REALTIME AUTOMATIC FACIAL LANDMARKING USING ACTIVE SHAPE MODELS ON GRAPHICS PROCESSOR UNITS

NICK VANDAL

MS STUDENT

ROBOTICS INSTITUTE, CARNEGIE MELLON UNIVERSITY

NVANDAL@CMU.EDU

CONTENTS

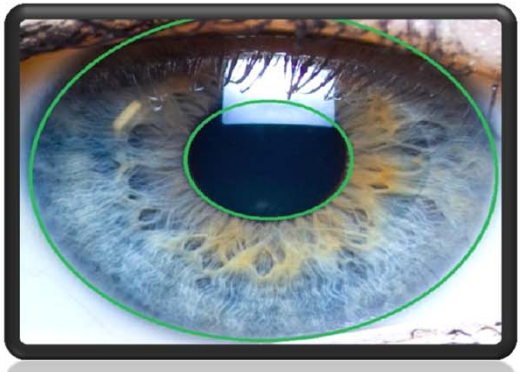
BIOMETRIC IDENTIFICATION

Motivation: secure identification & authentication

- Prevent ID theft
- Identification of criminals/terrorists
- Authentication of authorized users

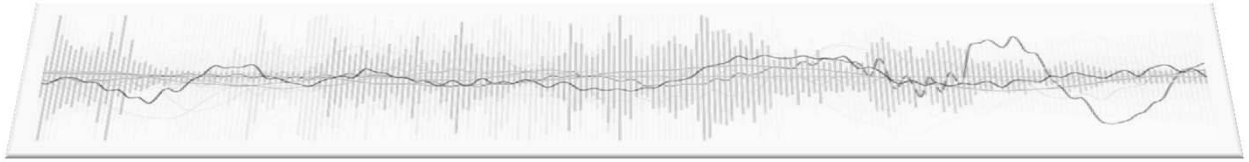
Primary Advantages

- Nothing to remember and/or carry
- Resistant to forgery



Many modalities

- Iris
- Gait
- Fingerprints
- Voice
- DNA
- Face



FACIAL BIOMETRICS

Advantages

- Noninvasive
- Ubiquitous security camera footage
- Relatively large target size
- Conveys emotional state

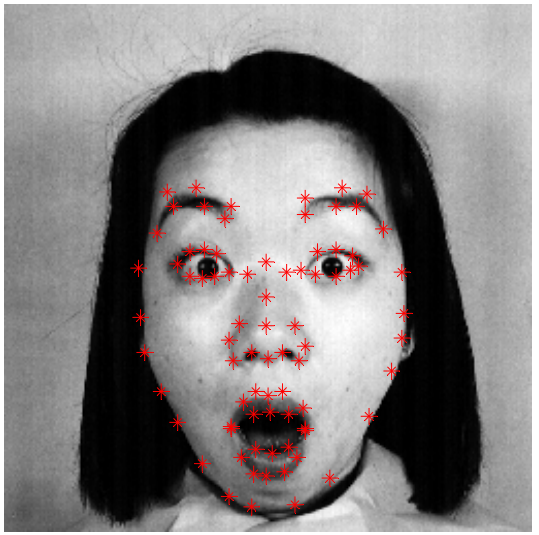
Disadvantages

- Relatively easy to alter
- Can be hidden
- Sensitive to illumination and pose
- Less discriminative than iris

FACIAL LANDMARKING

Key component in many face-based biometric identification systems

- Normalize raw faces for scale/face shape
- Facial expression analysis
- Extract consistent ROIs (eyebrow, nose, mouth, perioccular, etc.)
- Extract geometry
- Generate 3D face models
- Pose estimation



AUTOMATIC LANDMARKING

Goal: locate landmark points on an object

Requirements:

- Automatic
- Robust
- Consistent
- Dense

AUTOMATIC LANDMARKING METHODS

Correlation matching

- Register a labeled representative image

Deformable shape models

- Handcrafted, parameterized models
- Fourier models
- Energy minimizing active contour models
- Finite element models
- Statistical models

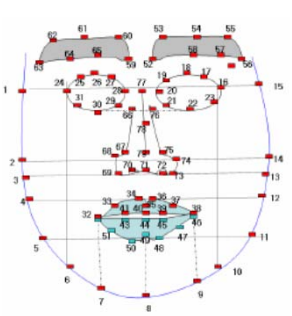
ACTIVE SHAPE MODELS (ASMS)

Deformable statistical model of shape

For each landmark learn:

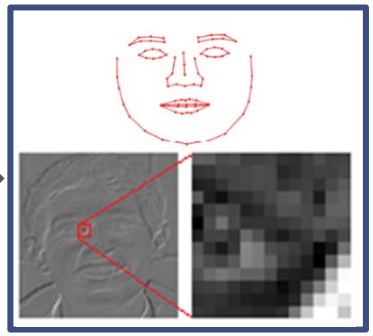
- Point Distribution Model (PDM) → “shape”
- Local Neighborhood Model (LNM) → “profile”

- Move each landmark to location where profile model fits local neighborhood best
- But, constrained to allow only valid shapes



Hand Labeled Landmarks

Training



Shape & Profile Models

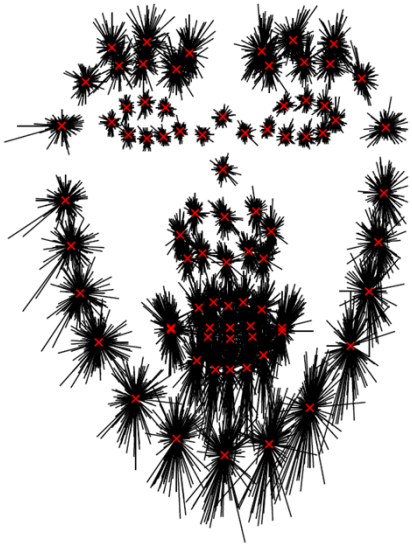
Testing

Automatic Landmarking

ASM TRAINING: ALIGNING THE TRAINING SET

Seek minimize weighted sum of squares between equivalent points on different shapes

- Allow scale, rotation, and translation
- Weights emphasize more “stable” points
- Generalized Procrustes Analysis



<u>To align set of N training shapes:</u>

0. Transform each shape to align with the first shape in the set
1. Calculate mean shape from aligned shapes
2. Normalize the mean shape
3. Realign every shape with the current mean shape
4. Repeat 1-3 until process converges

ASM TRAINING: CAPTURE SHAPE MODEL

Seeks to capture the variation of the coordinate "clouds" surrounding each point

- Principal Component Analysis (PCA)

$$\mathbf{x} = (x_0, ...x_n, y_0...y_n)^{\mathrm{T}}$$

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}$$

- Select those first t eigenvectors which account for a sufficiently large amount of the variance (>97%)

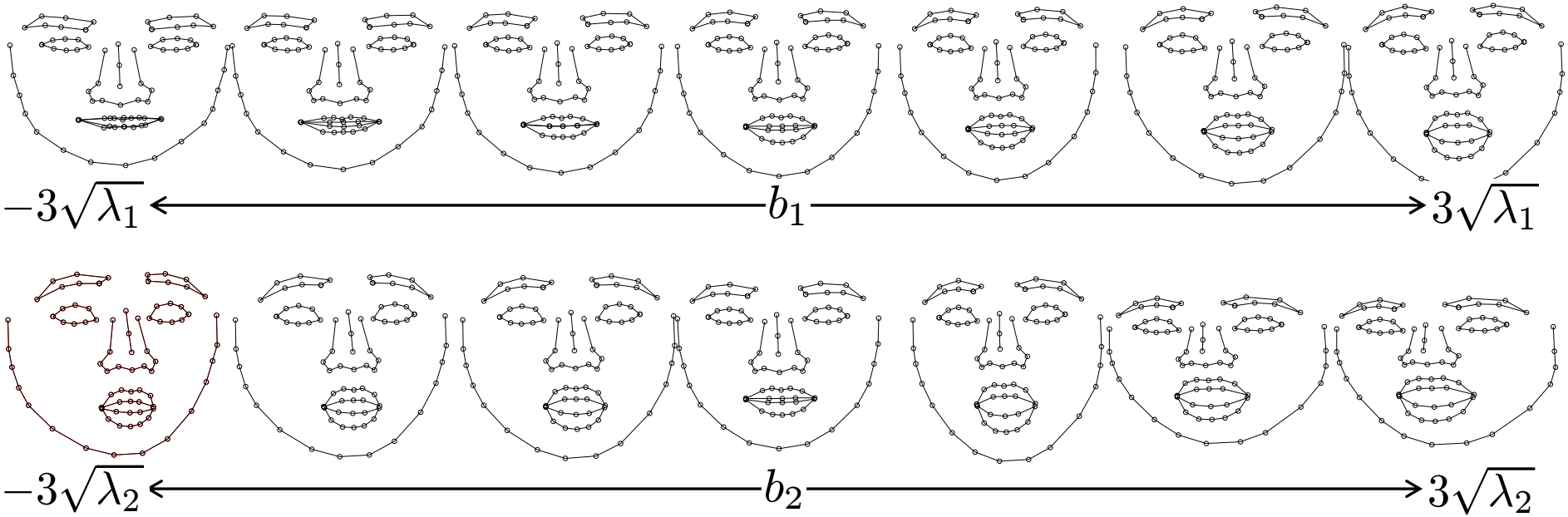
$$\mathbf{S}\mathbf{p}_k = \lambda_k \mathbf{p}_k$$

$$\mathbf{P}_\mathbf{s} = (\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_t)$$

ASM TRAINING: LEARN SHAPE MODEL

Any shape in the training set can be approximated using the mean shape and a linear combination of the eigenvectors

$$\tilde{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{P_s b} \qquad \mathbf{b} = (b_1, b_2 ... b_t)^{\mathrm{T}}$$



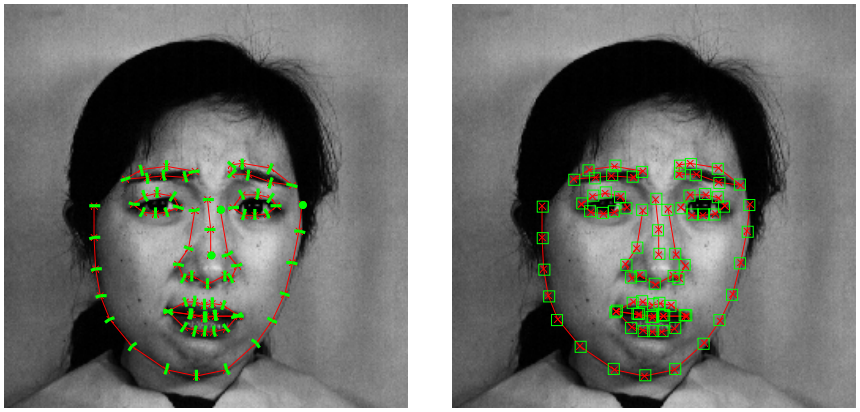
ASM TRAINING: LEARN PROFILE MODEL

For each model point *i* in each training image *j*:

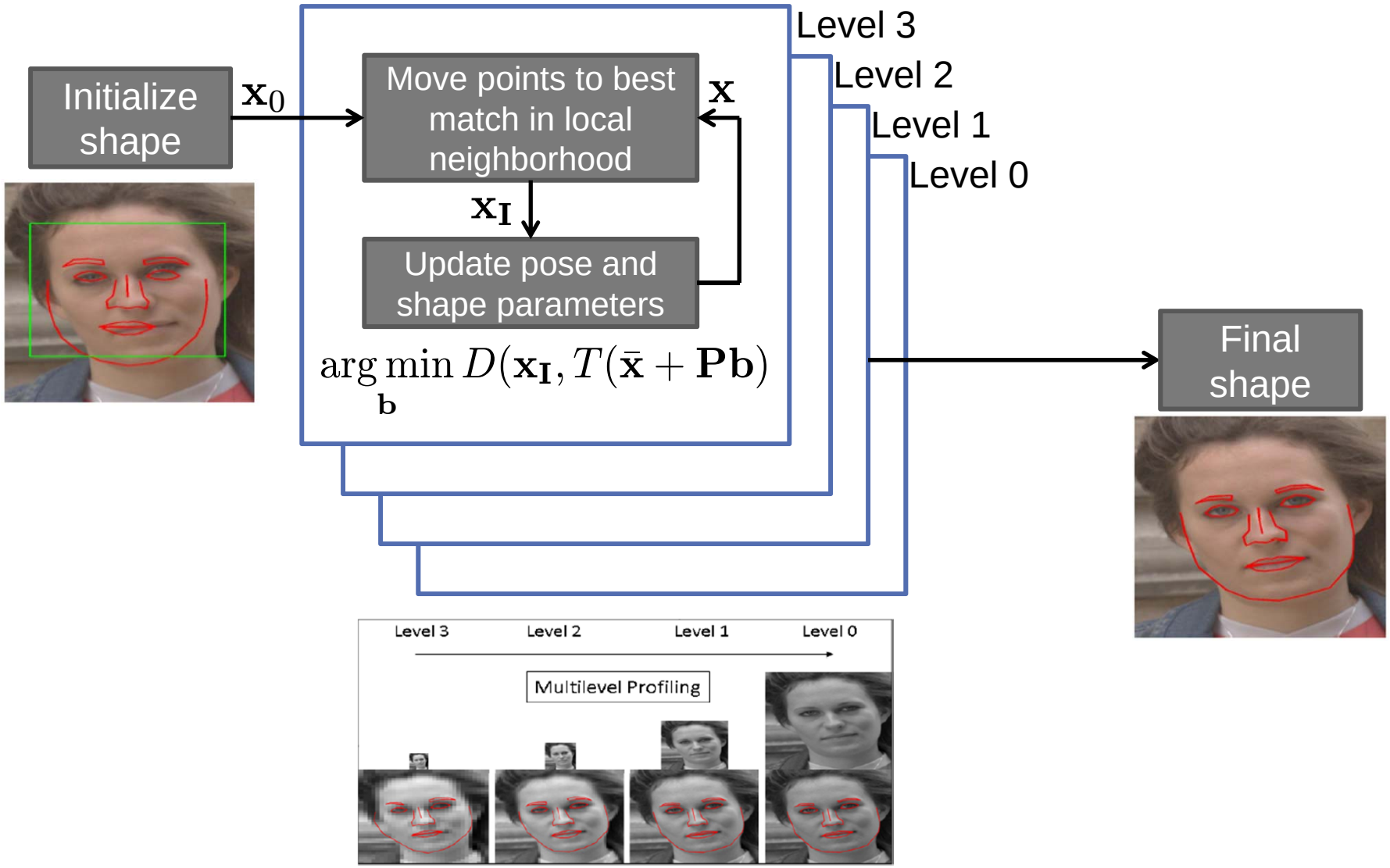
- Extract 1D texture profile $\bar{\mathbf{g}}_j^i$ of length n_p pixels centered at point and oriented “wisker”
- Calculate mean profile $\bar{\mathbf{g}}^i$ and covariance matrix $\mathbf{S}_{\mathbf{g}}{}^i$
- Alternatively 2D profiles can be extracted and vectorized

Performed at each level of Gaussian pyramid

- Multiscale approach



ASM TESTING: OVERVIEW



ASM TESTING: PROFILE SEARCH

Given initial estimate of model point position need set of adjustments which will move each point to a better position

- Move along normal to toward strongest edge
- Best match along normal of candidate profiles

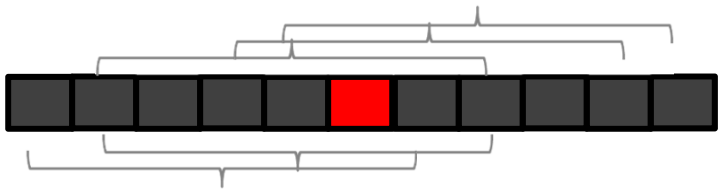
$$D_1(\mathbf{g}) = (\mathbf{g} - \bar{\mathbf{g}})^{\mathrm{T}} \mathbf{S_g}^{-1} (\mathbf{g} - \bar{\mathbf{g}})$$

$k = 3$

$n_p = 2k + 1 = 7$

$m = 2$

$n_l = 2m + 1 = 5$



ASM TESTING: CONSTRAIN SHAPE

$$\mathbf{x_L} = \bar{\mathbf{x}} + \mathbf{P_s b}$$

$$\arg\min_{\mathbf{b}} D(\mathbf{x_I}, T(\bar{\mathbf{x}} + \mathbf{P_s b})$$

$$b_i \in \left[-b_{max}\sqrt{\lambda_i}, b_{max}\sqrt{\lambda_i}\right]$$

The shape must now be constrained to be a legal shape

- Solve for pose and shape parameters which best approximate intermediate shape
- Iterative solution (Cootes et al)

Final constraint on shape parameters

- Ensures shape is plausible
- Limits derived from variation in shape parameters used to generate training set

MASM: MODIFIED ACTIVE SHAPE MODELS

ASM variant

- Recent work demonstrated decreased fitting error specifically in face landmarking
- More tolerant to variation in illumination and unseen face images
- Able to incorporate greater numbers of training samples

Major improvements

- Subspace based model of texture patches surrounding points
- New metric for choosing best candidate location in profile search phase
- Refitting of poorly fitted points

MASM: SUBSPACE TEXTURE MODEL

Utilize 2D gradient patches around each point *i*

- Recent work demonstrated decreased fitting error specifically in face landmarking
- Calculate mean profile $\bar{\mathbf{g}}^i$ and covariance matrix $\mathbf{S_g}^i$
- Additionally, select t_i eigenvectors which account for a sufficiently large amount of the variance (ie. >97%)
- Store basis in $\mathbf{P_g}^i$

MASM: DISTANCE METRIC

Utilize reconstruction distance

- First project profile candidate onto eigenvectors
- Reconstruct profile
- Calculate Mahalanobis distance from reconstructed profile to original profile

Edge information

- Only for facial boundary points

$$\mathbf{g}' = \mathbf{P_g}^{\mathrm{T}}(\mathbf{g} - \bar{\mathbf{g}})$$

$$\mathbf{g_r} = \bar{\mathbf{g}} + \mathbf{P_g}\mathbf{g}'$$

$$D_2(g) = (\mathbf{g_r} - \mathbf{g})^{\mathrm{T}}\mathbf{S_g}^{-1}(\mathbf{g_r} - \mathbf{g})$$

$$D_3(g) = (c - I)(\mathbf{g_r} - \mathbf{g})^{\mathrm{T}}\mathbf{S_g}^{-1}(\mathbf{g_r} - \mathbf{g})$$

MASM: REFITTING OF POORLY FITTED POINTS

After convergence of ASM fitting on the final (highest resolution) level of the pyramid

- Profile distances compared to an empirically determined threshold
- Those points with a error > threshold are subject to additional iterations
- Boosts accuracy and leads to overall improvement in fitting

MASM: COMPUTATIONALLY INTENSIVE

The improved robustness of MASM comes with greater computational cost

- Execution time is on the order of seconds to converge
  - Iterative process that runs until convergence, so runtimes are data dependent

Why? Profile search dominates runtime

- Computing reconstructed profile $2(t)(n_p)$ MAD operations
- Mahalanobis distance $n_p^2 + n_p$ MAD operations
- Fairly large 2D profiles (13 x 13) means our matrices are large. ie profile covariance matrix (169 x 169)
  - All matrix operations must be performed for each point over all possible offsets (5 x 5) → 1975 times per iteration
- Individual profile models for each point

MASM is useful, so what do we do?

GRAPHICS PROCESSING UNITS

One approach to accelerating compute-intensive tasks is to offload some or all of the computations to dedicated hardware

- FPGAs
- ASICs
- GPUs

Market demand for realistic 3D games has caused GPU to evolve

- No longer a fixed rendering pipline
- Fully programmable
- Massively parallel threading model
- Many core

GPUS: FLOPS COMPARISON

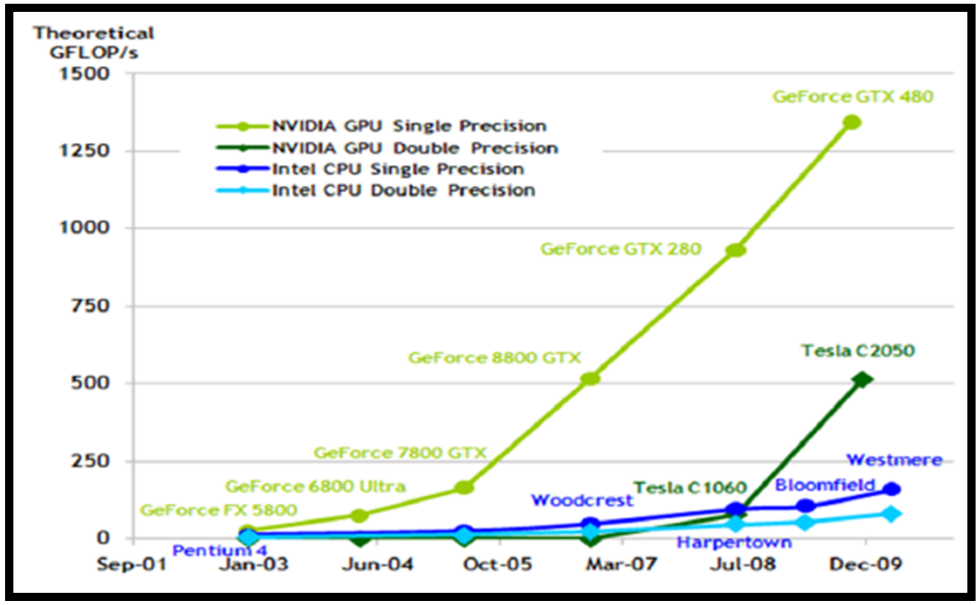
Growth rate of computational capability

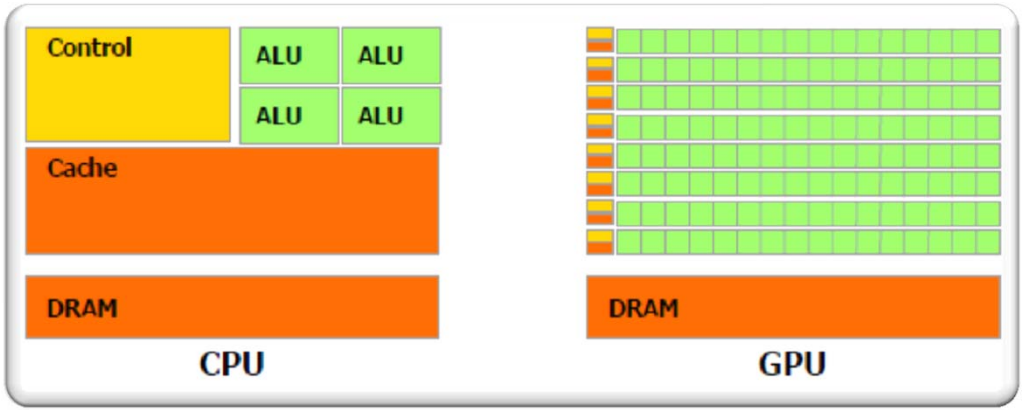
- GPU average yearly rate of 1.7 to 2.3
- CPU average yearly rate of 1.4

Why?

