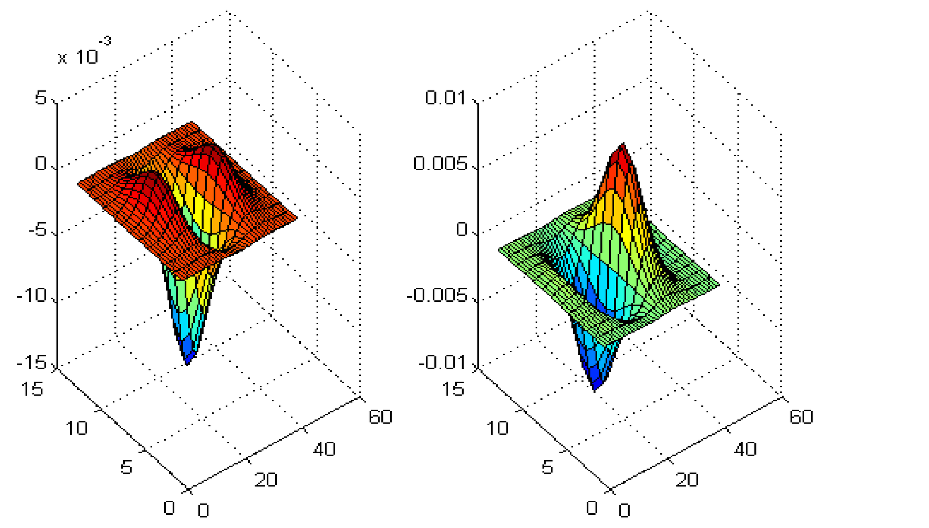


Step 4: Phase Quadrant Encoding of IrisCode

$$HD_{(A,B)} = \frac{\|(Code_A \oplus Code_B) \cap Mask_A \cap Mask_B\|}{\|Mask_A \cap Mask_B\|}$$

Step 5: Fractional 'Best of' Hamming Distance

The most widely employed procedure for feature extraction, pioneered by John Daugman, uses the phase response of 2D Gabor wavelets.



$$g(x,y) = s(x,y)\,w_r(x,y)$$

$$s(x,y) = e^{j(2\pi(u_0x+v_0y)+P)} \quad w_r(x,y) = Ke^{-\left(\pi\left(a^2(x-x_0)_r+b^2(y-y_0)_r\right)\right)}$$

$$(x-x_0)_r = (x-x_0)\cos\theta + (y-y_0)\sin\theta \quad (y-y_0)_r = -(x-x_0)\sin\theta + (y-y_0)\cos\theta$$

In addition to being a component of Daugman's algorithm, the template matching process (select the minimum fractional HD over a range of bitwise horizontal circular shifts) is a common final step of other iris recognition routines.

Implementation Overview

- Naïve implementation: exploit fine grain parallelism
  - Kernel below computes pairwise HD between a probe template and a gallery template determined by a unique two dimensional thread ID

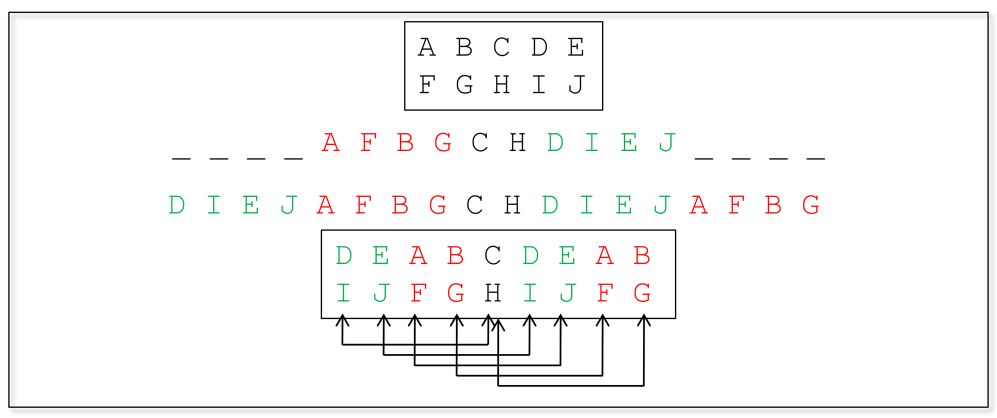
```
__device__  float fractionalMaskedHD(unsigned int* x, unsigned int* y,
unsigned int* maskX, unsigned int* maskY, unsigned int len)
{
   unsigned int nB_c =0;
   unsigned int nB_m =0;
   unsigned int num;
   unsigned int* xEnd = x+len;
   unsigned int *u, *v, *mu, *mv;

   for(u = x, v=y, mu=maskX, mv=maskY; u!=xEnd;)
   {
      num = *mu++ & *mv++;
      nB_m+=__popc(num);
      num = (*u++ ^ *v++) & num;
      nB_c+=__popc(num);
   }

   return __uint2float_rn(nB_c)/__uint2float_rn(nB_m);
}
```

