Machine Learning in the Cloud

Yucheng Low
Joey Gonzalez
Aapo Kyrola
Danny Bickson
Carlos Guestrin
Joe Hellerstein
David O’Hallaron

Select Lab
Carnegie Mellon
In ML we face **BIG** problems

- 13 Million Wikipedia Pages
- 500 Million Facebook Users
- 3.6 Billion Flickr Photos
- 24 Hours a Minute YouTube
Exponential Parallelism

- Exponentially Increasing Parallel Performance
- Constant Sequential Performance

Processor Speed GHz vs. Release Date
The Challenges of Parallelism

Wide array of different parallel architectures:

- GPUs
- Multicore
- Clusters
- Mini Clouds
- Clouds

New algorithm design challenges:
- Race conditions and deadlocks
- Distributed state

New software implementations challenges:
- Parallel debugging and profiling
- Hardware specific APIs
Our Current Solution

Graduate students repeatedly solve the same parallel design challenges:
- Implement and debug complex parallel system
- Tune for a specific parallel platform
- A month later the conference paper contains:

  "We implemented ______ in parallel."

avoid these problems by using high-level abstractions
MapReduce – Map Phase

Embarrassingly Parallel independent computation
No Communication needed
MapReduce – Map Phase

Embarrassingly Parallel independent computation
No Communication needed
MapReduce – Map Phase

Embarrassingly Parallel independent computation
No Communication needed
MapReduce – Reduce Phase

Fold/Aggregation
MapReduce and ML

- Excellent for large data-parallel tasks!

Data-Parallel

Map Reduce
- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

Is there more to Machine Learning?
Iterative Algorithms?

We can implement iterative algorithms in MapReduce:
MapAbuse: Iterative MapReduce

- Only a subset of data needs computation:
MapAbuse: Iterative MapReduce

- System is not optimized for iteration:
Data-Parallel Algorithms can be Inefficient

Limitations of MapReduce can lead to inefficient parallel algorithms

But distributed Splash BP was built from scratch... efficient, parallel implementation was painful, painful, painful to achieve
What about structured Problems?

Example Problem: Will I be successful in research?

Success depends on the success of others.

May not be able to safely update neighboring nodes. [e.g., Gibbs Sampling]

Interdependent Computation: Not Map-Reducible
Parallel Computing and ML

- Not all algorithms are **efficiently** data parallel

Data-Parallel

Map Reduce

- Feature Extraction
- Cross Validation

Computing Sufficient Statistics
1) Sparse Data Dependencies
   - Sparse Primal SVM
   - Tensor/Matrix Factorization

2) Local Computations
   - Sampling
   - Belief Propagation

3) Iterative Updates
   - Expectation Maximization
   - Optimization
Gibbs Sampling

1) Sparse Data Dependencies

2) Local Computations

3) Iterative Updates
GraphLab is the **Solution**

- Designed specifically for ML needs
  - Express data dependencies
  - Iterative

- Simplifies the design of parallel programs:
  - Abstract away hardware issues
  - Automatic data synchronization
  - Addresses multiple hardware architectures
    - **Multicore**
    - **Distributed**
    - **Cloud computing**
    - **GPU** implementation in progress
A New Framework for Parallel Machine Learning

Select Lab
Carnegie Mellon
GraphLab

Data Graph
Update Function

Shared Data Table

GraphLab Model

Update Functions and Scopes

Scheduling
Part 1: Data Graph

A **Graph** with data associated with every vertex and edge

\[ x_3: \text{Sample value} \]
\[ C(X_3): \text{sample counts} \]

\[ \Phi(X_6, X_9): \text{Binary potential} \]
Update Functions: operations applied on vertex → transform data in scope of vertex

Gibbs Update:
- Read samples on adjacent vertices
- Read edge potentials
- Compute new sample for current vertex
Update Function Schedule
Part 2: Update Function Schedule
Need for Dynamic Scheduling

Converged

Slowly Converging
Focus Effort
Dynamic Schedule

CPU 1

CPU 2

a
h
b

a
h
i

Sense
Learn
Act
Dynamic Schedule

Update Functions can insert new tasks into schedule

- FIFO Queue
- Priority Queue
- Splash Schedule
- Wildfire BP [Selvatici et al.]
- Residual BP [Elidan et al.]
- Splash BP [Gonzalez et al.]

Obtain different algorithms simply by changing a flag!

```
--scheduler=fifo
```

```
--scheduler=priority
```

```
--scheduler=splash
```
Global Information

What if we need global information?