- Moore's law has been for the benefit of both, but…
- GPU devotes more die space to transistors

Currently theoretical peak FLOPs ratio ~25:1





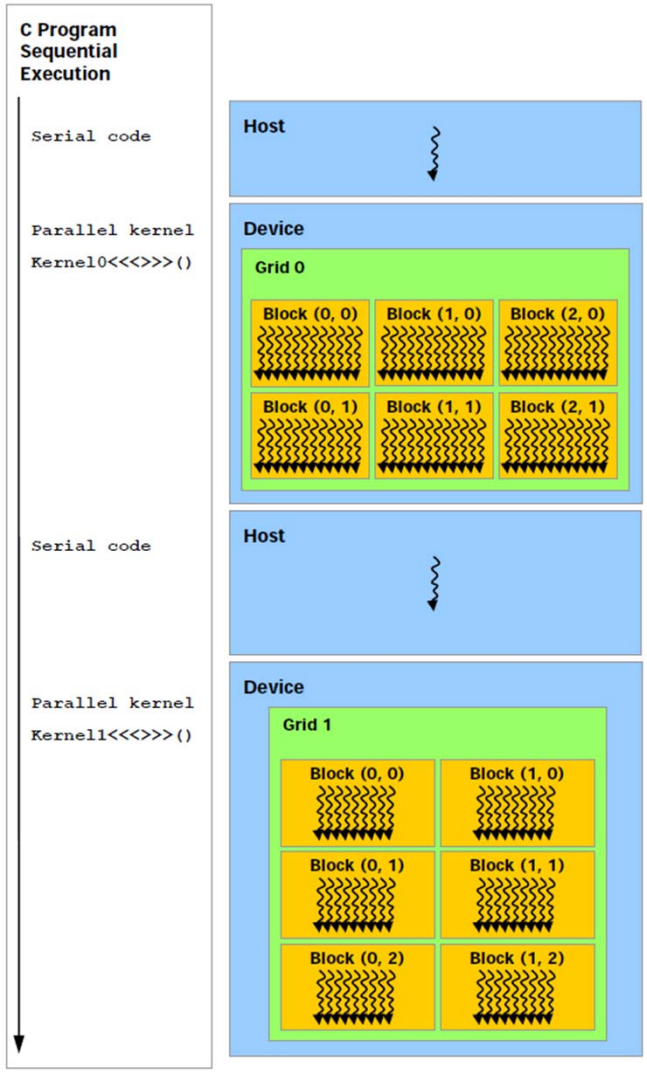
COMPUTE UNIFIED DEVICE ARCHITECTURE

Prior to introduction of CUDA in 2006 (GPGPU) was hard work:

- Express problems in terms of graphics primitives

What is CUDA?

- Nvidia's brand name for their hardware and software architecture
- Execute programs written with C, C++, Fortran, OpenCL, DirectCompute and other high level languages
- Heterogeneous programming



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

CUDA: KERNELS

Kernels: Parallel Extension to C/C++ functions

- Executed N times in parallel by N threads
- Specify the *execution configuration* using the <<< >>> syntax

```
//kernel definition
__global__ void VecAdd(float * A, float * B, float * C)
{
        int i = threadIdx.x;
        C[i] = A[i] + B[i];
}

int main()
{
        ...
        //kernel invocation with N threads
        VecAdd<<<1, N>>>(A, B, C);
}
```

CUDA: HARDWARE ABSTRACTION

3 key abstractions:

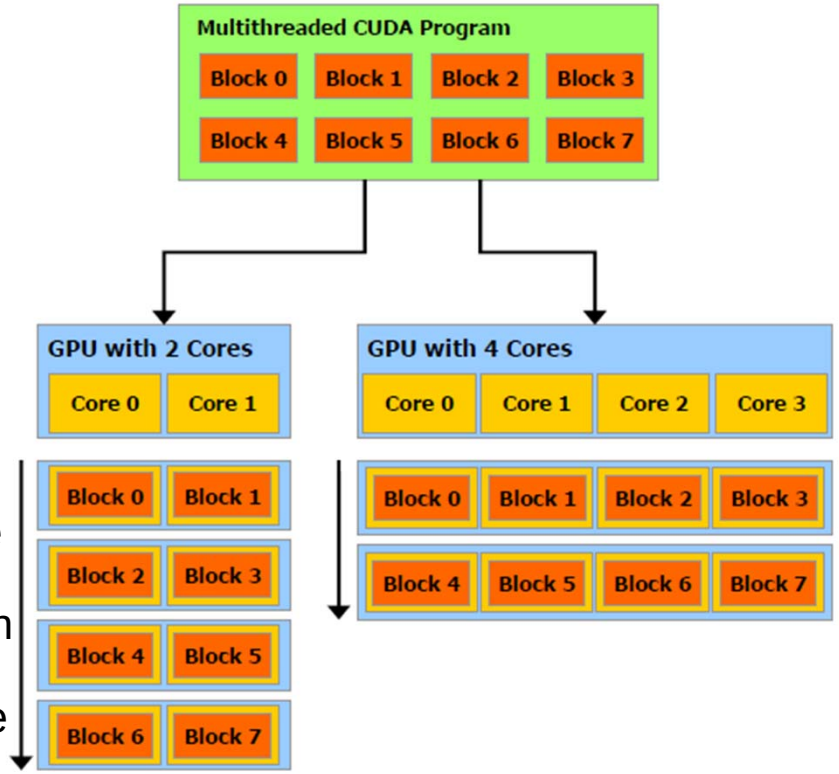
- Hierarchy of thread groups
- Shared memories
- Barrier synchronization

Partition problem

- Coarsely into sub-problems to be solved by blocks independently
- Within block, threads operate cooperatively

Automatic scalability

- Blocks can be scheduled on any available processor core
- threads of a block execute concurrently on a single SM
- multiple thread blocks can execute on one SM.
- As thread blocks terminate, new blocks are launched on the vacated multiprocessors



CUDA: THREAD HIERARCHY

Threads

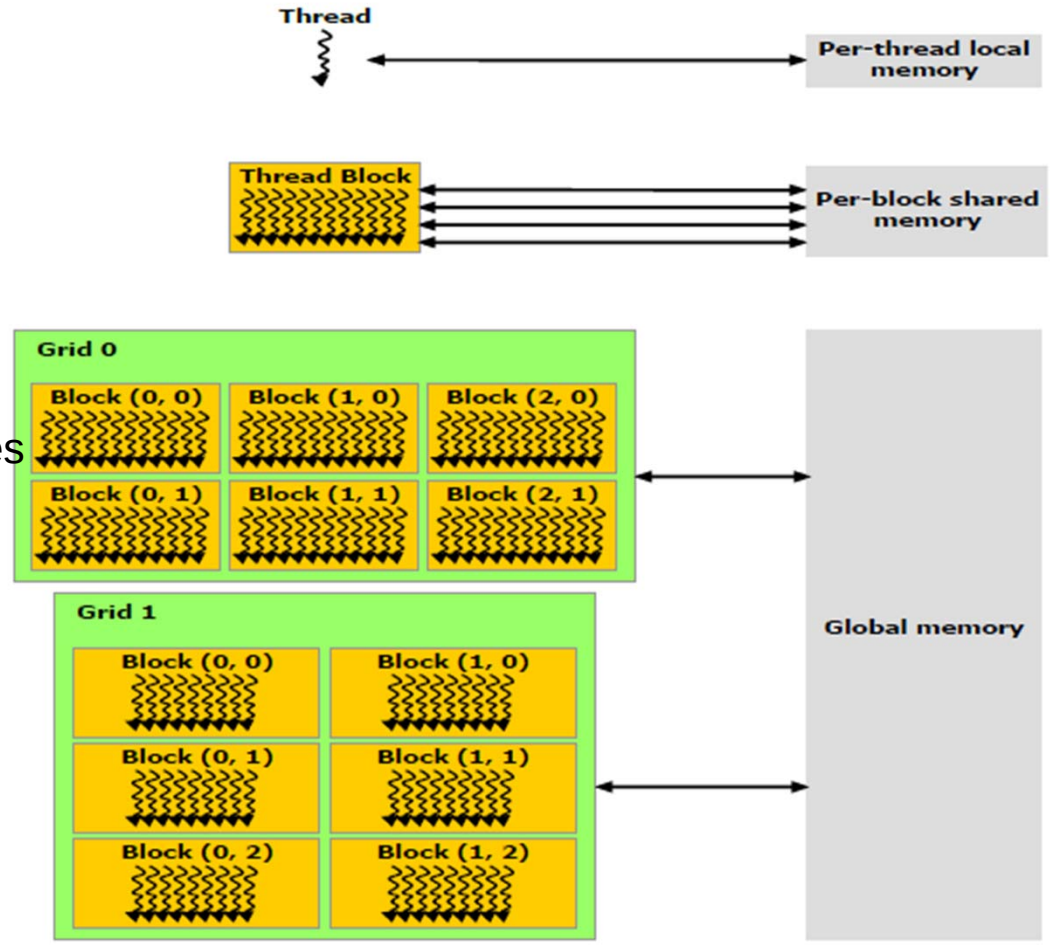
- unique thread ID and program counter
- private registers and local memory

Thread Block

- set of concurrently executing threads
- unique block ID
- synchronize with barrier primitives
- communicate with shared memory

Grid

- set of all (possibly concurrently executing) blocks in a kernel call
- communicate through global device memory



CUDA: MEMORY HIERARCHY

Registers

- Reside on chip
- Private to thread
- Lowest latency
- Very limited resource <48 per thread available

Local memory

- Private to thread
- Register spillover
- Despite name, reside in global memory

Shared memory

- Reside on chip
- 16KB/48KB per SM
- Shared by block
- Fast (2 clock cycles)

Global memory

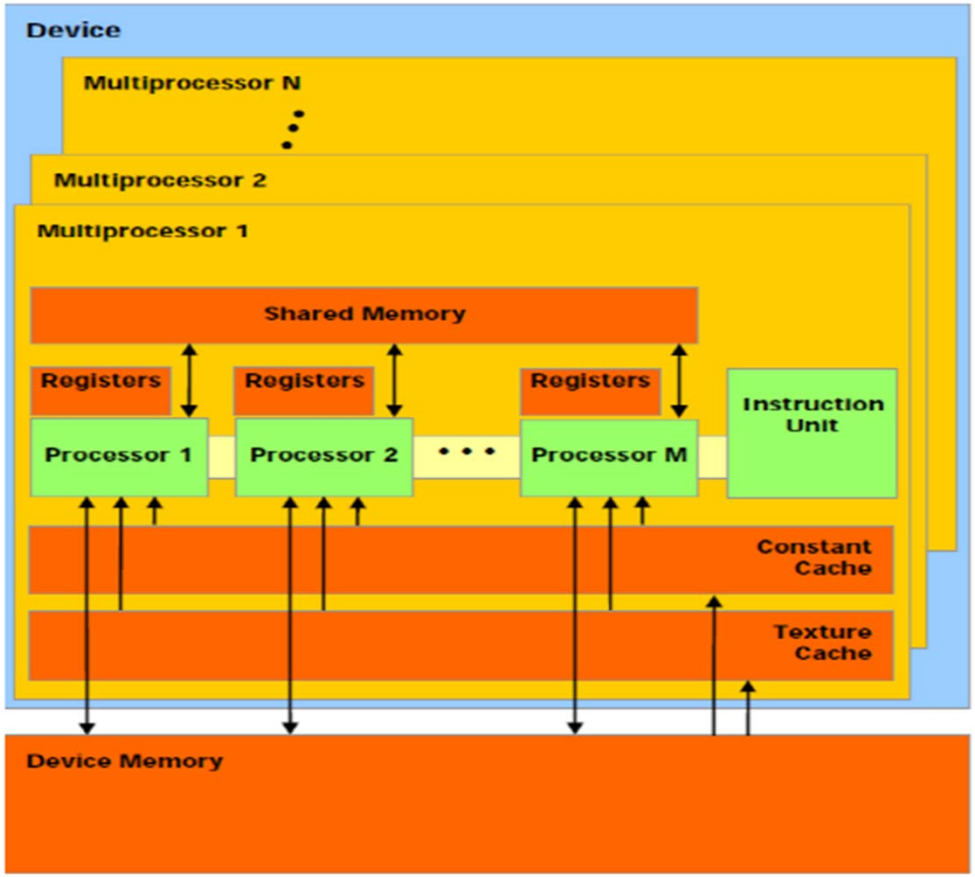
- Persistent between kernel launches
- Communication between blocks
- Communication between device and host
- Slow (hundreds of clock cycles)
- Un-cached in GPUS < capability 2.x

Constant cache

- Read-only from device

Texture cache

- Optimized for 2D spatial layouts
- Special addressing modes, interpolation, normalization



