GPUML: Graphical processors for speeding up kernel machines

http://www.umiacs.umd.edu/~balajiv/GPUML.htm

Balaji Vasan Srinivasan, Qi Hu, Ramani Duraiswami

Department of Computer Science,

University of Maryland, College Park

Workshop on High Performance Analytics – Algorithms, Implementations and Applications

Siam Conference on Data Mining, 2010

Large datasets

- Improved sensors – ease of data collection




- Large datasets
  - Millions of samples (tall)
  - Large number of attributes (fat)

- Objective: extract meaningful information from data

Extracting information from the data

- Raw data to an interpretable version
  - Example: Speech signal → speaker
  - Function estimation: $f: X \rightarrow Y$
- Information extraction categories:
  - Density estimation [evaluating the class membership]
  - Regression [fitting a continuous function]
    - $Y = \mathbf{R}$
    - Example: predicting temperature from historic data
  - Classification [classify into one of the predefined classes]
    - $Y=\{-1,+1\}$
    - Example: Object recognition, speaker recognition
  - Ranking / Similarity evaluation [preference relationships between classes]
    - Example: information retrieval
- *Learn the underlying structure in data for a target application*

Approaches to learning

- Parametric
  - A priori model assumption
  - Use training data to learn “model” parameters
  - Training data discarded after learning
  - Performance ⇔ a priori assumptions

- Nonparametric
  - No model assumption
  - *“Let the data speak for itself”*
  - Retain training data for prediction
  - Better performance
  - Computationally expensive

Kernel machines

- Robust non-parametric machine learning approaches
- At their core: linear algebra operations on matrices of kernel functions
- Given: data in R^d, $\mathbf{X}=\{x_1,x_2,\ldots,x_N\}$;
  - Kernel matrix ⇔ similarity between pairs of data points

$$\mathrm{K} = \begin{pmatrix} k_{11} & \ldots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{N1} & \ldots & k_{NN} \end{pmatrix}$$

- Each element given by a function; for example,

$$\mathrm{k}_{ij} = s\exp\left(-\frac{\|x_i - x_j\|^2}{h^2}\right)$$

Popular kernel machines

- Most of these kernel based learning approaches scale O(N^2) or O(N^3) in time with respect to data

	Training (N examples)	Prediction (at N points)	Choosing parameters
Kernel regression	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Gaussian processes	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^3)$
SVM	$\mathcal{O}(N_{sv}^3)$	$\mathcal{O}(N_{sv}N)$	$\mathcal{O}(N_{sv}^3)$
Ranking	$\mathcal{O}(N^2)$		
KDE		$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Laplacian eigenmaps	$\mathcal{O}(N^3)$		
Kernel PCA	$\mathcal{O}(N^3)$		

- There is also O(N^2) memory requirement in many of these
- This is undesirable for very large datasets

Computational bottlenecks in kernel machines

1. Weighted kernel summation $\sum_{i=1}^{N} q_i k(x_i, x)$; $\mathbf{Kq}$
   - Ex. Kernel density estimation
   - $O(N^2)$ time and space complexity
2. Kernel matrix-vector product within iterative solvers
   - Ex. kernel PCA, kernel ridge regression
   - $O(N^2)$ time and space complexity per iteration
3. Kernel matrix decompositions (Cholesky/QR)
   - $O(N^3)$ time and $O(N^2)$ space complexity

Objective

- Address the scalability issue (time and memory) using GPUs
- Illustrate in several learning applications
  - Kernel density estimation
  - Mean shift clustering
  - Gaussian process regression
  - Ranking
  - Spectral regression kernel discriminant analysis (SRKDA)

Overview

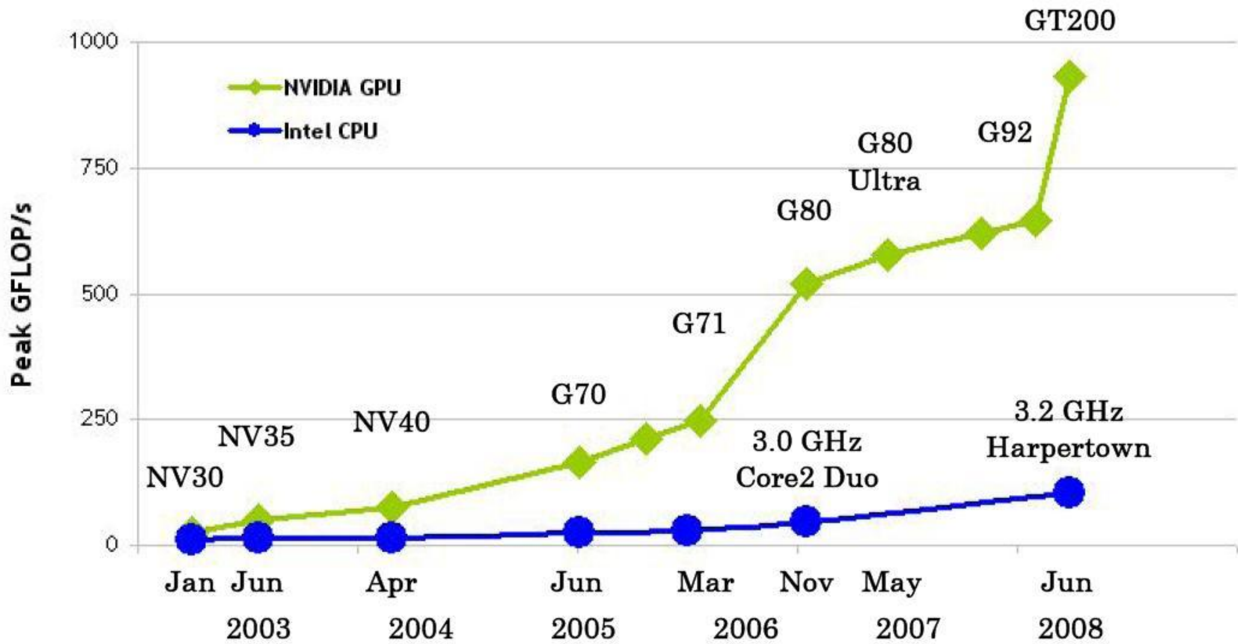
- Graphical processors
  - CUDA architecture
- Category1: Kernel summation
  - Algorithm
  - **Application:** *Kernel density estimation*
- Category2: Iterative formulation
  - **Application:** *Mean shift clustering*
  - **Application:** *Gaussian process regression*
  - **Application:** *Ranking*
- Category3: Kernel decomposition
  - Approach
  - **Application:** *Spectral regression kernel discriminant analysis*