- Algorithm Parameters?
- Sufficient Statistics?
- Sum of all the vertices?
Part 3: Shared Data Table (SDT)

- Global constant parameters

**Constant:** Temperature

**Constant:** Total # Samples
Sync Operation

- **Sync** is a fold/reduce operation over the graph
- **Accumulate** performs an aggregation over vertices
- **Apply** makes a final modification to the accumulated data
- **Example:** Compute the average of all the vertices
Shared Data Table (SDT)

- Global constant parameters
- Global computation (Sync Operation)

### Constant:
- Temperature

### Sync:
- Loglikelihood
- Sample Statistics
- Total # Samples
Safety
and
Consistency
Write-Write Race
If adjacent update functions write simultaneously

Left update writes:  
Right update writes:  
Final Value
Race Conditions + Deadlocks

- Just one of the many possible races
- Race-free code is extremely difficult to write

GraphLab design ensures race-free operation
(through user-tunable consistency mechanism)
Part 4: Scope Rules

Guaranteed safety for all update functions
Full Consistency

Parallel update only allowed two vertices apart \( \Rightarrow \)
Reduced opportunities for parallelism
Obtaining More Parallelism

Not all update functions will modify the entire scope!

**Belief Propagation**: Only uses edge data
**Gibbs Sampling**: Only needs to read adjacent vertices
Edge Consistency
"Map" operations. Feature extraction on vertex data.
Vertex Consistency
GraphLab guarantees **sequential consistency**

∀ parallel execution, ∃ sequential execution of update functions which produce same result

---

**Thm: Sequential Consistency**
GraphLab

- Data Graph
- Update Function
- Shared Data Table
- Model
- Scheduling
- Update Functions and Scopes
Multicore Experiments
Multicore Experiments

- Shared Memory Implementation in C++ using Pthreads
- Tested on a 16 processor machine
  - 4x Quad Core AMD Opteron 8384
  - 64 GB RAM

- Belief Propagation + Parameter Learning
- Gibbs Sampling
- CoEM
- Lasso

- Compressed Sensing
- SVM
- PageRank
- Tensor Factorization
Graphical Model Learning

3D retinal image denoising

Data Graph: 256x64x64 (1M) vertices

Update Function
Belief Propagation

Sync
Acc: Compute inference statistics
Apply: Take a gradient step
Graphical Model Learning

Standard parameter learning takes gradient only after inference is complete.

With GraphLab:
Take gradient step while inference is running.

Runtime

2100 sec

700 sec

3x faster!

Iterated Simultaneous
Lasso

\[
\arg \min_w \|Xw - y\|_2^2 + \lambda \|w\|_1
\]

Data matrix, \( n \times d \)

weights \( d \times 1 \)

Observations \( n \times 1 \)

\[X = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 \\
6 & 7 \\
8 & 9 & 10
\end{bmatrix}\]

5 Features

4 Examples

Shooting Algorithm [Coordinate Descent]
- Updates on weight vertices modify losses on observation vertices.

Financial prediction dataset from Kogan et al [2009].

Requires the Full Consistency Model
Full Consistency

![Graph showing speedup vs. number of CPUs for different data types: Dense, Sparse, and Optimal. The graph indicates that the speedup increases as the number of CPUs increases, with Optimal achieving the highest speedup.]
Relaxing Consistency

Why does this work? (We may have an answer soon.)
Named Entity Recognition Task

Is "Dog" an animal?
Is "Catalina" a place?

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.2M</td>
</tr>
<tr>
<td>Large</td>
<td>2M</td>
</tr>
</tbody>
</table>

- Hadoop
  - 95 Cores
  - 7.5 hrs

Australia travelled to <X>
Catalina Island <X> is pleasant
### CoEM (Rosie Jones, 2005)

<table>
<thead>
<tr>
<th></th>
<th>Number of CPUs</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95 Cores</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>GraphLab</td>
<td>16 Cores</td>
<td>30 min</td>
</tr>
</tbody>
</table>

**Highlights:**
- **6x fewer CPUs!**
- **15x Faster!**
GraphLab in the Cloud
Moving towards the cloud...

- Purchasing and maintaining computers is very expensive
  - Most computing resources seldomly used
    - Only for deadlines...

