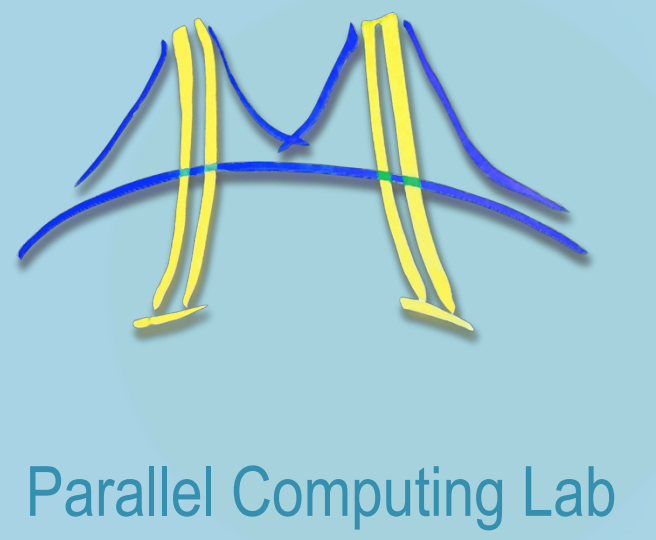




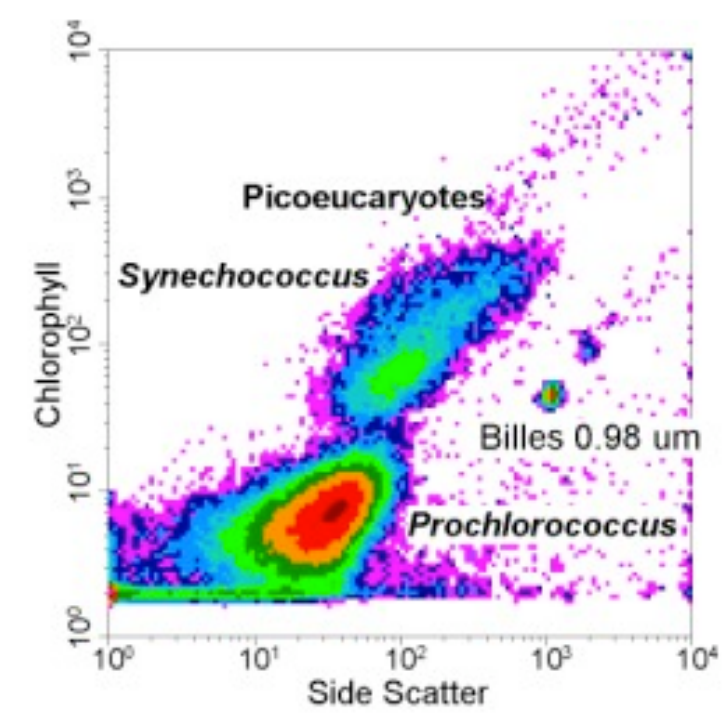
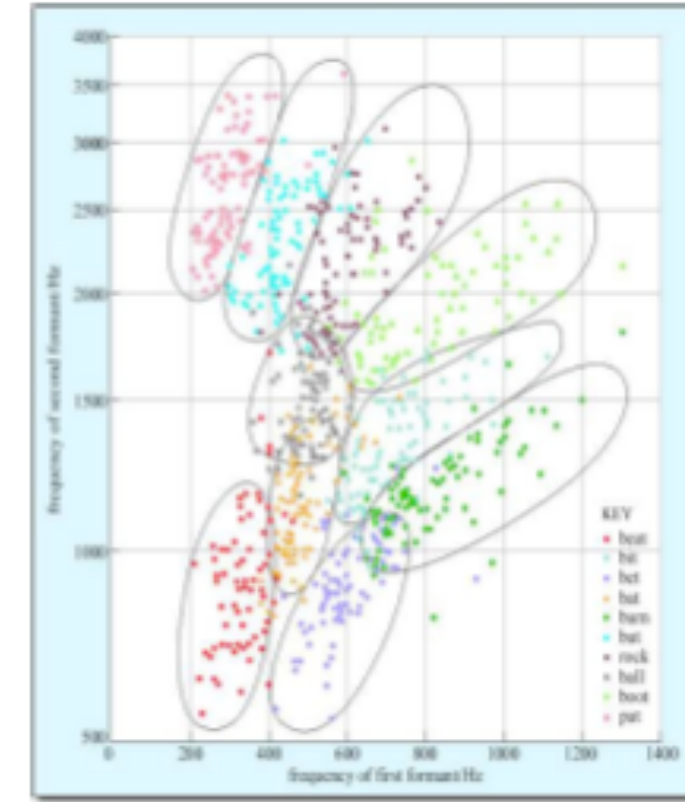
A SEJIT Specializer for Gaussian Mixture Model Computations on Parallel Platforms (With a Case Study in Speaker Diarization)