Graphical processing units (GPU)

Graphics processors

- Graphics processors were developed to cater to the demands of real-time high definition graphics
- Graphics processing units (GPU)
  - Highly parallel, multi-core processor
  - Tremendous computational horsepower
  - High memory bandwidth
- General purpose computation (GPGPU)
  - Single program multiple data architecture
  - High arithmetic intensity

CPU vs GPU



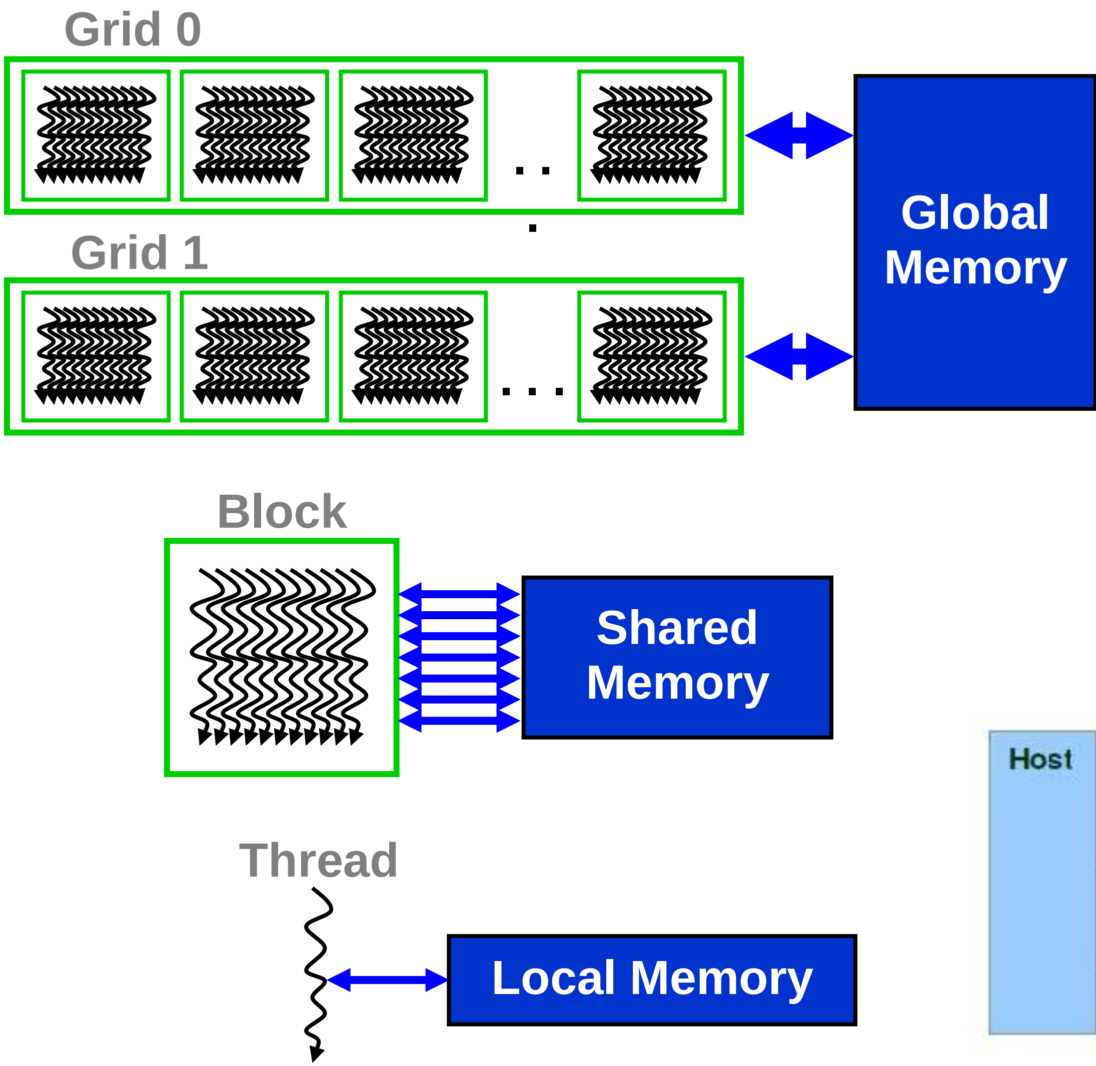
GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