EXPLOITING PARALLELISM IN MASM

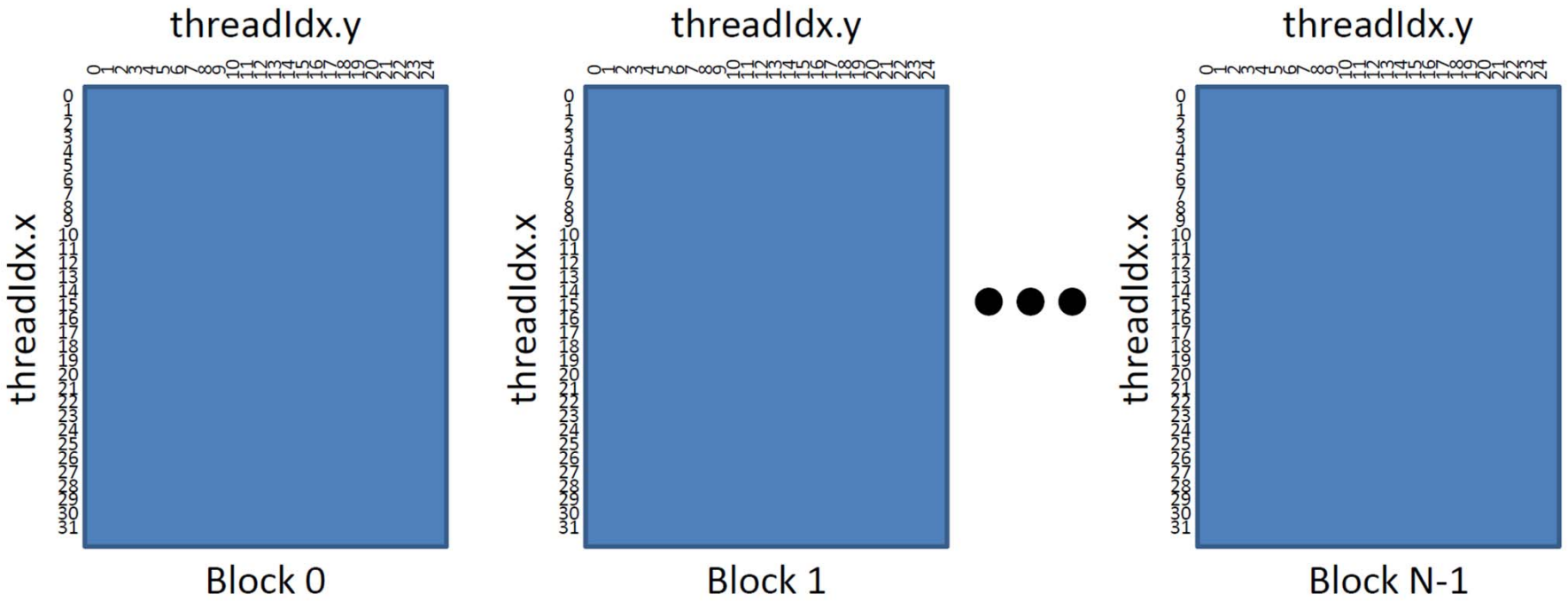
The runtime of MASM is dominated by the profile search step

- Even after heavy optimization and parallelization in our final parallel GPU implementation it accounts for 73% of the runtime

Where is the parallelism ? How should we partition?

- 79 landmark points?
  - Partition landmark points between available cores (ala CPU implementation)?
  - Can do this…but insufficient to fully saturate device designed to have 1000's of threads
- 25 neighborhoods around each point?
  - 79 blocks of 32 threads (next power of 2)
  - Able to evaluate all of the 25 offsets and pick best location all in shared memory!
  - All threads in a block able to share the same profile models!
  - But, 2528 threads still insufficient to hide memory latency AND due to high shared memory usage, number of threads per block should be >> 32
- The matrix operations?
  - Instead of iterating through all of the matrix multiplications sequentially, can work in parallel
  - Assign 32 worker threads for each 25 offsets (800 threads per block)
  - 63200 total threads better able to saturate device and hide memory latency

cuMASM: PROFILE SEARCH EXECUTION CONFIGURATION



OffsetIdx = threadIdx.y

PointIdx = blockIdx.x

WorkerThreadIdx = threadIdx.x

We intialize our routine by loading all overall parameters and model parameters to device global memory once prior to beginning to process any images

cuMASM: PROFILE SEARCH KERNEL

Major non-trivial components:

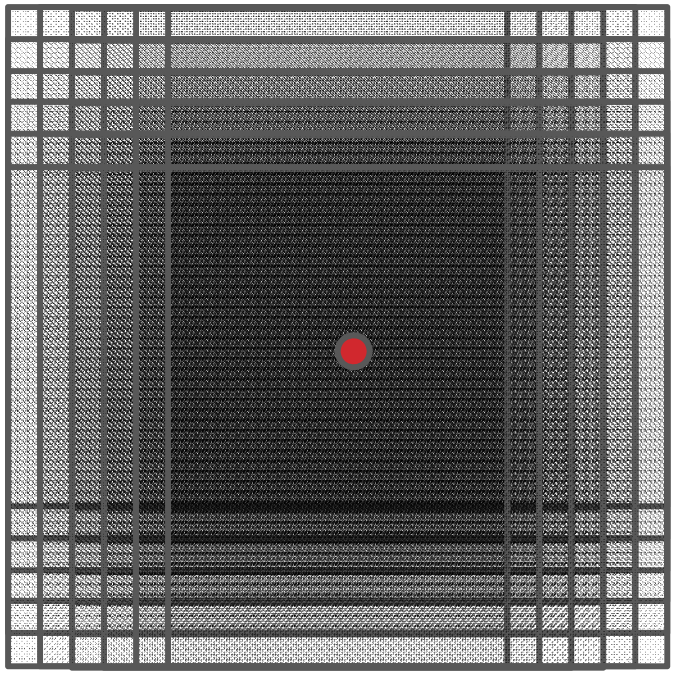
- Patch fetching
- Intra-block reduction
- Matrix-vector multiplication
- Mahalanobis distance

Implementation makes extensive use of templates/preprocessor macros to reduce runtime decisions as much as possible

PROFILE SEARCH KERNEL COMPONENTS: PATCH FETCHING

Fetch surrounding patch of gradient for each offset at every point

- Each of the 32 worker threads for a given offset cooperate to loads their 2D patch of gradient and place it into a shared memory vector
- Significant overlap between the 25 patches (13 x 13)
- Use 2D texture memory to help reduce global memory access cost
  - texture cache hit rate of 99.69%.
- Optional hardware bilinear interpolation
  - Low precision; actually increased overall runtime

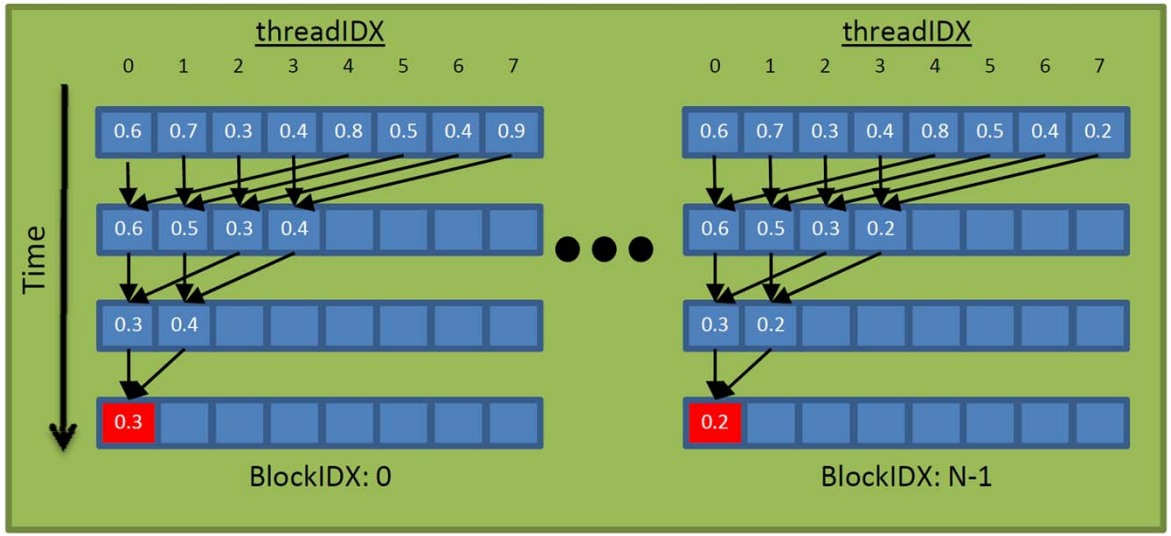


PROFILE SEARCH KERNEL COMPONENTS: INTRA-BLOCK REDUCTION

```
//each thread puts its local distance into shared memory
s_mem [threadID] = dist;
__syncthreads();

//do reduction in shared mem
#pragma unroll
for(unsigned int s=(blockSize>>1); s>0; s>>=1){
  if (threadID < s)
    s_mem[threadID] = dist = min(dist, s_mem[threadID+s]);
  __syncthreads();
}

//All threads get local copy of the min distance for the
//entire block (which is the point we are operating on)
min_dist2 = s_mem[0];
```



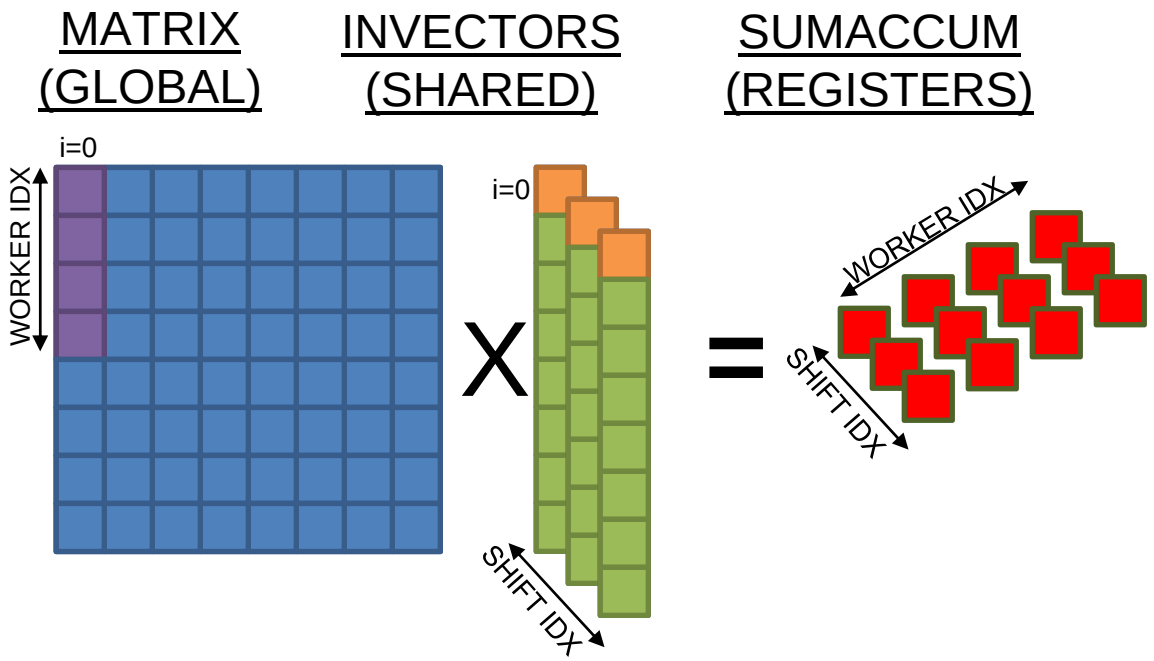
Reduction is a common parallel primitive

- commutative binary operator (ie. sum, product, min, max) is applied to all the elements of a vector to produce a scalar value
- Used in Profile Search:
  - which shift within a thread-block results in the minimum reconstruction distance
  - component of more complex matrix routines