Listing 1: CUDA inline device code that computes single fractional Hamming distance using intrinsic function __popc to count set bits.

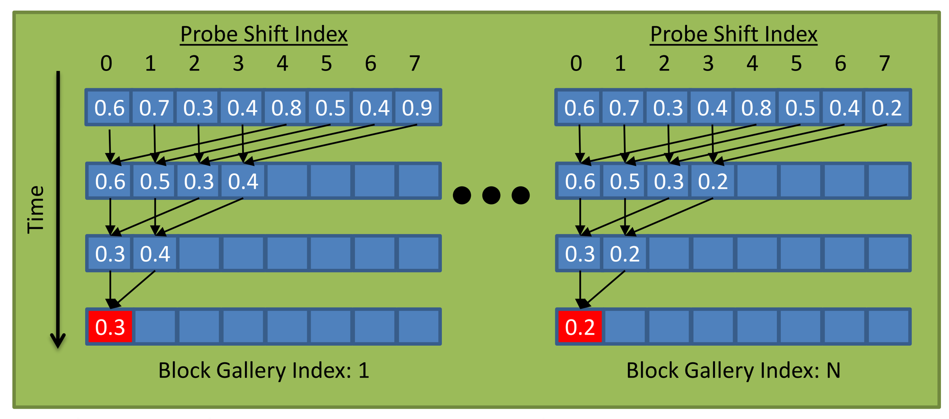
```
__global__ void pdist2_smem_kernel(unsigned int* X, unsigned int* Y,
unsigned int* mX, unsigned int* mY, float D, unsigned int mx , unsigned int my,
unsigned int n)
{
   extern __shared__ unsigned int smem[];
   unsigned int* probeS = (unsigned int*)smem;
   unsigned int* maskS = (unsigned int*)&smem[n];

   unsigned int y =IMUL(blockDim.y , blockIdx.y) + threadIdx.y;
   unsigned int tid = threadIdx.y;

   // Load the probe, which is constant within a given block into shared mem
   blockMemcpy<unsigned int>(probeS ,&X[blockIdx.x * n] , tid , n , blockDim.y);
   blockMemcpy<unsigned int>(maskS ,&mX[blockIdx.x * n] , tid , n , blockDim.y);

   __syncthreads();

   if(y>=my)
      return;

   //Fractional HD
   D[(y*mx) + blockIdx.x] = fractionalMaskedHD(probeS ,&Y[y*n] , maskS ,&mY[y*n] , n);
}
```

Listing 2: CUDA kernel code that utilizes shared memory, for nonshifted pairwise computation of fractional Hamming distance for set of probe and gallery templates.

- Optimizations: kernel is extremely memory bound
  - Exploit memory hierarchy → shared memory and texture cache reduce global memory bandwidth usage/latency
- Rotation invariance:
  - Perform shifts in shared memory and reduce within thread block to find minimal HD for a given probe



Method using cooperative memcpy within a block's shared memory to execute all horizontal shifts from +2 to -2. Assumes column major matrix order with vertically packed bits



Simultaneous multi-row reduction performed in block shared memory used to find minimal fractional HDs over all possible shifts of probe.