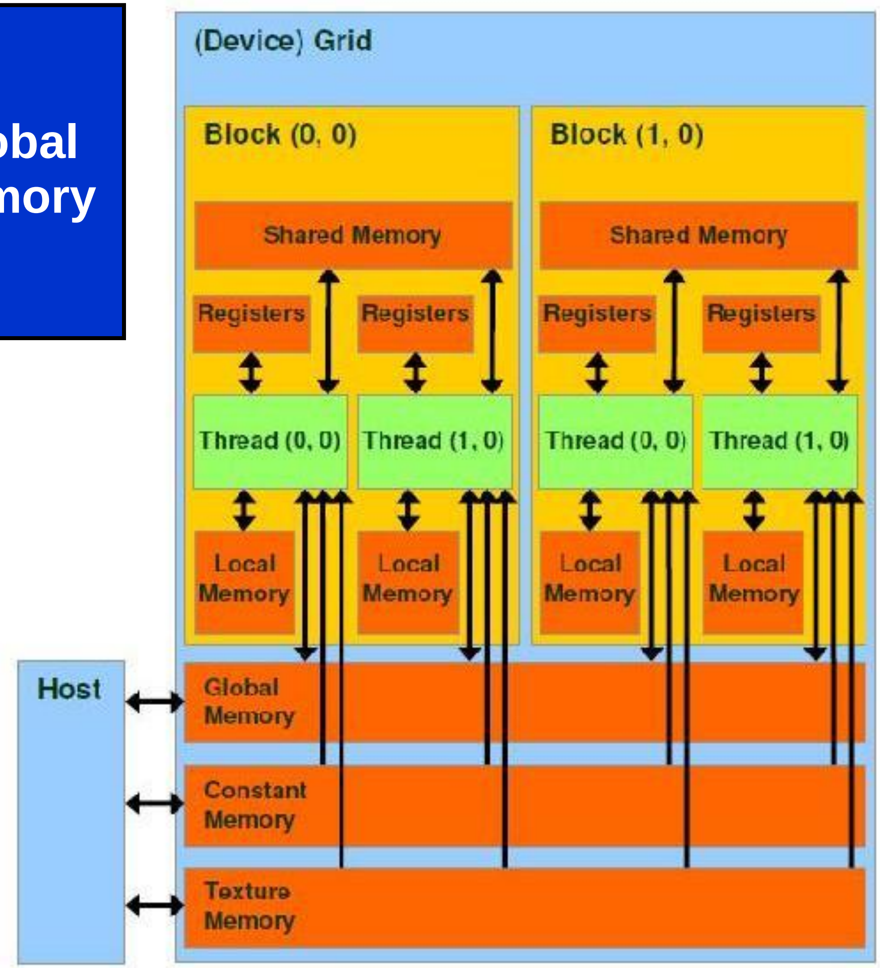
Figure from: NVIDIA CUDA Programming Guide 3.0. 2010

Compute Unified Device Architechture (CUDA)

- NVIDIA introduced CUDA in November 2006
  - Resulted in GPUs to be viewed as a bunch of parallel coprocessors assisting the main processor
  - Result in more easier use of GPUs for general purpose problems
- Different GPU memories
  - Global memory - access time: 400 clock cycles
    - Cheaper to access consecutive memory locations
  - Shared memory & Registers
    - Cheapest access time, as less as an instruction execution time
- Main concerns in GPU
  - Memory accesses
- Transfer to local cache and operate on this data.

