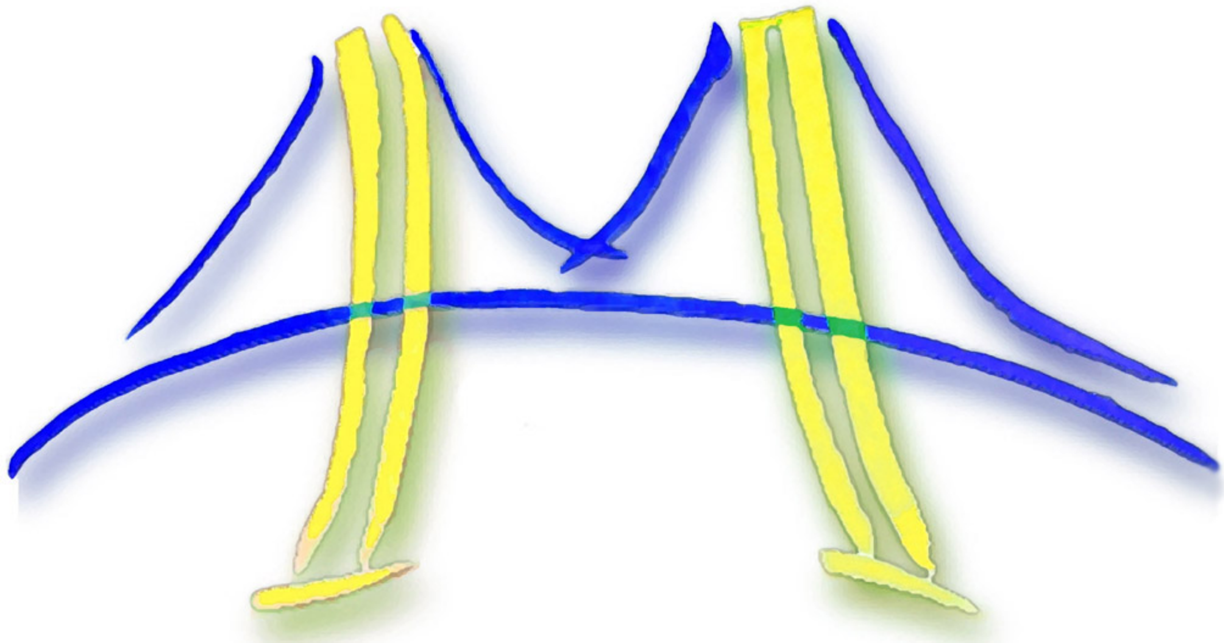


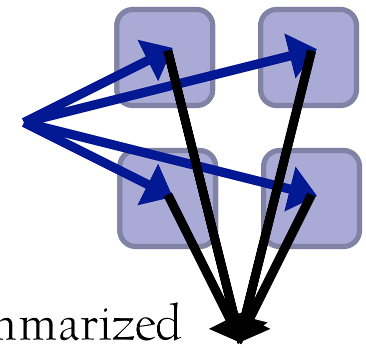
A Map Reduce Framework for Programming Graphics Processors

Bryan Catanzaro
Narayanan Sundaram
Kurt Keutzer

Overview

- Map Reduce is a good abstraction to map to GPUs
  - It is easy for programmers to understand a computation in terms of Map Reduce
  - Map Reduce can map well to GPUs
- Programming efficient Map Reduce on the GPU can be hard
  - Reduction is a global operation, requiring coordination
- We show how a code generation framework for Map Reduce can ease programming and provide high performance