Katya Gonina, Henry Cook, Adam Jiang, Shoab Kamil, Gerald Friedland, Armando Fox, David Patterson

Gaussian Mixture Models (GMM)

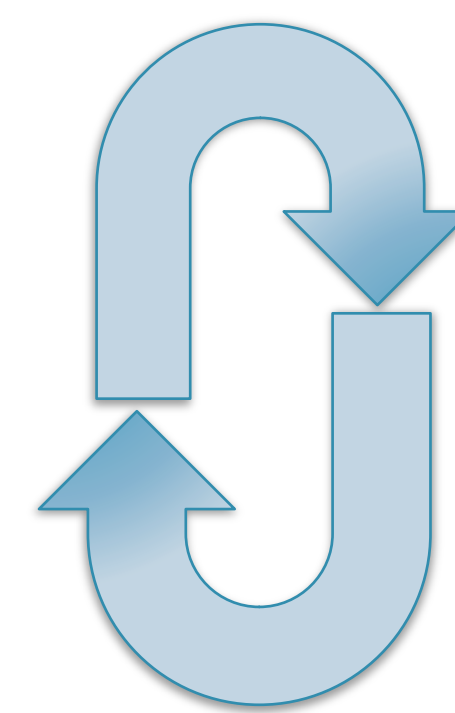
- Probabilistic model for clustering data
- Assumes the distribution of observations follows a mixture of multidimensional Gaussian distributions
- Each Gaussian in the mixture has a mean (μ) and a variance (σ) parameter, as well as a weight (π)



- Example applications
 - Speech Recognition – speaker classification, acoustic modeling for speech recognition
 - Computer Vision – image segmentation, hand writing recognition
 - Biology – flow cytometry
 - Data mining – topics in documents