CUDA Memory Model





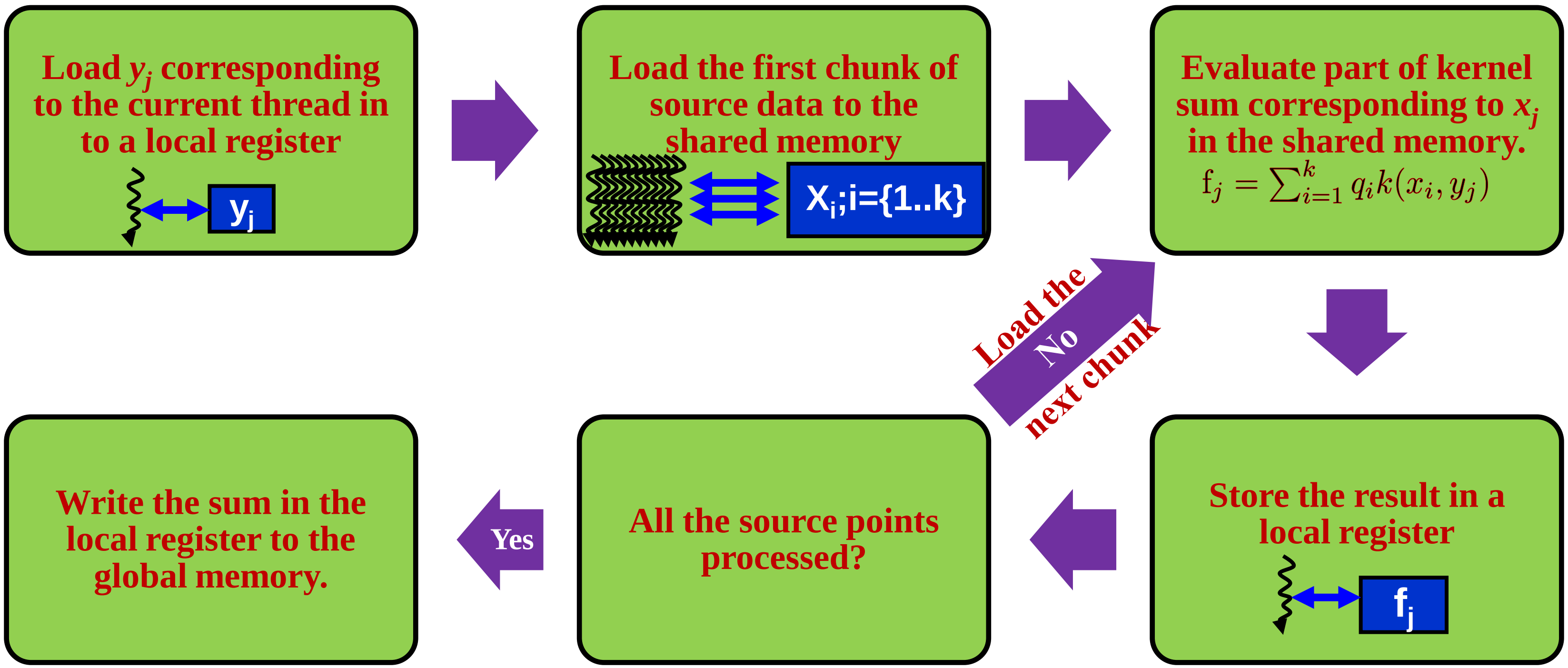
Category1: Kernel summation

Kernel summation on GPU

- Data:
  - Source points x_i, i=1,…,N,
  - Evaluation points y_j, j=1,…,M

$$f_j = \sum_{i=1}^{N} q_i k(x_i, y_j)$$

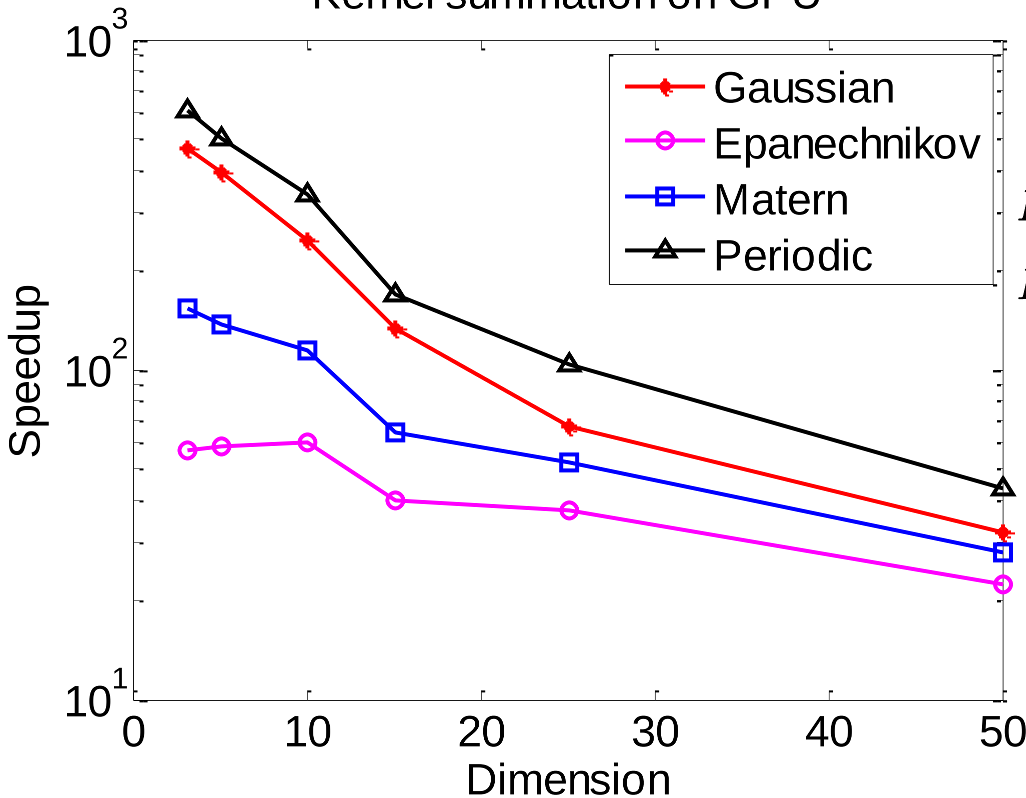
- Each thread evaluates the sum corresponding to one evaluation point:
- Algorithm:



GPU based speedups

$$\sum_{i=1}^{N} q_i k(x_i, x); \quad \mathbf{Kq}$$

Kernel summation on GPU



$$K(x_i, x) = s \exp\left(-\frac{(d(x_i, x))^2}{2}\right),$$

$$K(x_i, x) = s(1 - d(x_i, x)^2) \times 1(d(x_i, x) < 1),$$

$$K(x_i, x) = s(1 + \sqrt{3}d(x_i, x)) \times \exp(-\sqrt{3}d(x_i, x)),$$

$$K(x_i, x) = s \exp\left(-2\sin^2(\pi * d(x_i, x))\right),$$

Advantages:

Can be easily extended to any kernel

Performs well up to 100 dimensions

Disadvantages:

Memory restrictions

Quadratic time complexity

CPU: Intel Quad core processor

GPU: Tesla C1060

FIGTREE

- Algorithmic acceleration of Gaussian summation
  - Guaranteed ε-error bound
- Automatically tunes between two O(*N*) approaches
  - Tree-based approach (low Gaussian bandwidths)
  - Improved fast Gauss transform (large Gaussian bandwidths)
- Advantage: O(*N*)
- Disadvantage: time advantage only up to 10 data dimensions

Yang, C., Duraiswami, C., and Davis, L. **Efficient kernel machines using the improved fast gauss transform.** *In Advances in Neural Information Processing Systems*, 2004.