Results

Rotation Invariant

	Template Size (bits)	GPU Exec Time HDs (ms)	GPU Exec Time Total (ms)	GPU Calc HD Rate (matches/s)	GPU Total Rate (matches/s)
67,108,864 Matches	(32 X 32) 1024	7576.78	7760.73	8,857,176.08	8,647,235.32
	(64 x 32) 2048	15936.50	16125.07	4,211,016.78	4,161,772.93
	(64 x 64) 4096	46968.48	47161.08	1,428,806.37	1,422,971.29
	(176 x 24) 4224	61575.40	61766.54	1,089,864.81	1,086,492.22
	(64 x 96) 6144	69403.42	69611.80	966,938.91	964,044.31
	(64 x 128) 8192	90406.05	90590.37	742,304.99	740,794.68

	Template Size (bits)	CPU Exec Time (ms)	CPU Matches/sec
Table Lookup	(32 X 32) 1024	127538.63	526,185
	(64 x 32) 2048	234918.5	285,669
	(64 x 64) 4096	454712.95	147,585
	(176 x 24) 4224	569861	117,764
	(64 x 96) 6144	675662.78	99,323
	(64 x 128) 8192	898720.51	74,672
SSE4.2 POPCOUNT	(32 X 32) 1024	80453.74	834,130
	(64 x 32) 2048	138769.04	483,601
	(64 x 64) 4096	236847.72	283,342
	(176 x 24) 4224	317480.8	211,379
	(64 x 96) 6144	342103.57	196,165
	(64 x 128) 8192	443273.51	151,394

	Template Size (bits)	GPU Speedup (HD Only)	GPU Speedup (with memcpy)
Table Lookup	(32 X 32) 1024	16.8	16.4
	(64 x 32) 2048	14.7	14.6
	(64 x 64) 4096	9.7	9.6
	(176 x 24) 4224	9.3	9.2
	(64 x 96) 6144	9.7	9.7
	(64 x 128) 8192	9.9	9.9
SSE4.2 POPCOUNT	(32 X 32) 1024	10.6	10.4
	(64 x 32) 2048	8.7	8.6
	(64 x 64) 4096	5.0	5.0
	(176 x 24) 4224	5.2	5.1
	(64 x 96) 6144	4.9	4.9
	(64 x 128) 8192	4.9	4.9

Non-Rotation Invariant

	Template Size (bits)	GPU Exec Time HDs (ms)	GPU Exec Time Total (ms)	GPU Calc HD Rate (matches/s)	GPU Total Rate (matches/s)
67,108,864 Matches	(32 X 32) 1024	1515.70	1619.76	44,275,742	41,431,253
	(64 x 32) 2048	762.67	850.48	87,991,713	78,907,328
	(64 x 64) 4096	6126.10	6216.88	10,954,574	10,794,626
	(176 x 24) 4224	6684.06	6774.10	10,040,130	9,906,689
	(64 x 96) 6144	11072.07	11164.34	6,061,094	6,011,000
	(64 x 128) 8192	14822.66	14916.39	4,527,452	4,499,002

	Template Size (bits)	CPU Exec Time (ms)	CPU Matches/sec
Table Lookup	(32 X 32) 1024	9530.77	7,041,282
	(64 x 32) 2048	18187.42	3,689,852
	(64 x 64) 4096	41063.11	1,634,286
	(176 x 24) 4224	41594.70	1,613,399
	(64 x 96) 6144	60921.31	1,101,566
	(64 x 128) 8192	79505.10	844,083
SSE4.2 POPCOUNT	(32 X 32) 1024	4282.87	15,669,151
	(64 x 32) 2048	8330.13	8,056,165
	(64 x 64) 4096	16033.12	4,185,641
	(176 x 24) 4224	16227.66	4,135,463
	(64 x 96) 6144	23438.37	2,863,206
	(64 x 128) 8192	30714.91	2,184,896

	Template Size (bits)	GPU Speedup (HD Only)	GPU Speedup (with memcpy)
Table Lookup	(32 X 32) 1024	12.5	11.2
	(64 x 32) 2048	12.0	11.4
	(64 x 64) 4096	6.7	6.6
	(176 x 24) 4224	6.2	6.2
	(64 x 96) 6144	5.5	5.5
	(64 x 128) 8192	5.4	5.3
SSE4.2 POPCOUNT	(32 X 32) 1024	5.6	5.0
	(64 x 32) 2048	5.5	5.2
	(64 x 64) 4096	2.6	2.6
	(176 x 24) 4224	2.4	2.4
	(64 x 96) 6144	2.1	2.1
	(64 x 128) 8192	2.1	2.1

Conclusions

- Achieved rates of 44 million iris template matches/s without rotation invariance. With tolerance to head tilt, 4.2 million matches/s (template size 2048 bits)
- Show a 14X speedup over optimized CPU implementation
- In contrast to other published work, our parallel implementation incorporates shifting for rotation invariance