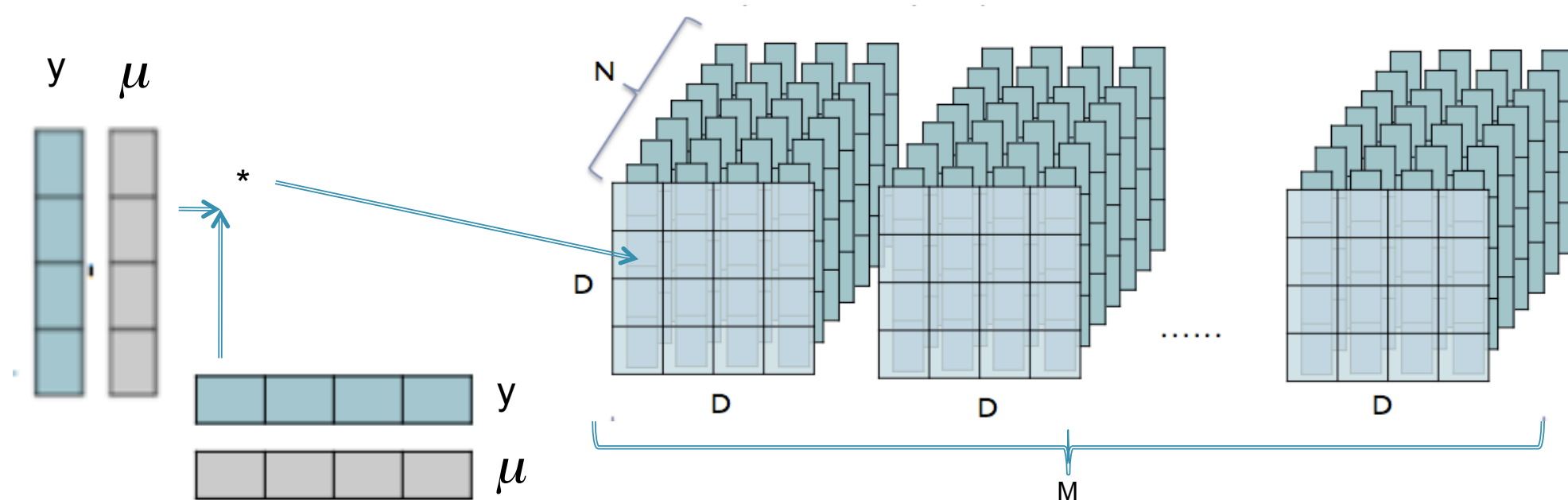
GMM Training (EM algorithm)

- Given a set of observations/events: find the maximum likelihood estimates of the set of Gaussian Mixture parameters (μ, σ, π) and classify observations
- Expectation Maximization (EM) Algorithm
 - E step
 - Compute probabilities of events given model parameters
 - M step
 - Compute model parameters given probabilities
 - Weights, mean, **covariance matrix**
 - Iterate until convergence



Covariance Matrix Computation Code Variants

- Specialization of the covariance matrix computation
- Problem parameters:
 - N – number of events, ~10K-100K
 - D – event dimension, ~10-40
 - M – number of Gaussians (clusters), ~1-128
 - Matrix is symmetric – only compute the lower $D^2/2$ cells
- Platform parameters (GPU):
 - Number of SMs
 - Number of SIMD vector lanes
 - Size of per-block shared memory
 - Size of global memory