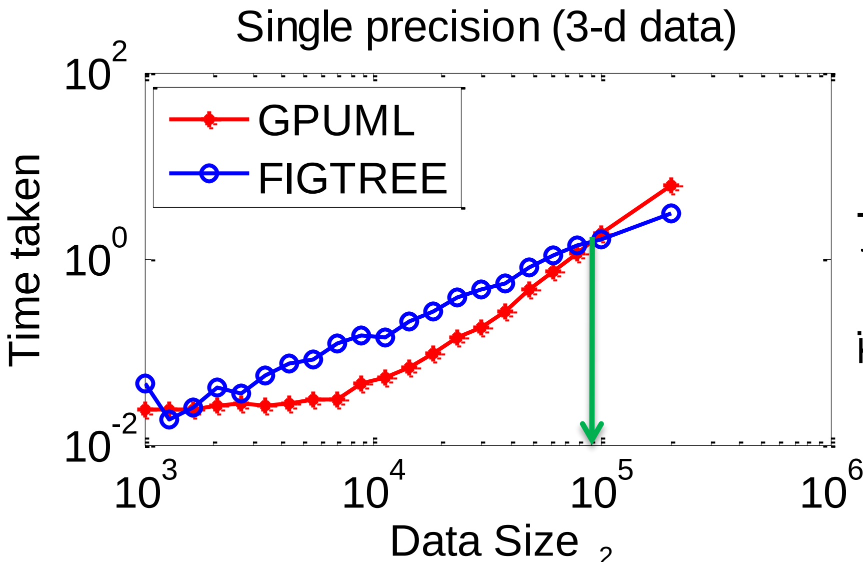
D. Lee, A. Gray, and A. Moore. **Dual-tree fast Gauss transforms.** *In Advances in Neural Information Processing Systems 18, pages 747–754. 2006.*

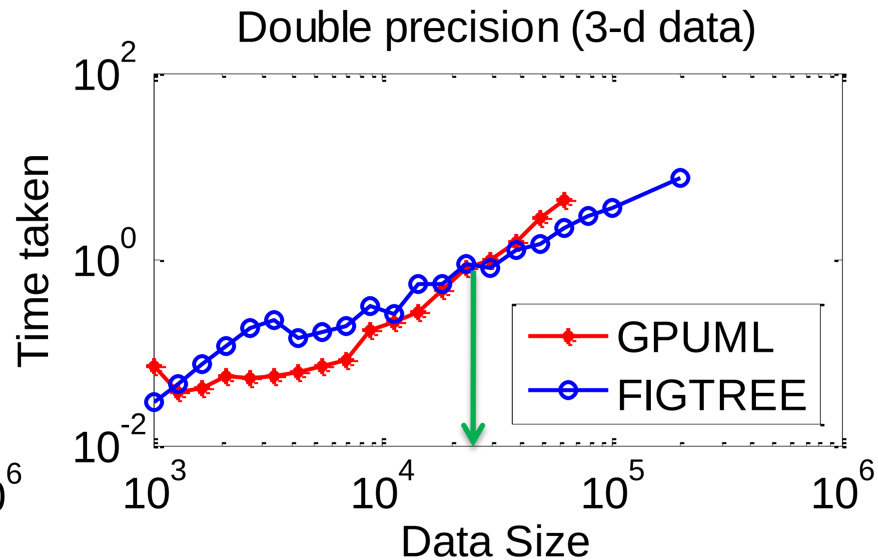
Raykar, V.C. and Duraiswami, R. **The improved fast Gauss transform with applications to machine learning.** *In Large Scale Kernel Machines*, pp. 175–201, 2007.

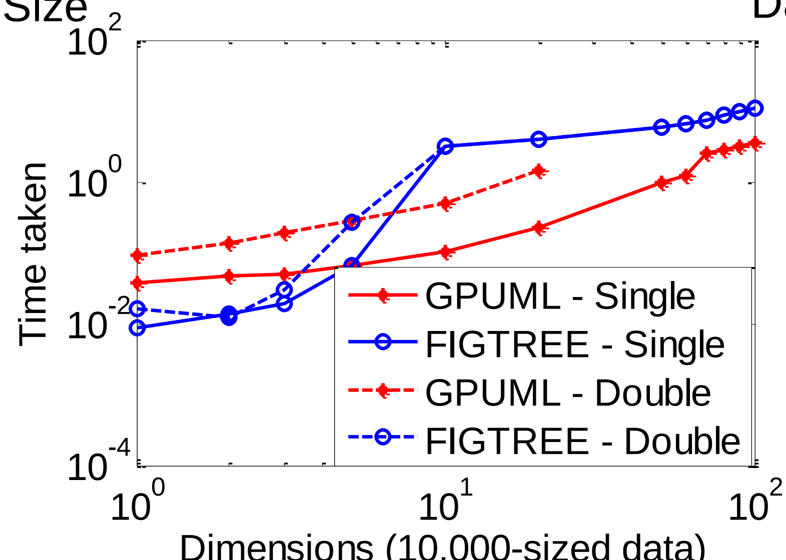
Morariu, V., Srinivasan, B.V., Raykar, V.C., Duraiswami, R., and Davis, L. **Automatic online tuning for fast Gaussian summation.** *In Advances in Neural Information Processing Systems*, 2008.

Available at: http://www.umiacs.umd.edu/~morariu/figtree/

GPUML vs FIGTREE



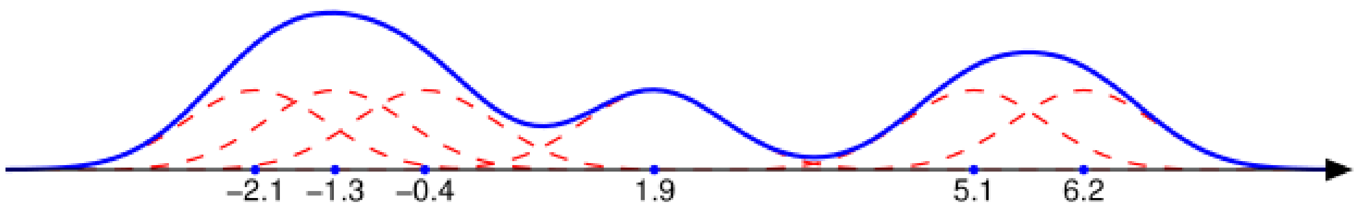




Application1: Kernel Density Estimation

- Non-parametric way of estimating probability density function of a random variable

$$f(x) = 1/(Nh) \sum_{i=1}^{N} K(x, x_i)$$



- Two popular kernels: Gaussian and Epanechnikov

$$K(x_i, x) = s \times \exp\left(-\frac{(d(x_i, x))^2}{2}\right), \quad (1)$$

$$K(x_i, x) = s \times (1 - d(x_i, x)^2) \times 1(d(x_i, x) < 1), \quad (2)$$

Application1: Results on standard distributions

- Performed KDE on 15 normal mixture densities from [1] based on 10,000 samples:

Gaussian kernel	CPU time	25.14s
	GPU time	**0.02s**
	Mean absolute error	~10^{-7}
Epanechnikov kernel	CPU time	25.11s
	GPU time	**0.01s**
	Mean absolute error	~10^{-7}

1. J. S. Marron and M. P. Wand, "**Exact Mean Integrated Squared Error**", *The Annals of Statistics*, Vol. 20, No. 2, 712-736, 1992

Category2: Iterative formulations

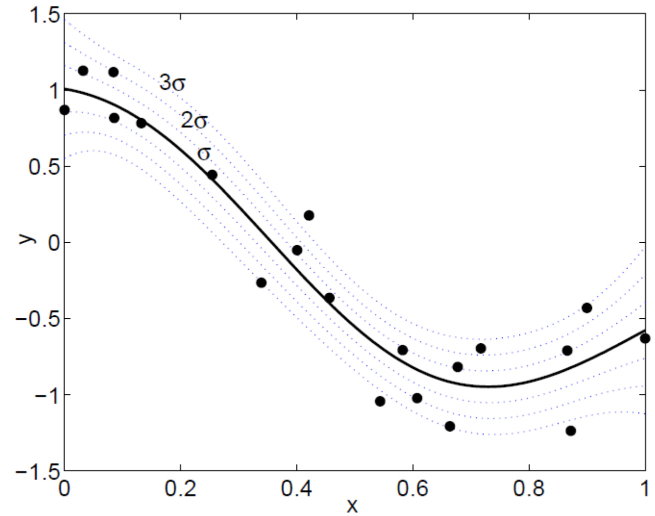
Application2: Mean shift clustering

- Mode seeking approach
- Gradient ascent with kernel density estimates

- Took only ~15s to converge against 1.7 hours by a direct approach

D. Comaniciu and P. Meer, "**Mean shift: a robust approach toward feature space analysis**", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (2002), pp. 603–619.

Application3: Gaussian Process Regression



- Bayesian regression
- Given data $D = \{x_i, y_i\}$, $i=1..N$
  - Learn: $y = f(x) + \epsilon, \epsilon \sim N(0, \sigma^2)$
  - Test point x_*, need to find $f(x^*)$ or f_*
- Gaussian process:
  - $f(x)$: zero-mean Gaussian process
  - Process variance: $K(x, x')$ ⇔ kernel function
- For Gaussian noise: $P(f_*|D,x_*) = N(m,V)$
  - $m = k_*(x)(K+\sigma^2 I)^{-1} y$
  - $V = k(x_*,x_*) - k_*(x)\,(K+\sigma^2 I)^{-1} k_*(x)$
  - K = kernel matrix of training data
  - k^* = kernel vector of test point w.r.t all training data

C. Rasmussen and C. Williams. **Gaussian Processes for Machine Learning.** The MIT Press, 2005.

Application3: Gaussian Process Regression

- GPR model ➔ $f_* = k_*(x)(K+\sigma^2 I)^{-1} y$
- Complexity – $O(N^3)$: solving the linear system
- Alternative1: Low ranked approximation[1]
  - Train using a rank-m ($m<N$) approximation to matrix 'K' to get $O(m^2N)$
- Alternative2: Train on a subset (size $m<N$) of the actual data[1]
- Alternative3: $O(kN^2)$ using iterative solvers like *Conjugate gradient*[1]
  - Accelerate each iteration using GPU
  - Have also designed a novel preconditioner for better convergence[1]

1. C. Rasmussen and C. Williams. **Gaussian Processes for Machine Learning.** The MIT Press, 2005 (chapter 8)
2. Srinivasan BV, Duraiswami R, Gumerov N, "***Fast matrix-vector product based FGMRES for kernel machines***", 11th Copper Mountain Conference on Iterative Methods, April 2010

Application3: GPR using GPUML

Dataset	d	N	CPU	GPU	GPU with preconditioner
Boston housing	13	506	1.8s (23)	**0.11s** (23)	0.43s **(3)**
Stock	9	950	6.6s (28)	**0.174s** (28)	0.786s **(4)**
Abalone	7	4177	105s (25)	0.6s (26)	**0.4s** **(2)**
Computer activity	8	4499	920s (48)	6s (47)	**3.5s** **(3)**
California housing	9	950	--	28s (84)	**39s** **(2)**
Sarcos	27	44440	--	1399s (166)	**797s** **(4)**

Iterations to converge shown in braces

Application4: Ranking

- Information retrieval
  - Given features X_i and X_j
  - Learn preference relationships between X_i & X_j
- Ranking function: $f: R^d \rightarrow R$
  - $f(X_i)>f(X_j)$ if X_i preferred over X_j
- Maximize Wilcoxon-Mann-Whitney statistic

$$\mathbf{WMW}(f) = \frac{\sum_{\epsilon_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \mathbf{1}_{f(x_l^j) \geq f(x_k^i)}}{\sum_{\epsilon_{ij}} \sum_{k=1}^{m_i} \sum_{l=1}^{m_j} \mathbf{1}} \Rightarrow \log \prod_{\epsilon_{ij}} \prod_{k=1}^{m_i} \prod_{l=1}^{m_j} Pr\left[f_w(x_l^j) > f_w(x_k^i) | w \right]$$

$$\Downarrow$$

$$\text{erfc}() \Leftarrow \sigma\left[w^T (x_l^j - x_k^i) \right]$$

G. Omer, R. Rosales, and B. Krishnapuram, "**Learning rankings via convex hull separation**", in *Advances in Neural Information Processing Systems*, 2006, pp. 395–402.