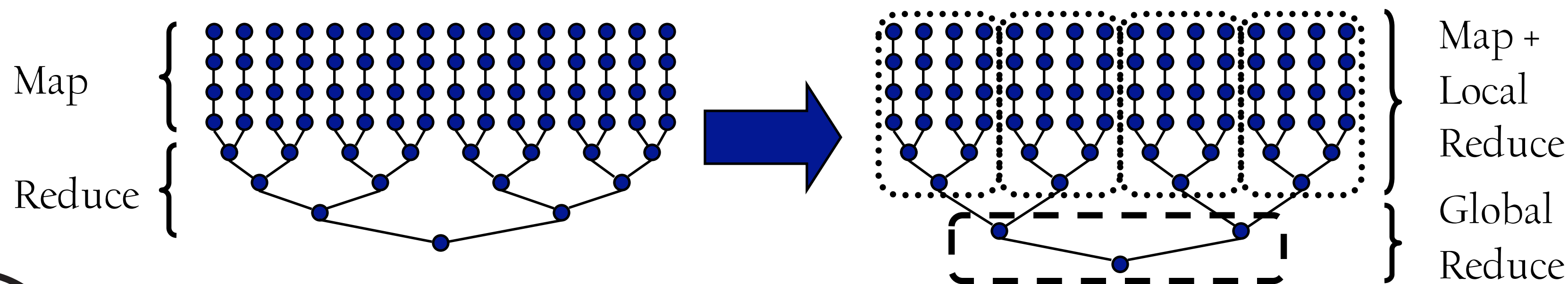
What is Map Reduce?



- "Map Reduce" can mean various things
- To us, it means
  - A map stage, where threads compute independently
  - A reduce stage, where the results of the map stage are summarized
- This is a pattern of computation and communication
  - Not tied to key/value pairs, etc...
- We consider Map Reduce computations where:
  - Each instance of a map function produces one set of outputs
  - Each of a set of reduce functions, gated by per element predicates, produces a set of outputs

Map Reduce on the GPU

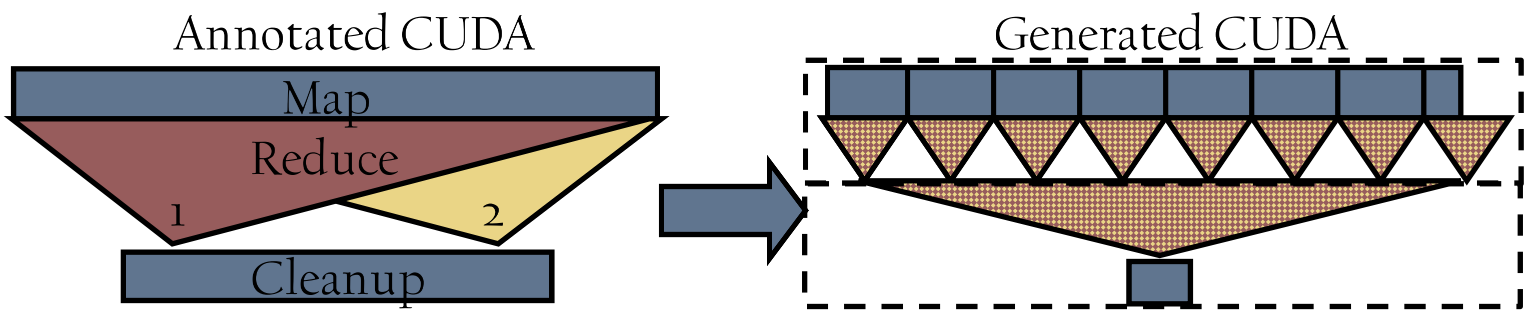
- GPUs are well suited for the Map phase of Map Reduce
  - Lots of parallelism to execute independent threads, multithreading, high bandwidth
- The reduce phase is more difficult, since it introduces dependences
- The natural dependence graph must be restructured to provide these dependences
  - Only local communication allowed, global synchronization very expensive



Reduction on the GPU

- It's well known that efficient reductions on the GPU are difficult
- Many choices
  - How much serialization
  - How much loop unrolling
- Pitfalls
  - Tree structure of reduction can map poorly to SIMD architectures
  - Bank conflicts
- Strongly data size dependent
  - The best reduction for one data set size may be 60x worse than the best for another data set size
- Solution: Have a framework take care of the reductions
  - At present, we provide two variations of a logarithmic reduction, that differ in their loop unrolling