Covariance matrix computation dimensions

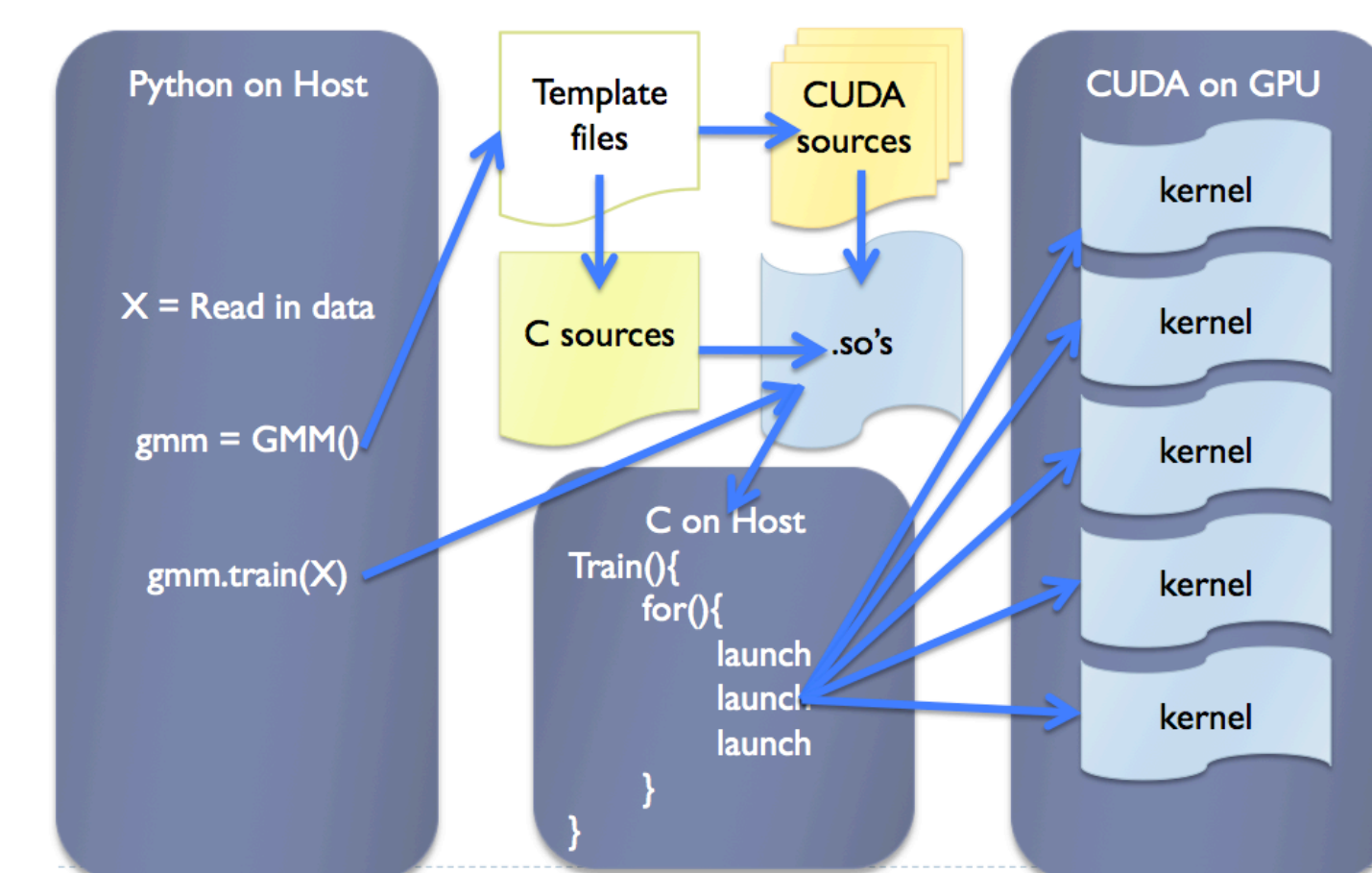
- Optimal-performing code variant depends both on the specific platform and the specific problem parameters
- Need to develop an automatic selection mechanism that intelligently selects between the code variants based on problem and platform parameters

SEJIT Specializer Framework

- High level goal: automatically transform high-level abstraction of a machine learning algorithm to highly efficient parallel code

General Specializer Setup

- Application code is written in Python
- Specialization is done by:
 - Creating templates for both the host and device (CPU and GPU) code in C and CUDA
 - Filling templates with the correct code variant and associated runtime parameters
- ASP Specializer (Mako, CodePy, PyUBLAS)
 - Takes in the problem and platform parameters
 - Selects appropriate code variant (currently tries all and remembers best-performing one)
 - Pulls in the template for the code variant, parameterizes it and compiles to binary



Code Domains

- Python code handles application
 - Manipulates problem data, determines ML targets
- C code that runs quickly on the CPU
 - Allocates GPU memory
 - Performs main EM iterative loop
 - Until convergence, call E step(s) and M step(s)
 - Calls variants of mstep_covariance(events, GMM_model)
- CUDA code that runs quickly on the GPU
 - Defines GPU kernels and their operation
 - Contains kernel code variants

Data Sharing and Allocation

- To allow trained parameters to be read in Python after training, we pass references to data allocated in Python to C
- C code allocates GPU memory and temporary data structures on the CPU, performs training, and copies the data back
- Allocate new event data on demand – i.e. if we're training models on the same data in a loop, we do not allocate and copy event data every iteration