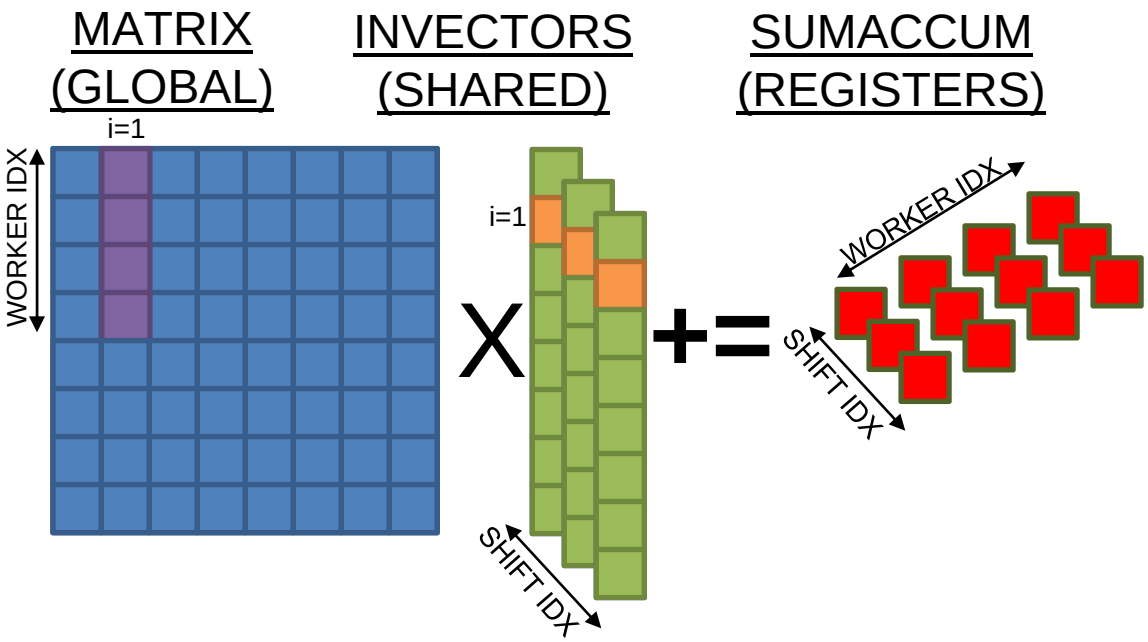
PROFILE SEARCH KERNEL COMPONENTS: MATRIX-VECTOR MULTIPLICATION

GEMV is used to project and reconstruct each profile.

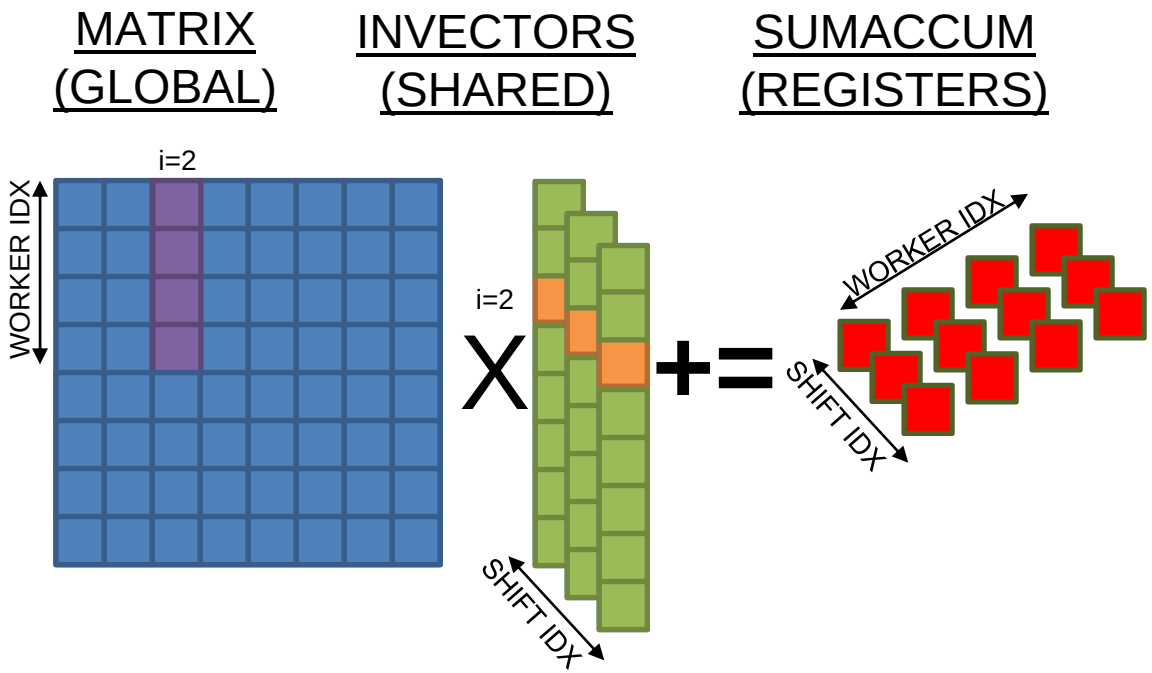
- Additionally, since this is performed for each of the possible shifts, this can be viewed as a general matrix multiplication.



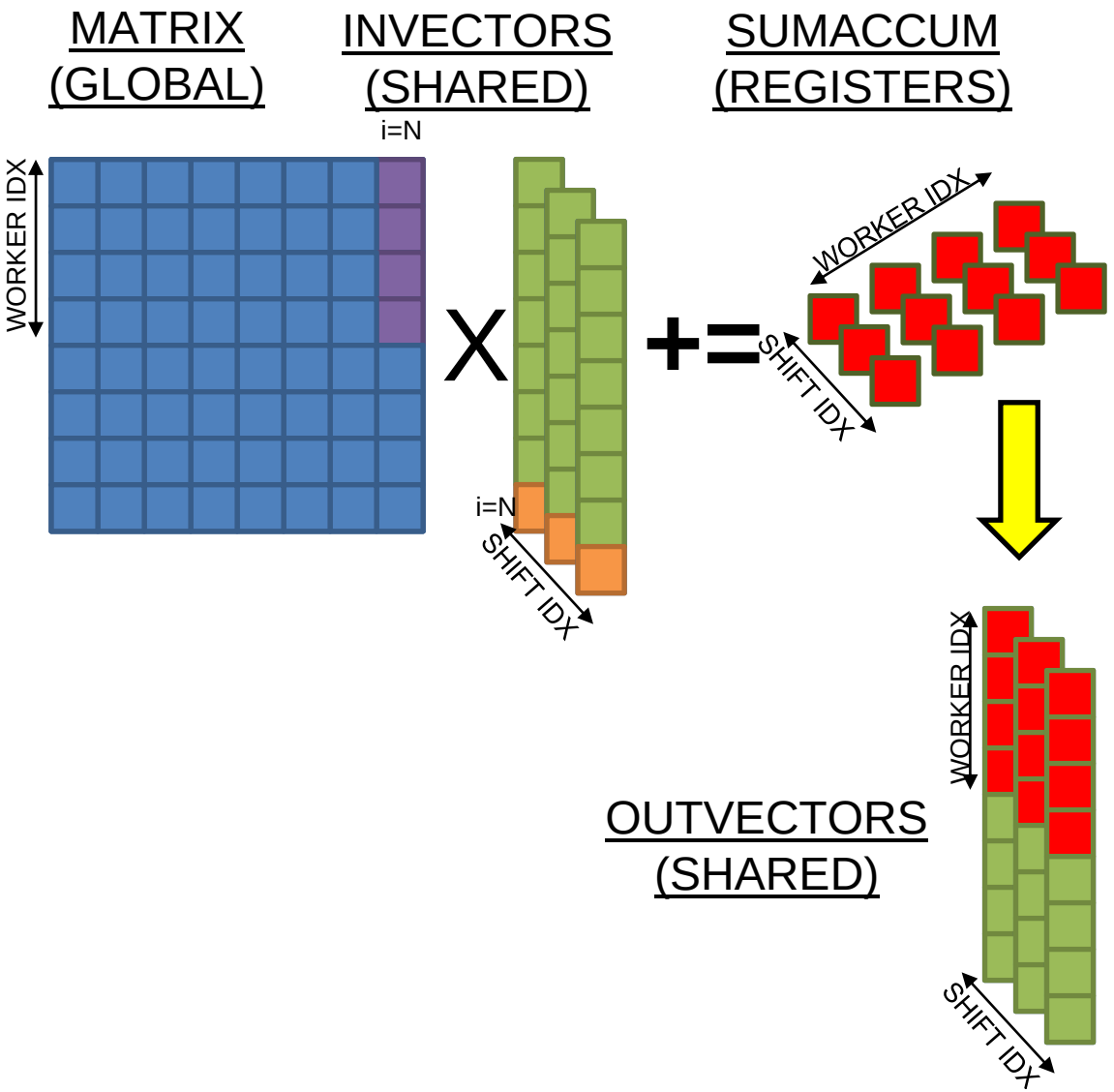
PROFILE SEARCH KERNEL COMPONENTS: MATRIX-VECTOR MULTIPLICATION



PROFILE SEARCH KERNEL COMPONENTS: MATRIX-VECTOR MULTIPLICATION

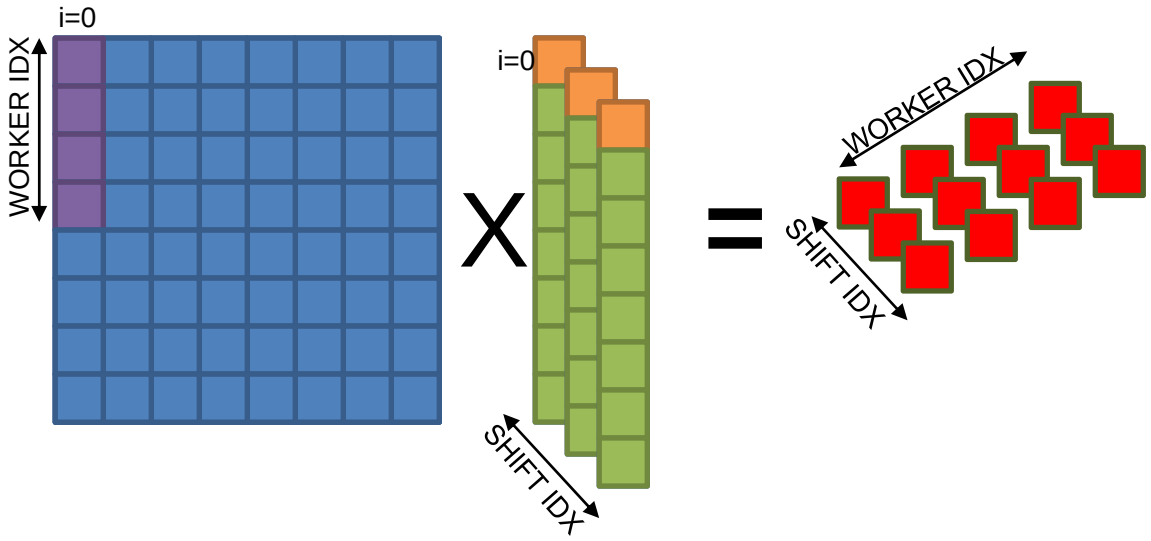


PROFILE SEARCH KERNEL COMPONENTS: MATRIX-VECTOR MULTIPLICATION



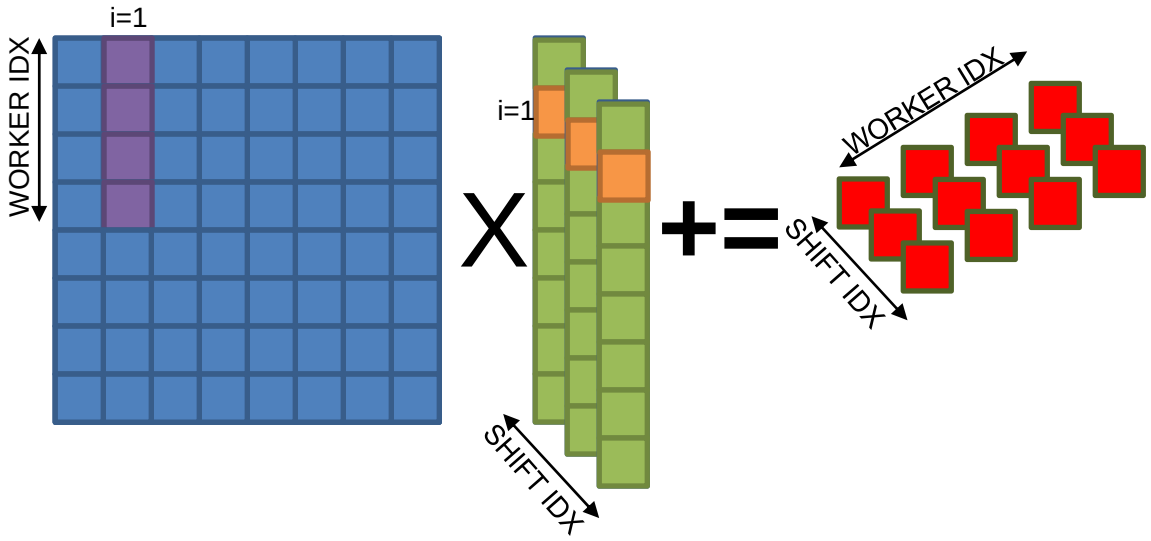
PROFILE SEARCH KERNEL COMPONENTS: MAHANOLOBIS DISTANCE

MATRIX (GLOBAL) INVECTORS (SHARED) SUMACCUM (REGISTERS)