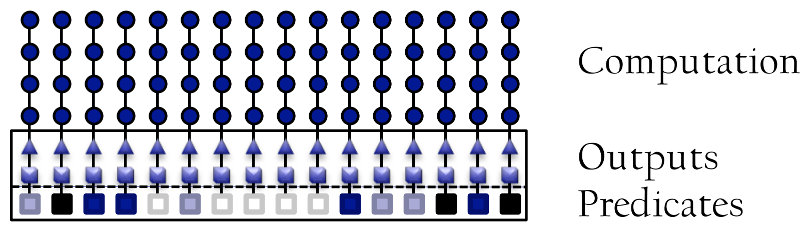
Code Generation Framework



- Our framework takes as inputs:
  - A Map function, written in CUDA, which produces:
    - A set of outputs in local memory
    - A set of predicates, controlling how the outputs should be used in the various reduce functions
  - A set of binary reduce operators
  - And a cleanup function which operates on the outputs of the reductions
- And creates:
  - A map + local reduce function
  - A global reduce + cleanup function

Map Reduce: Map

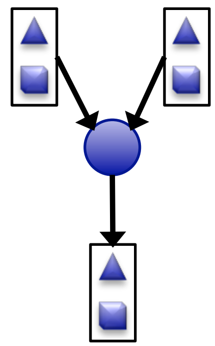
- Map function produces outputs in local memory
  - Each output is an array with one entry per thread



- The Map function produces predicates which gate participation of each output in each reduction function
  - The predicates are stored in an array of integers, one entry per thread
    - This means we currently don't support more than 32 reduce functions, since we use a bit in every entry for each reduce
  - Predicates provide algorithmic selection (limited "keys" from Google MR) and solve thread count boundary issues

Map Reduce: Reduce

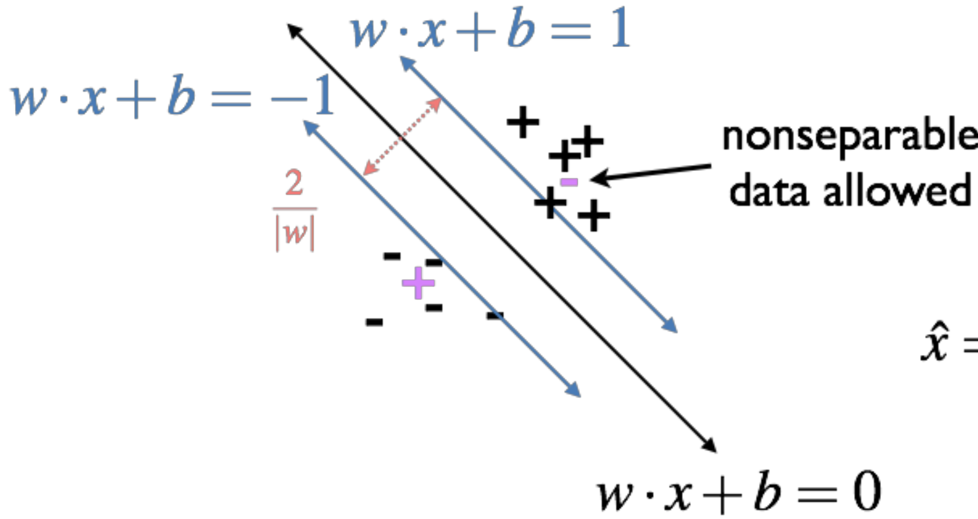
- Reduce operators take two sets of inputs, and produce one set of outputs



- Reduce operators must be associative (or at least pseudo-associative, like floating-point add)
  - This gives us flexibility to restructure the reduction however is best
- Reduce operators must provide an identity value

Support Vector Machines

- SVMs are a popular binary classification technique
  - Recognition, Bioinformatics, Text processing, Network security, etc.
- The idea is to find a hyperplane separating labeled training data with maximal margin
- New points are classified against the hyperplane
- Maximal margin criterion provides generality
- Use of kernel functions allows for nonlinearity



$$\hat{x} = \mathrm{sgn}(w \cdot x + b)$$

SVM Training

- Quadratic Program

$$\max \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\alpha^T Q \alpha$$

$$s.t. \quad 0 \le \alpha_i \le C, \quad \forall i \in [1, l]$$

$$y^T \alpha = 0$$

$$Q_{ij} = y_i y_j \Phi(x_i, x_j)$$

- Some kernel functions:

$\Phi(x_i, x_j; a, r, d) = (a x_i \cdot x_j + r)^d$ Polynomial

$\Phi(x_i, x_j; \gamma) = \exp\{-\gamma ||x_i - x_j||^2\}$ Radial Basis Function

Variables:

α: Weight for each training point (determines classifier)

Data:

l: number of training points

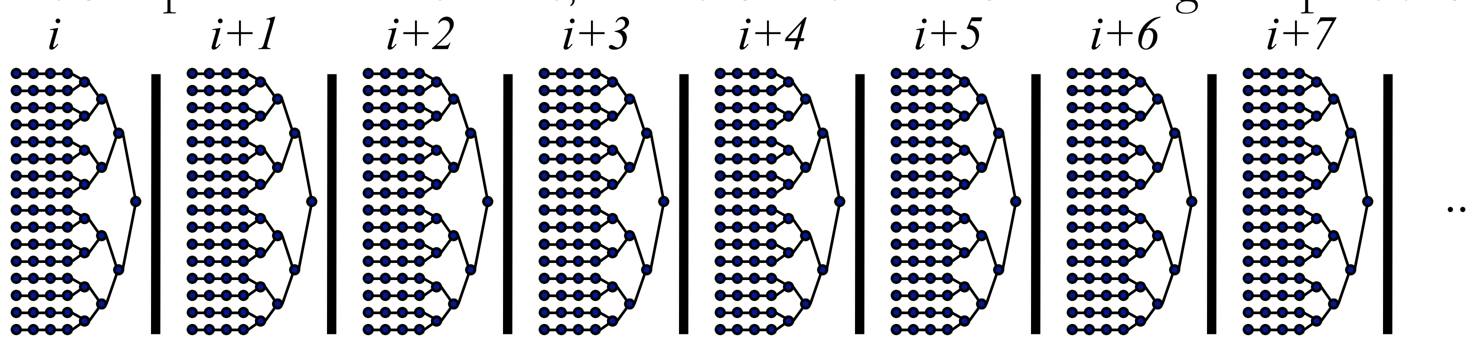
C: trades off error on training set for generalization performance

y: Label (+/- 1) for each training point

x: training points

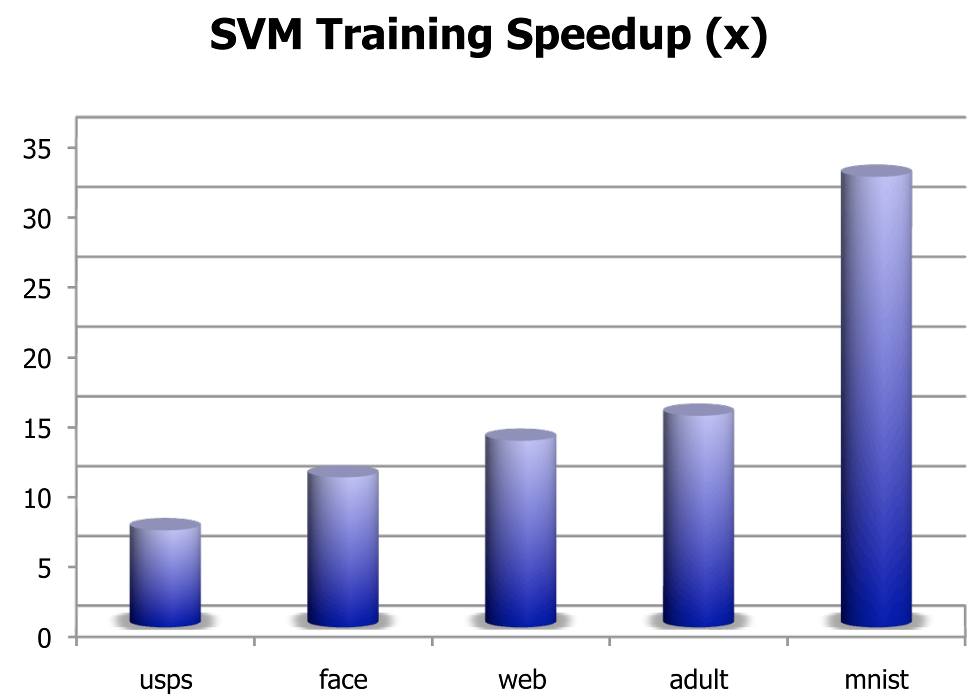
SVM Training: Implementation

- We use the Sequential Minimal Optimization algorithm
- The computation is iterative, with each iteration containing a Map Reduce