Example Application Code

Agglomerative Hierarchical Clustering

```
def ahc(x):
    # from 50 to 1 cluster
    N_start = 50
    N_end = 1
    rissosen_list = []
    for M in range(N_start, N_end, -1):
        likelihood = self.f.gmm.train(Coeff(X))
        # compute rissosen score for this gmm with M clusters
        rissosen = likelihood + 8.5 * (self.f.gmm.compute_distance_rissosen(c1, c2) - 1) * math.log(Coeff(X) * self.f.gmm.D)
        rissosen_list.append((rissosen, self.f.gmm))
        # find closest clusters and merge
        if M > 2:
            # merge if there is only one cluster
            new_list = []
            for c1 in range(0, self.f.gmm.M):
                for c2 in range(c1 + 1, self.f.gmm.M):
                    new_clusters, dist = self.f.gmm.compute_distance_rissosen(c1, c2)
                    new_list.append((dist, c1, c2, new_cluster))
            # merge the closest
            min_dist, min_c1, min_c2 = self.f.gmm.get_min_merge(new_list)
            self.f.gmm.merge_clusters(min_c1, min_c2, min_c1 + min_c2)
            # keep track of the rissosen score and the corresponding gmm clusters
            rissosen_list.append((rissosen, self.f.gmm))
    return self.f.gmm(rissosen_list) # return gmm with best rissosen score
```

- Perform GMM training within an outer loop that decreases number of clusters
- Select best “fitting” GMM – number of clusters that best describes the event data
- Used in speaker diarization – unsupervised identification of speakers in an audio sample