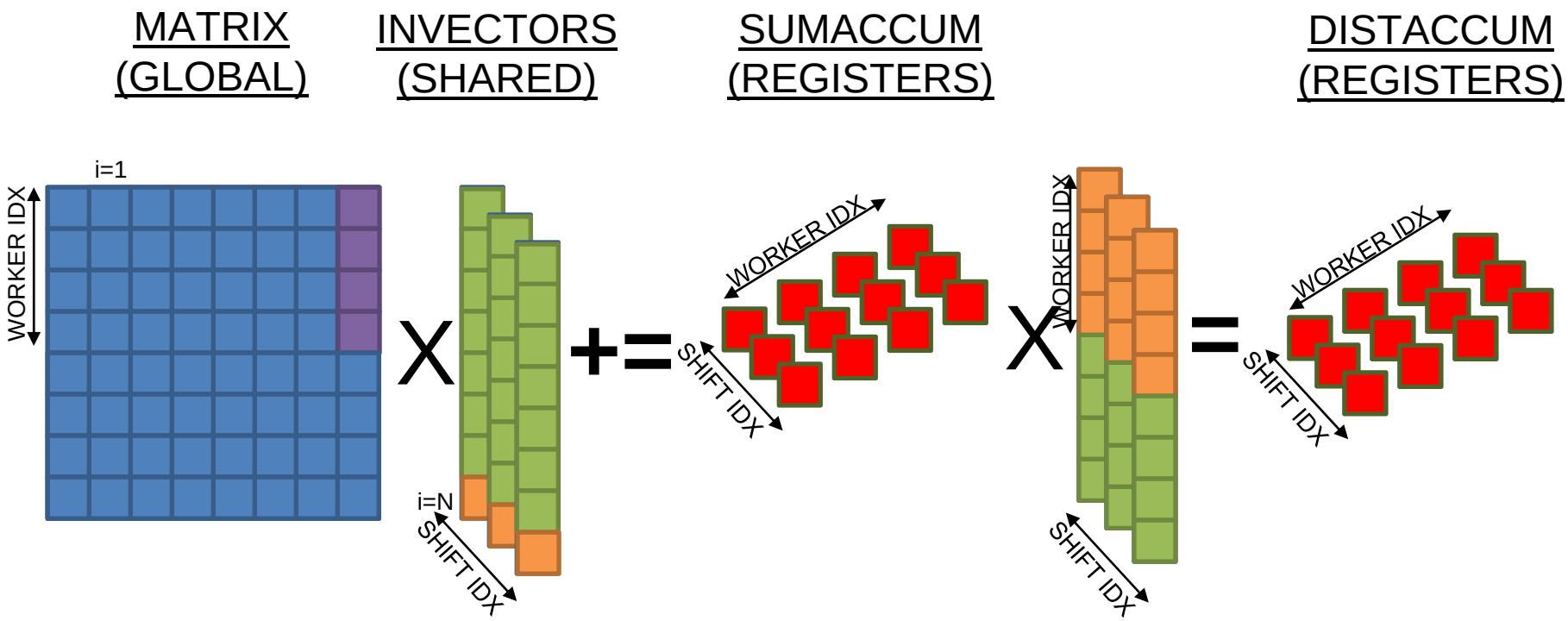


PROFILE SEARCH KERNEL COMPONENTS: MAHANOLOBIS DISTANCE

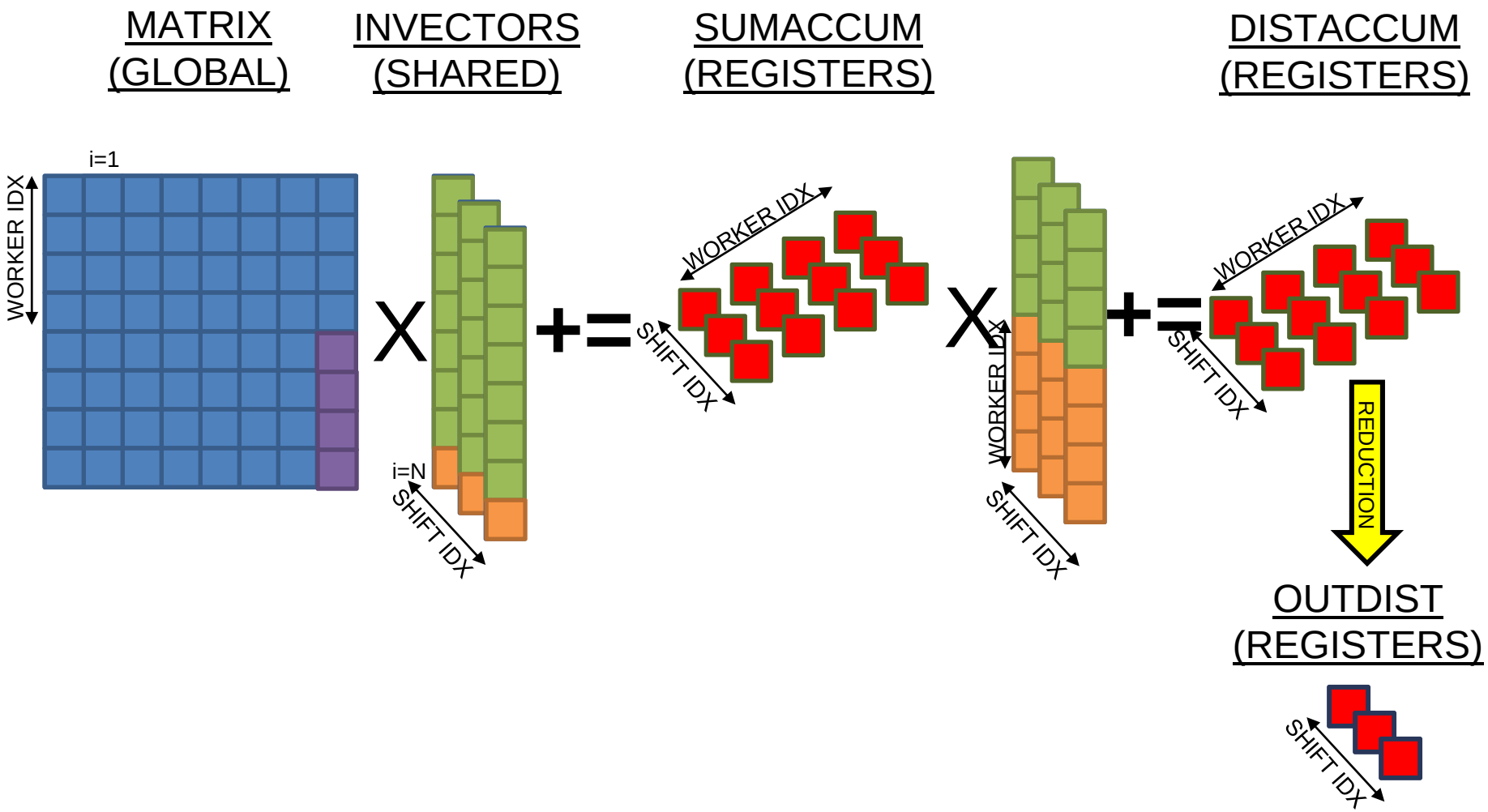
MATRIX (GLOBAL) | INVECTORS (SHARED) | SUMACCUM (REGISTERS)



PROFILE SEARCH KERNEL COMPONENTS: MAHANOLOBIS DISTANCE



PROFILE SEARCH KERNEL COMPONENTS: MAHANOLOBIS DISTANCE



NAÏVE MATRIX OPERATIONS

These matrix operations presented are somewhat naïve

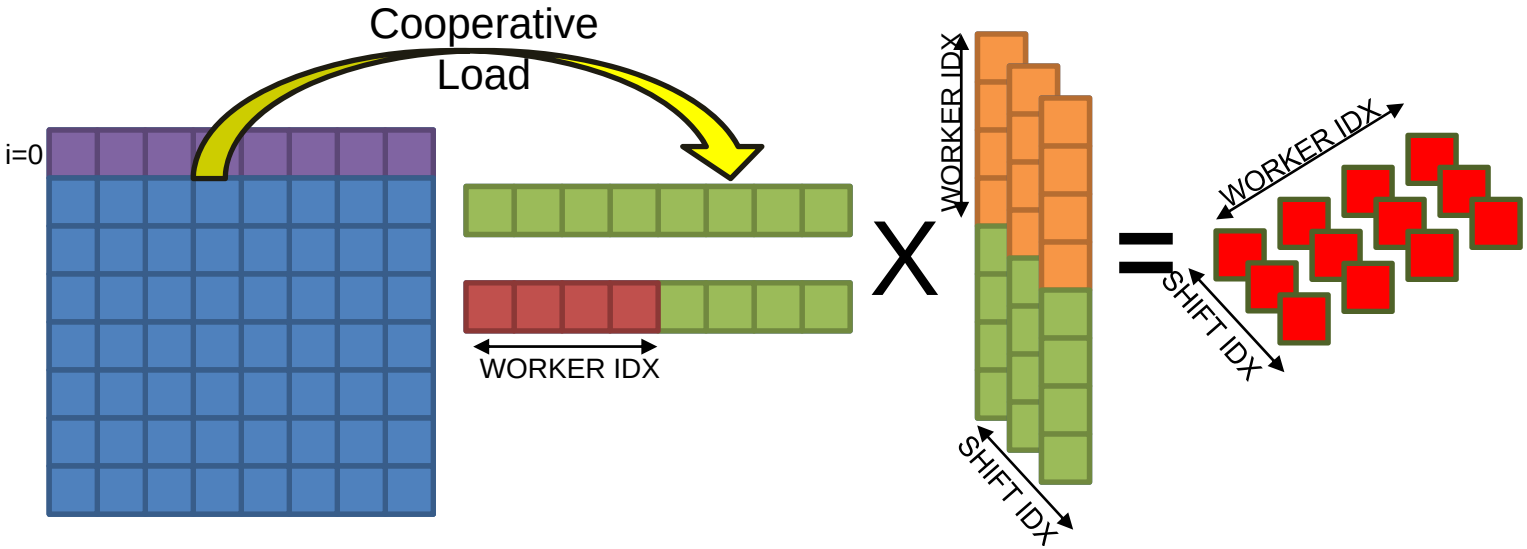
- The constant cache is used to reduce cost of accessing model parameters,
  - Especially the large covariance matrices required for computing Mahalanobis distance locations,
  - 111kB (169x169 matrices of single precision floats) are too large to completely store in shared memory
- Although operations occur in shared memory or register file and input vectors also reside in shared memory
  - cache can offset the cost of accessing the large matrices only to a limited extent
  - High latency in our memory global memory access
- Access pattern is also nonoptimal
  - Iterating down columns in a row-major matrix
  - Warps (groups of 32 threads) are accessing noncontiguous memory