V. Raykar, R. Duraiswami, and B. Krishnapuram, "**A fast algorithm for learning a ranking function from large-scale data sets**", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, pp. 1158–1170.

Application4: Ranking

Dataset	d x N	Raykar et al.	GPU	Error in WMW statistic	
				Training data	Test data
Auto	8 x 392	0.75s	0.52s	$\sim 10^{-4}$	$\sim 10^{-4}$
California housing	9 x 20640	105s	28s	$\sim 10^{-3}$	$\sim 10^{-3}$
Computer Activity	22 x 8192	5.6s	5.5s	$\sim 10^{-4}$	$\sim 10^{-4}$
Abalone	8 x 4177	10s	5s	$\sim 10^{-3}$	$\sim 10^{-3}$

V. Raykar, R. Duraiswami, and B. Krishnapuram, "**A fast algorithm for learning a ranking function from large-scale data sets**", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, pp. 1158–1170.

Category3: Kernel decomposition

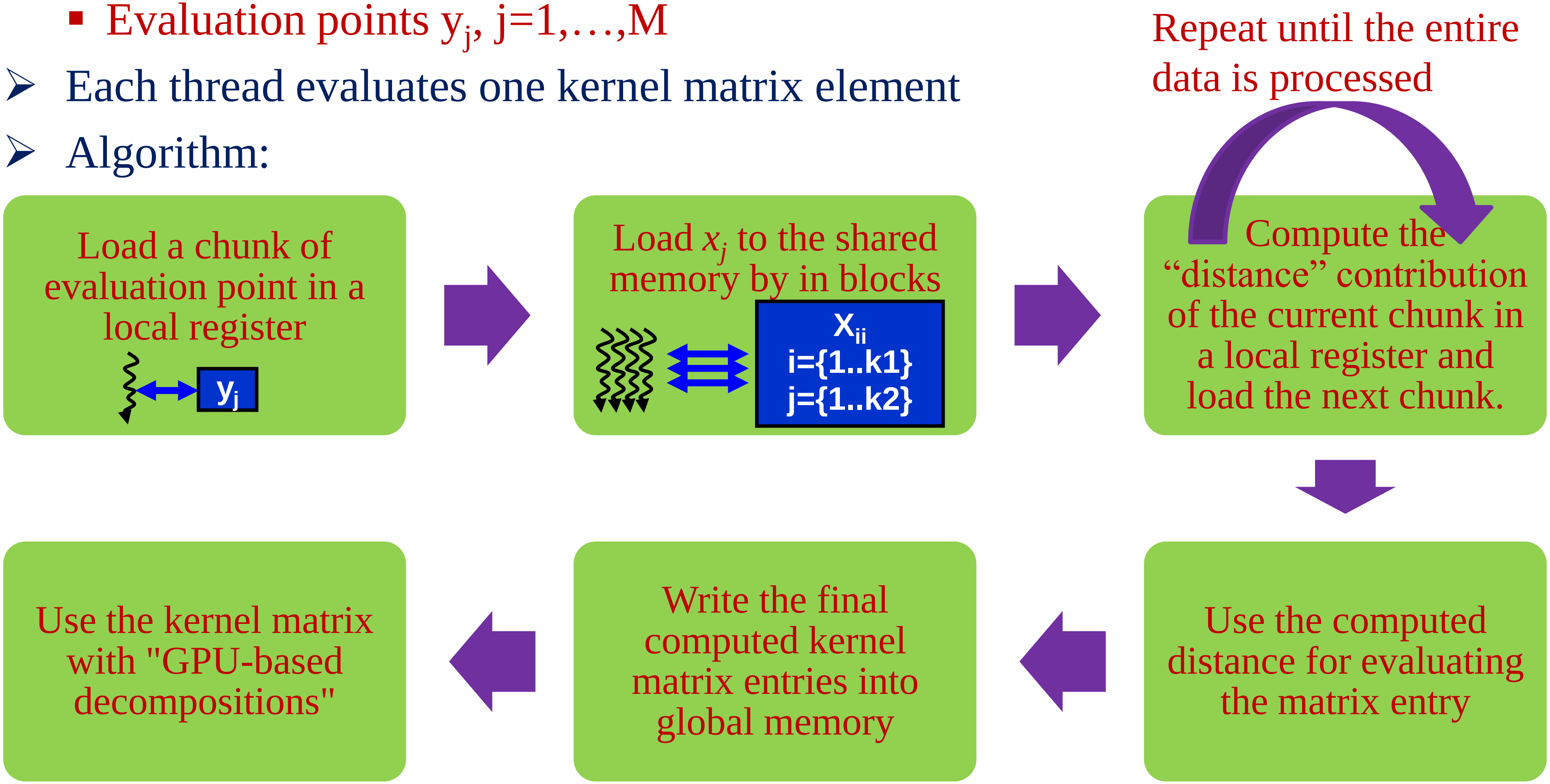
Cholesky / QR decompositions on GPU

- Several GPU-based approaches exist
  - Can be used as is!
- As data size/dimension increase
  - Kernel construction → bottleneck
- Solution:
  - Construct kernel matrix on GPU
  - Use accelerated decompositions