Probability Computation

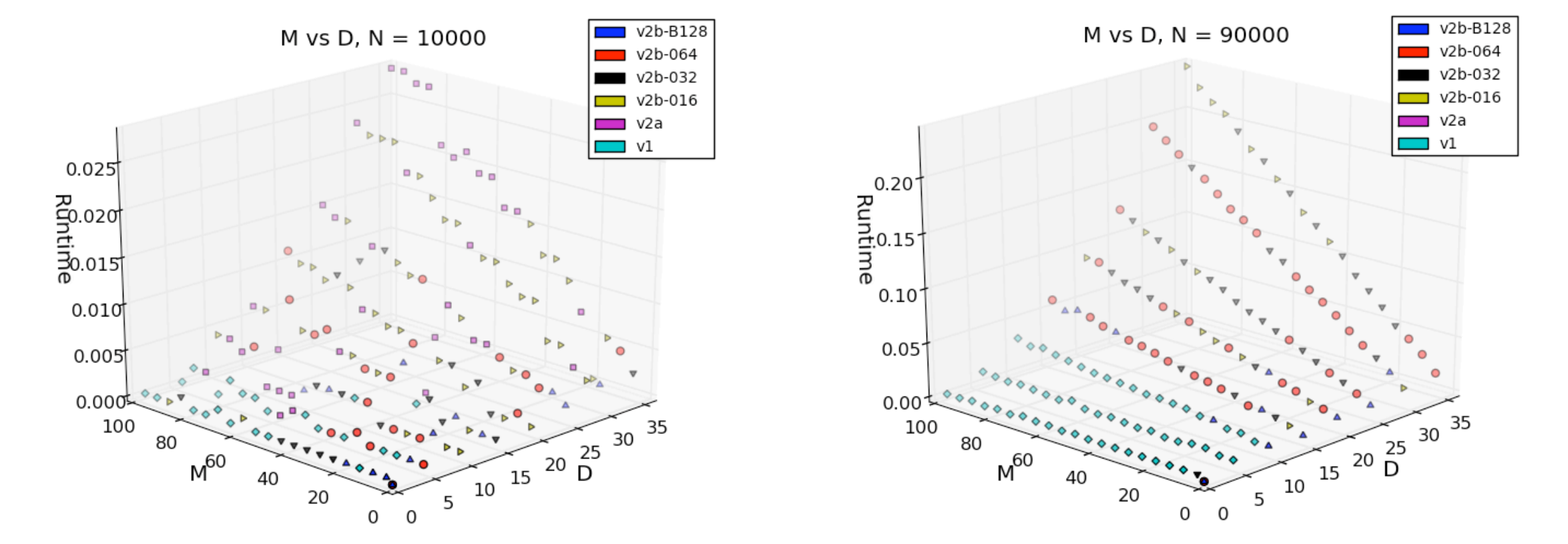
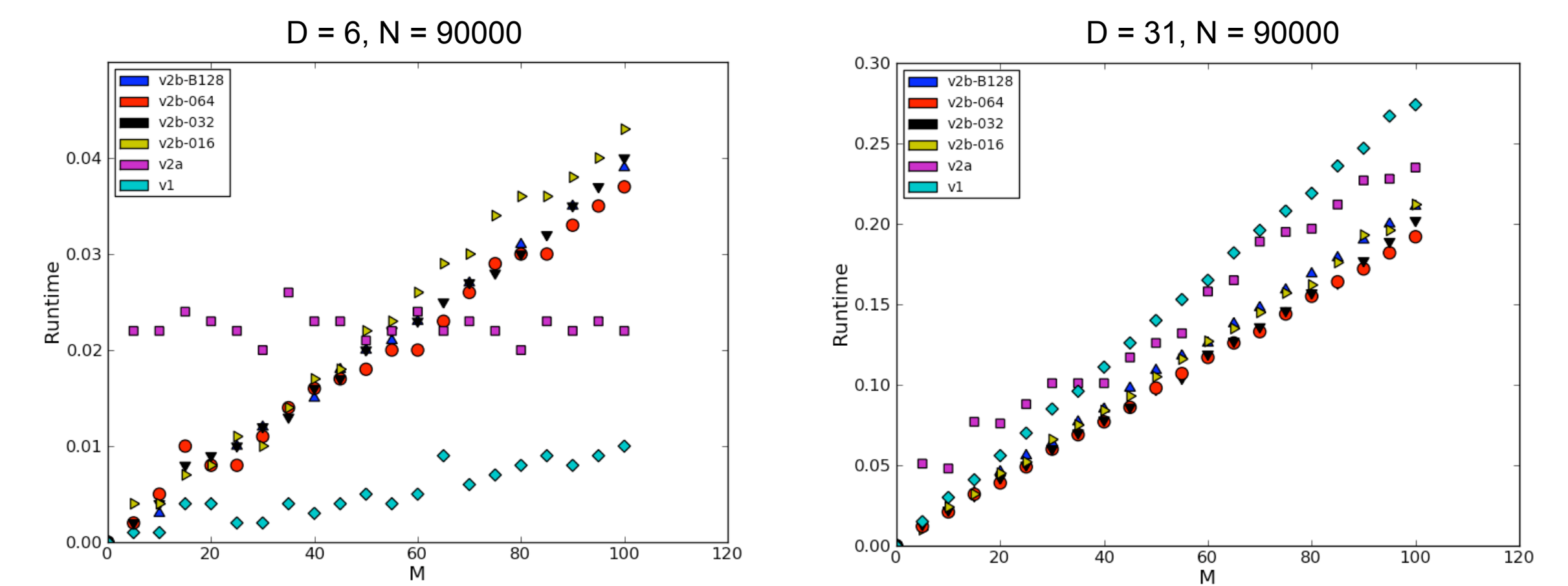
```
X = get_data()
means, covars = get_model()
gmm = GMM(M, D, means, covars)
Y = gmm.predict(X)
```

- Compute the probability of observing an event given the trained model
- Used in speech recognition to compute the observation probability of an audio sample