COOPERATIVE MATRIX OPERATIONS

Cooperative preload rows of each matrix from global memory into shared memory

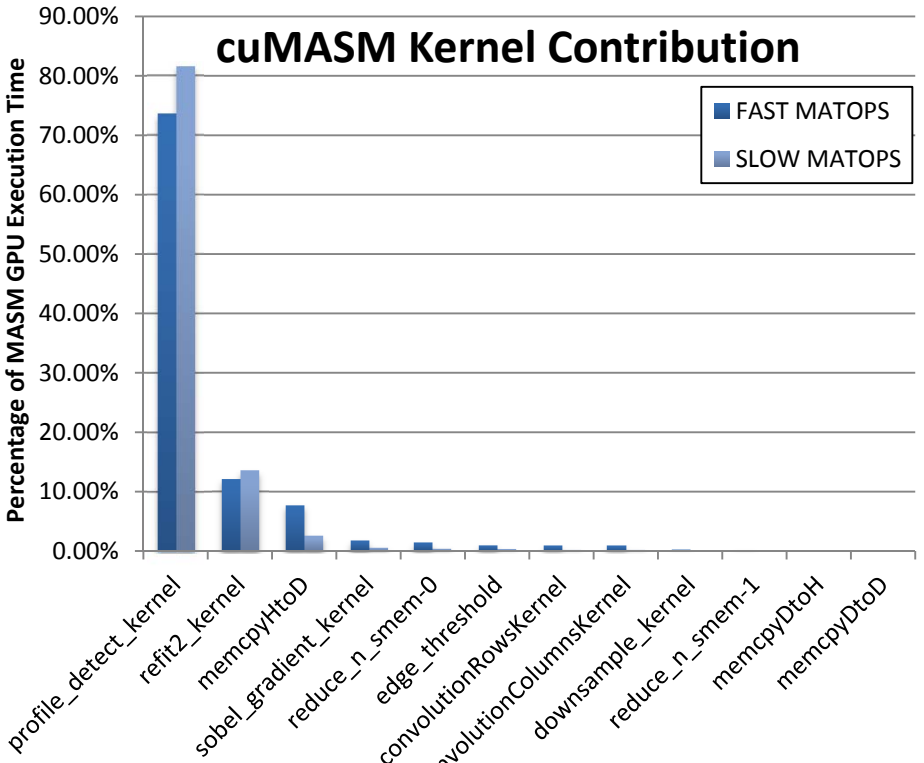
- greatly reduce the total global memory load requests
- reduce our memory bandwidth requirements
- better hide global memory latency

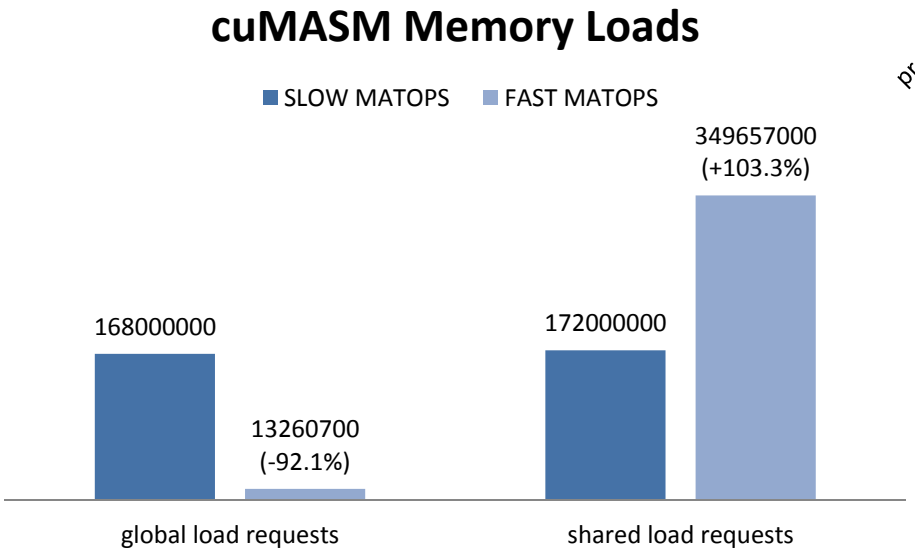
Change memory access pattern: iterate down the rows of matrix, and cooperatively partition the columns



COOPERATIVE MATRIX OPERATIONS

Result of these modifications are immediately measurable using the CUDA profiler





EXPERIMENTAL SETUP

GPU Testing

- Nvidia GTX 470 (compute capability 2.0)
- CUDA runtime v3.2
- 448 cores (14 SMs)

CPU Testing

- OpenMP and OpenCV implementation
- Intel Core i7 920 (Quadcore) @ 3.19GHz

Training

- MATLAB implementation (not performance critical)

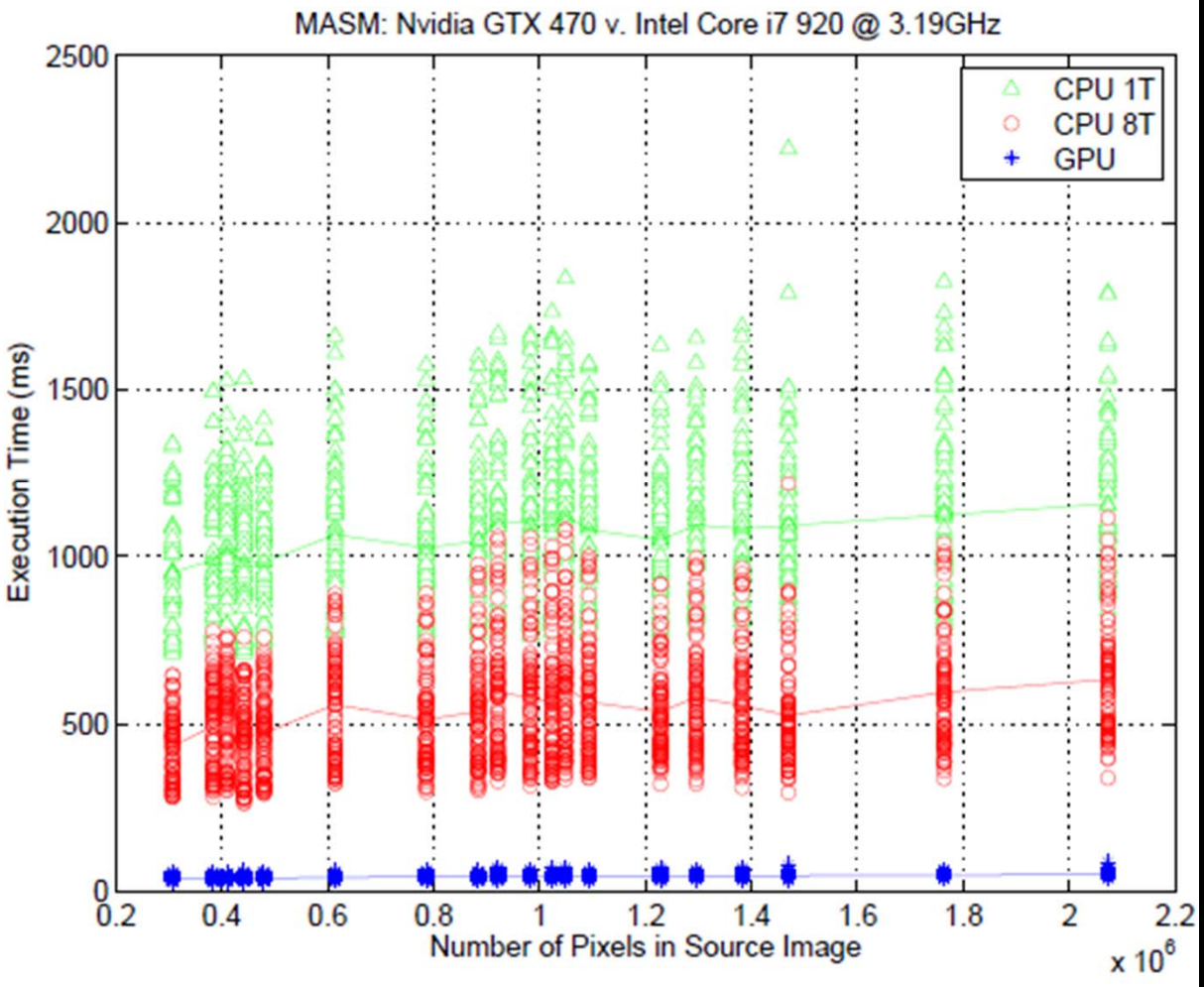
SPEED EVALUATION: GPU/CPU

HD video stream

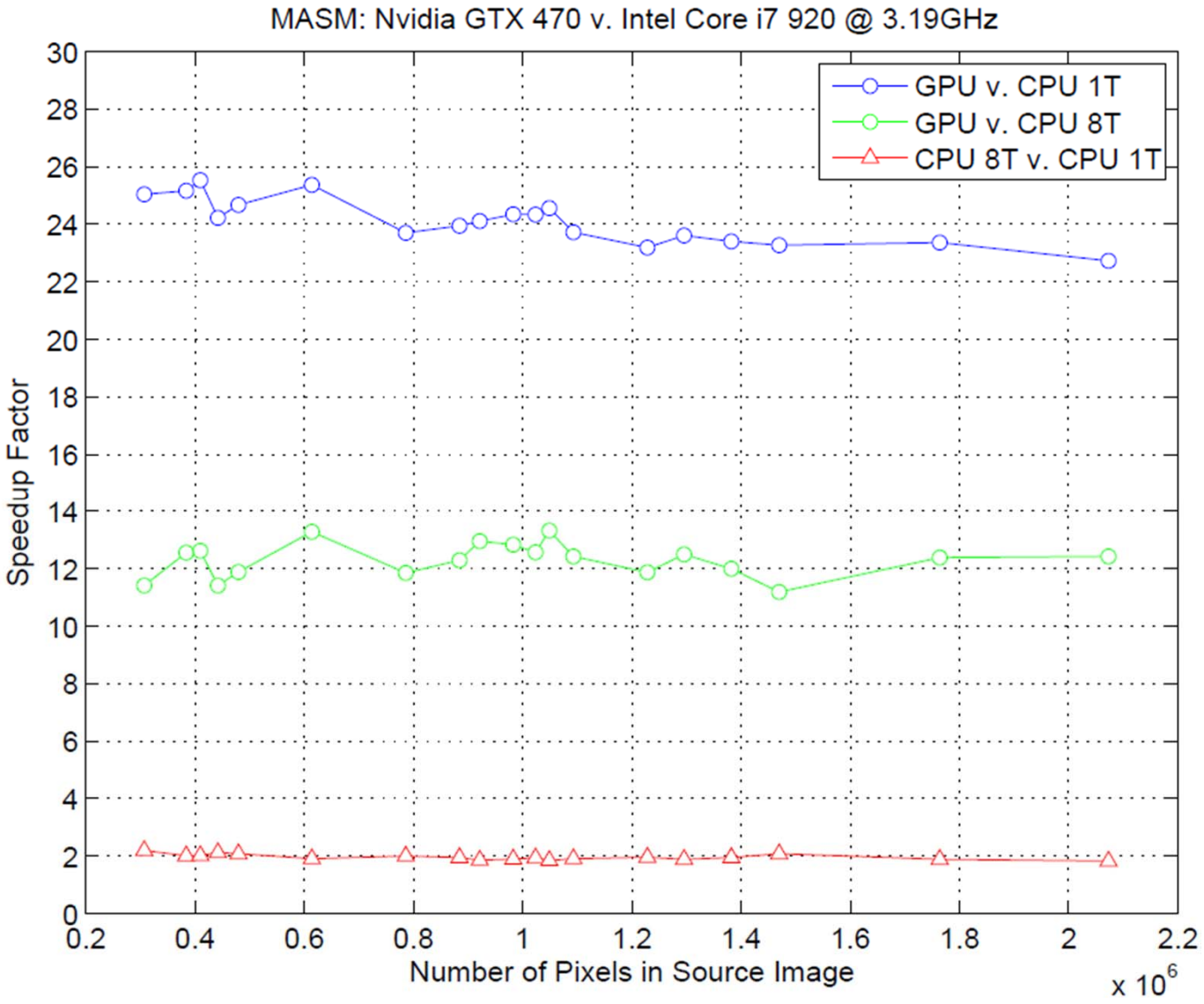
- captured images (of various resolutions)
- subjects from our lab performing work at their desks
- Captures variability needed to evaluate runtimes over range of required iterations