- The framework enables composition of the loop & the Map Reduce
  - Library based approaches have too much overhead
- At each iteration, we find the arg max and arg min of two data dependent subsets of a vector
  - Predication used for algorithmic purposes
- SVM Training requires computation of a large matrix
  - We cache rows of this matrix on the GPU, managing the cache on the CPU

SVM Training Results



- Comparing GeForce 8800GTX to LibSVM, on 2.66 GHz Intel Core 2 Duo
- 10-30x speedup
- This despite our currently naive algorithm compared to competitors
- Map Reduce Framework reduced kernel LOC by 34%

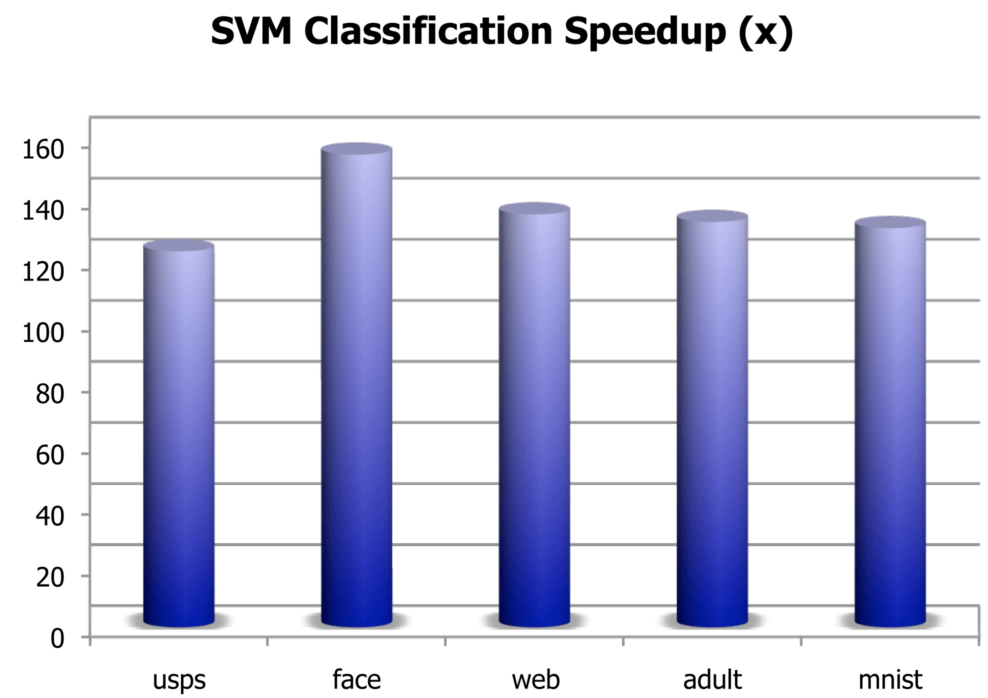
SVM Classification

- Training points with nonzero weights determine the classifier
  - "Support Vectors"
- Classify new point against support vectors:

$$\hat{z} = \operatorname{sgn}\left\{ b + \sum_{i=1}^{l} y_i \alpha_i \Phi(x_i, z) \right\}$$

- SVM Classification involves lots of dot products
- We cast the dot products as an SGEMM, and then use the Map Reduce framework to finish the classification

SVM Classification Results



- 120-150x speedup
  - Some of this is due to suboptimal implementation by LibSVM
- Map Reduce Framework reduced kernel LOC by 64%

Conclusion & Future Work

- The Map Reduce programming framework is a natural fit for GPUs
- Using the framework saves significant programmer effort
  - The most error prone sections of code subsumed in framework
  - Framework enables composition of Map Reduce computations
- Our SVM training and classification implementations perform well on the G80

- Future work
  - Prove applicability of framework with more applications
  - Add more reduction styles (including hybrid CPU/GPU reductions)

The end