Results

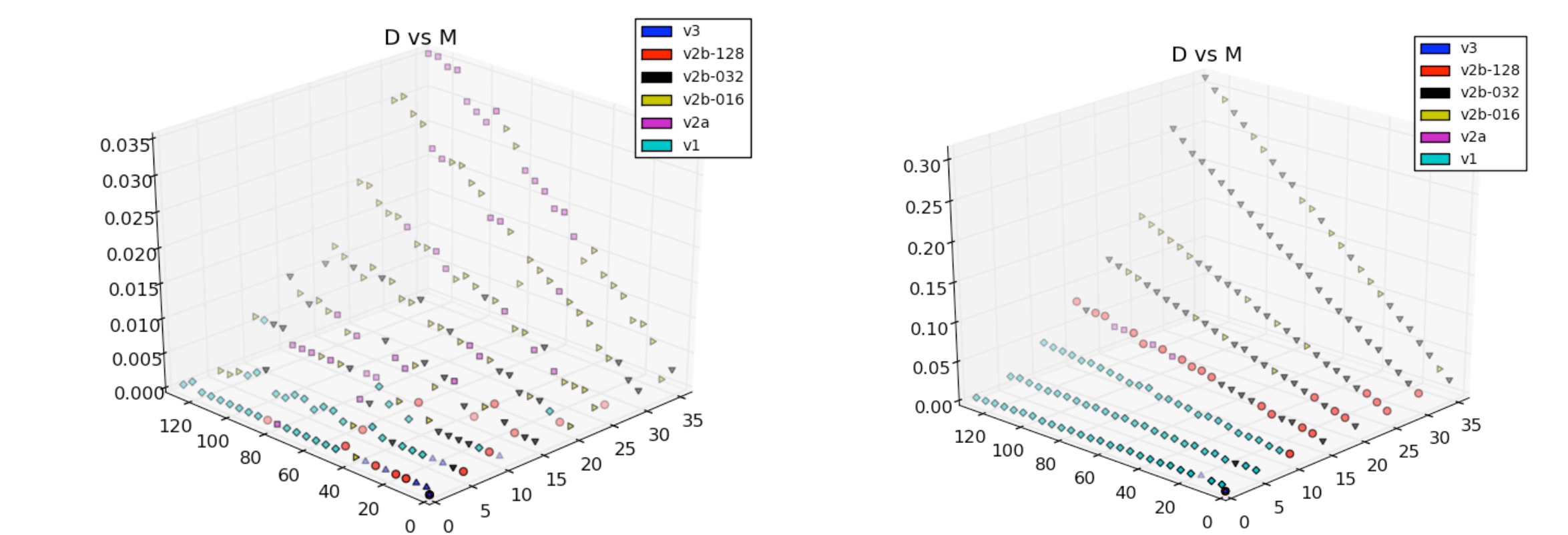
Evaluation platforms:

- GTX480 (Fermi)
 - 14 SM, 32 SIMD, 48K shared mem, 3GB



GTX 285

- 30 SM, 8 SIMD, 16K shared mem, 1GB DRAM



- Code variant selection gave at least 30% performance improvement for problem sizes tested – with larger problems the improvement increases

Future Work

- More intelligent code variant selection mechanism, given platform and problem parameters:
 - Pull from existing database of best-performing code variants
 - Use machine learning to predict the best-performing code variant
- Expand framework to specialize other GMM computations:
 - Probability computation
 - Cluster distance computation functions (BIC, AIC)
- Expand framework to other applications (computer vision, data mining) and architectures (OpenCL, RISC-V)
- Performance improvement of the GMM framework for particular application common use cases to reduce overhead

CV:	1	2	2b	3
N	thread	Sequential loop for each thread	blockIdx.y, sequential loop for each thread within block	thread
M	blockIdx.x	blockIdx.x	blockIdx.x	Sequential loop
D	blockIdx.y	thread	thread	blockIdx.x
SH MEM	Means for each cluster	Means for each cluster, Covariance matrix	Means for each cluster, Partial covariance matrix	Events (if small), Means for all clusters