Initialization

- OpenCV Viola-Jones face detection

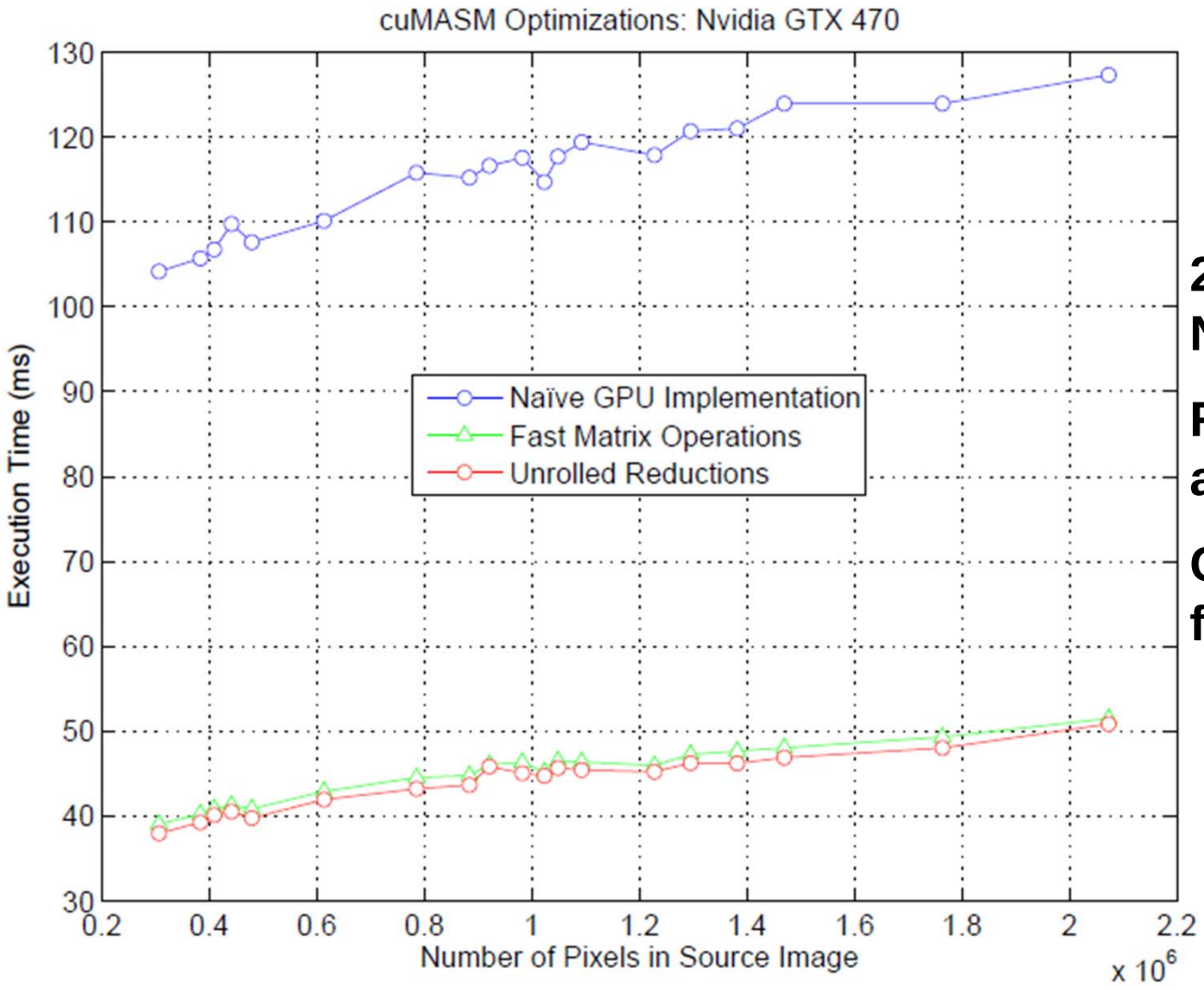


SPEED EVALUATION: GPU/CPU



Our GPU implementation achieves 12X speedup against multithreaded implementation on CPU

SPEED EVALUATION: OPTIMIZATIONS MATTER

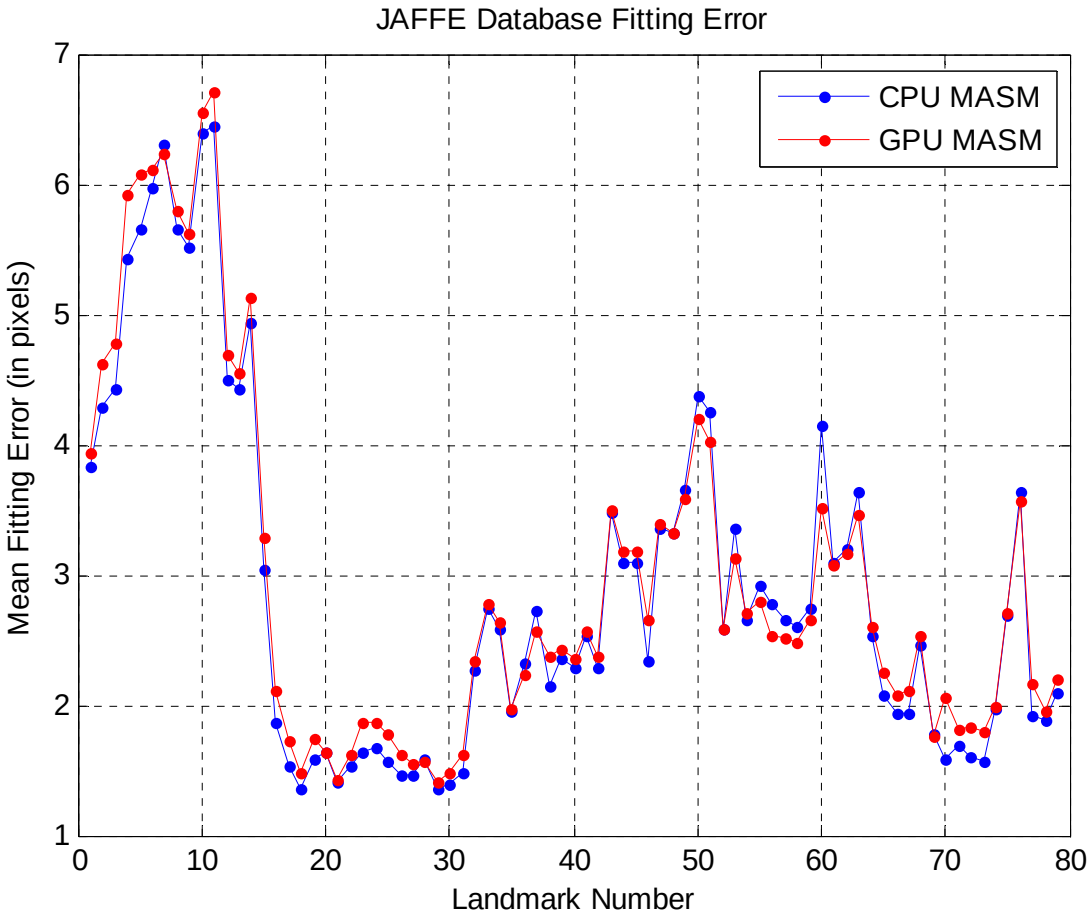


2.75X speedup over Naïve implementation

Reduced run time approximately by 63%

Capable of achieving frame rates ~20FPS

PERFORMANCE EVALUATION



Evaluation of MASM fitting error over JAFFE database

- Unseen test set
- 213 images (256 x 256) of 10 subjects
- Expression database
- Verification that our GPU implementation is performing as well as the serial CPU implementation

$$RMSE_{CPU} = 4.2828px$$

$$RMSE_{GPU} = 4.2773px$$

FUTURE WORK: BIOMETRIC SYSTEM INTEGRATION

MASM is integrated into demonstration software that simply annotates an incoming video stream with landmarks

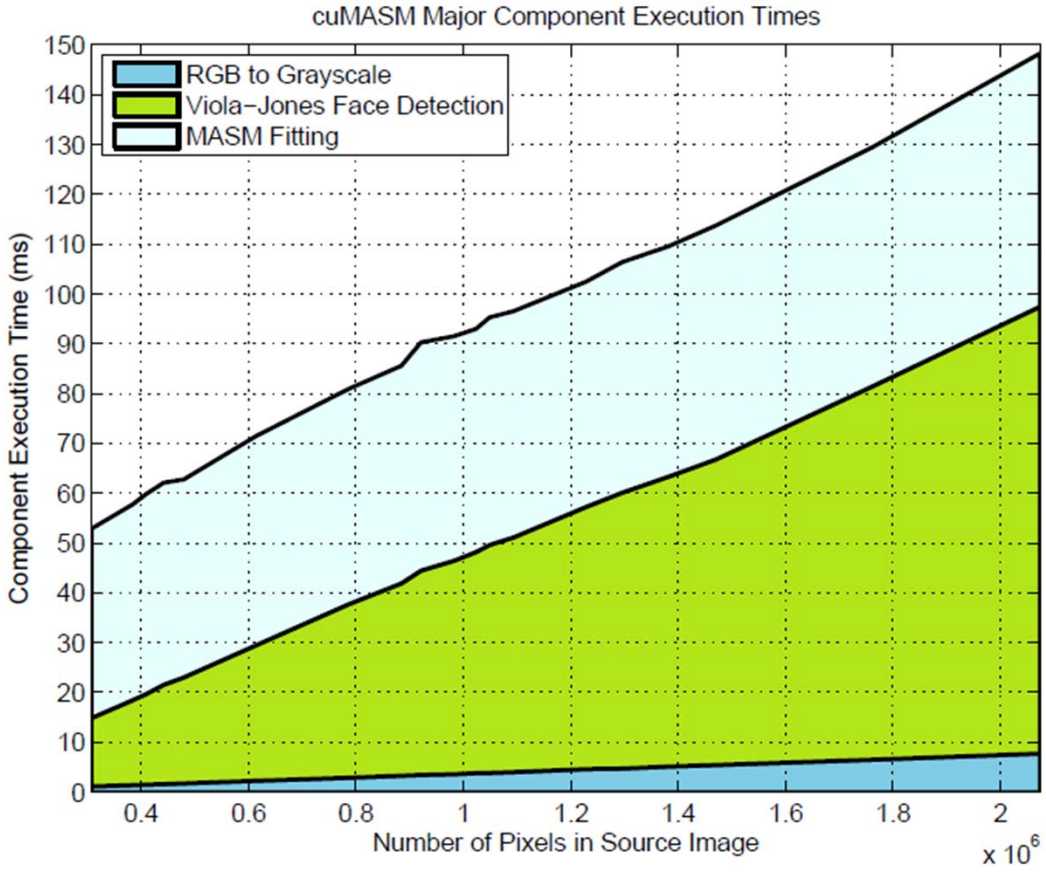
MASM initialization currently provided by OpenCV Viola-Jones (haar-like feature cascade classifier) face detection

- Currently: Re-initialization occurs each frame with new face detection
- Ongoing: Tracking faces using a Kalman filter to update position of each landmark point independently
  - Provides better initialization for subsequent frame than axis aligned box
  - Faster convergence, reduced fitting error
- Future: integration into larger system which depends on landmark points downstream
  - 3D model generation

FUTURE WORK: GPU FACE DETECTION INTEGRATION

At larger image sizes, initialization with OpenCV Viola-Jones face detector dominates runtime MASM

- Clearly not sustainable
- Ongoing: development of GPU implementation of Haar-cascade classifier
- Ongoing: development of GPU multilayer perceptron-based face detector



FUTURE WORK: MULTIPLE CARDS

cuMASM currently only makes use of a single GPU within a given system

- Often computers may have multiple devices (systems capable of supporting up to 8 GPUs exist)
- Implementation could easily be adapted
  - Distribute incoming frames between multiple devices OR
  - Multiple faces per frame OR
  - Partition the 79 points between the cards

CONCLUSIONS

Massively parallel architecture of GPUs has become significantly easier due to advent of high level language support

- Achieving significant acceleration requires awareness of underlying architecture

Achieved 24X speedup over single-threaded, and 12X over fully multithreaded CPU implementation on modern quad core processor

Capable of providing automatic facial landmarking at 20 FPS

Fast face detection is a limiting factor for real systems using HD images