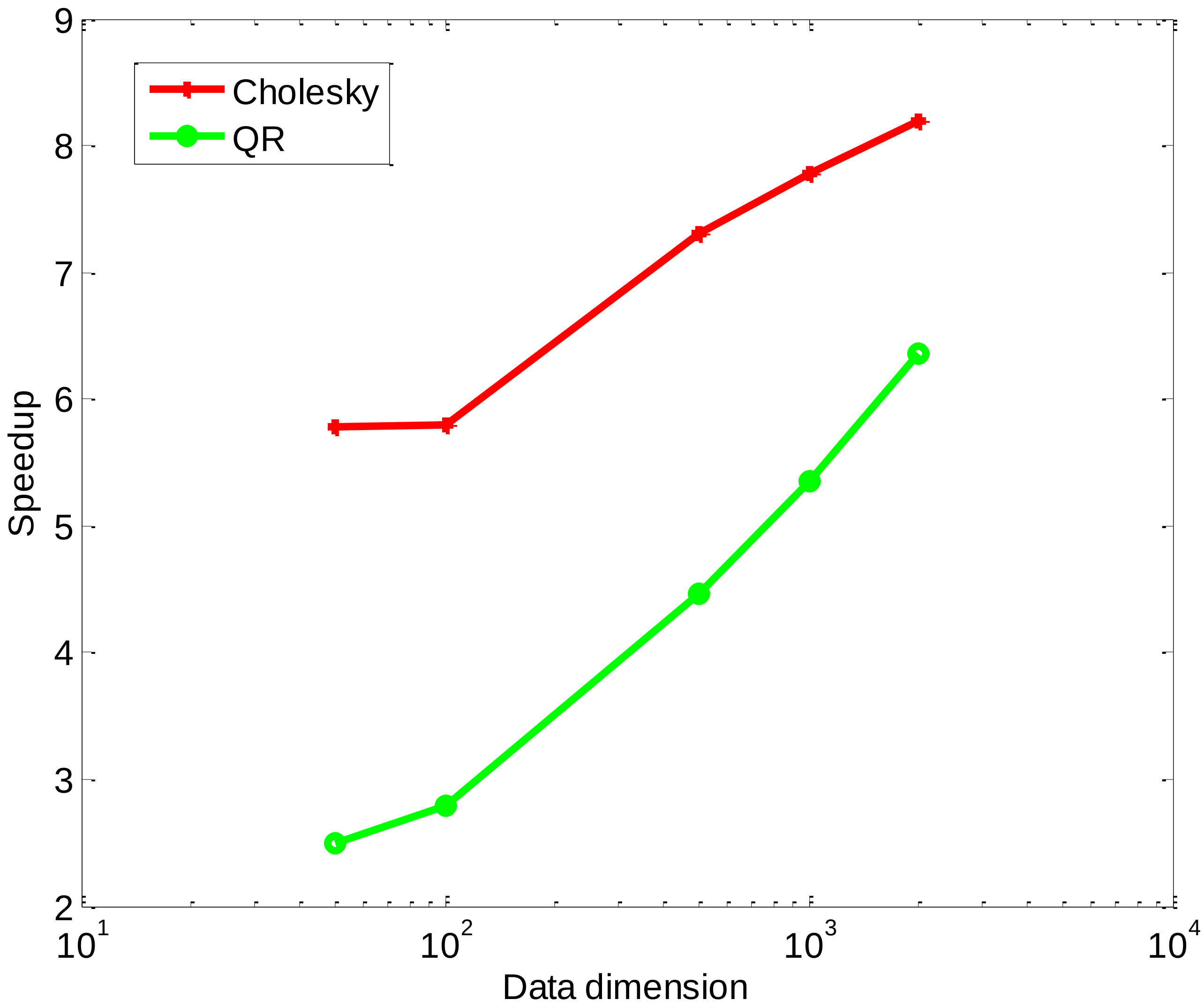
V. Volkov and J. Demmel, **"LU, QR and Cholesky factorizations using vector capabilities of GPUs"**, *Tech Rep. UCB/EECS-2008-49, EECS Department, University of California, Berkeley,* May 2008.

Kernel construction on GPU

- Data:
  - Source points x_i, i=1,…,N,
  - Evaluation points y_j, j=1,…,M
- Each thread evaluates one kernel matrix element
- Algorithm:



Kernel decomposition on GPU



Application5: SRKDA

- Linear Discriminant Analysis (LDA):
  - Maximize inter-class variance
  - Minimize intra-class variance
- Kernel Discriminant Analysis (KDA)
  - LDA in kernel space
  - Eigen decomposition of kernel matrix
- SRKDA
  - Cast KDA as a spectral regression problem
  - Solve kernel system using Cholesky decomposition

DataSize	Direct	GPU
1000	0.6s	0.3s
2500	4.4s	2.1s
5000	22s	12s
7500	60s	37s

D. Cai, X. He, and J. Han, **"Efficient kernel discriminant analysis via spectral regression"**, *in IEEE International Conference on Data Mining*, IEEE Computer Society, 2007, pp. 427–432

Summary

- Kernel machines ➔ robust, but computationally expensive
  - Lack of scalability
- Address this using GPUs
- Illustrated with:
  - Kernel density estimation
  - Mean shift clustering
  - Gaussian process regression
  - Ranking
  - Spectral Regression KDA
- Released as an open source package, **GPUML**
  - *http://www.umiacs.umd.edu/~balajiv/GPUML.htm*

GPUML: Graphical processors for speeding up kernel machines

http://www.umiacs.umd.edu/~balajiv/GPUML.htm

Balaji Vasan Srinivasan, Qi Hu, Ramani Duraiswami
Department of Computer Science,
University of Maryland, College Park

Workshop on High Performance Analytics –
Algorithms, Implementations and Applications
Siam Conference on Data Mining, 2010