- Buy time, access hundreds or thousands of processors
  - Only pay for needed resources
Addressing cloud computing challenges

- **Challenge:** Distributed memory
  - **GraphLab solution:** Optimized data partition

- **Challenge:** Limited bandwidth
  - **GraphLab solution:** Smart caching, interleave computation/comm

- **Challenge:** High latency
  - **GraphLab solution:** Push data, latency hiding mechanism
GraphLab in the Cloud Experiments

- Highly optimized implementation
  - Computer clusters
  - Amazon EC2

- **GraphLab automatically configures and distributes through EC2**
  - Very easy to use

- Thoroughly evaluated in three case studies:
  - coEM
  - probabilistic tensor factorization
  - video co-segmentation
    - inference & learning in a huge graphical model
## Experiment Setup

Tested on both **Regular** and **HPC** nodes, using up to **32 machines**

<table>
<thead>
<tr>
<th>Regular Nodes</th>
<th>8 Cores per node</th>
<th>Up to 128 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC Node</td>
<td>16 Cores per node</td>
<td>Up to 256 Cores</td>
</tr>
</tbody>
</table>
CoEM (Rosie Jones, 2005)

<table>
<thead>
<tr>
<th>Method</th>
<th>Cores</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>GraphLab</td>
<td>16</td>
<td>30 min</td>
</tr>
<tr>
<td><strong>GraphLab in the Cloud</strong></td>
<td><strong>32 EC2 machines</strong></td>
<td><strong>80 secs</strong></td>
</tr>
</tbody>
</table>

0.3% of Hadoop time
Video Cosegmentation

A lot of data! Complex solutions infeasible!
Naïve Idea:
Treat patches independently

Use Gaussian EM clustering (on image features)

**E step:** Predict membership of each patch given cluster centers

**M step:** Compute cluster centers given memberships of each patch

Does not take relationships among patches into account!
Better Idea:
Connect the patches using an **MRF**. Set edge potentials so that adjacent (spatially and temporally) patches prefer to be of the same cluster.

Gaussian EM clustering with a twist:

**E step:** Make unary potentials for each patch using cluster centers.
Predict membership of each patch using BP

**M step:** Compute cluster centers given memberships of each patch

Discover “coherent” segment types across a video (extends Batra et al. ‘10)

1. Form super-voxels video
2. EM & inference in Markov random field

Huge model: 23 million nodes, 390 million edges
Cost-Time Tradeoff

video co-segmentation results

- A few machines helps a lot
- Diminishing returns

More machines, higher cost

Running time

Cost ($)
Bayesian Tensor Factorization

**Vertices** store **user-factors** and **movie-factors**

**Edges** store user<->movie **ratings**

**Time-factors** are stored as Shared Data
Bayesian Tensor Factorization

Netflix dataset:
480K users, 18K movies, 27 time periods, 100M ratings

<table>
<thead>
<tr>
<th>Method</th>
<th>Cores</th>
<th>Time per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiong et al (2010)</td>
<td>1 Core</td>
<td>2160s</td>
</tr>
<tr>
<td>Distributed GraphLab (HPC)</td>
<td>256 Cores</td>
<td>6.8s</td>
</tr>
</tbody>
</table>
Parallel GraphLab 1.1
Multicore Available Today
GraphLab in the Cloud soon...

Documentation... Code... Tutorials...

http://graphlab.ml.cmu.edu
GraphLab Release 1.1

- Parallel **PThread** based implementation
- **Matlab™ 2010b** interface using **EMLC**
- **Java**, **Jython** interface through **JNI**
- Tutorials and Demonstration Code
C++, Java and Python

Native C++ interface
- highest Performance
- access to complete GraphLab feature set
- template heavy code can be difficult for novice C++ programmers

Pure Java API for GraphLab
- use plain Java objects for the whole graph
- write update function in Java
- full Graphlab C++ performance through Java Native Interface (JNI)

Python API for GraphLab (via Jython)
- vertex and edge data can be any Python type
- Python allows very concise update functions
- works on top of Java interface for GraphLab
Matlab Interface

- Update Functions are written in a Matlab subset (embedded Matlab)
- Update Functions compiled to **native C++ code** which do not depend on Matlab
- **Generated MEX interface** allow resultant GraphLab program to interface with Matlab easily
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml

    vdata = get_vertex_data(handle, currentvertex);
    vdata.belief = vdata.unary;
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        vdata.belief = vdata.belief .* inedata.msg;
        vdata.belief = vdata.belief / sum(vdata.belief);
    end
    set_vertex_data(handle, currentvertex, vdata);
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        outedata = get_edge_data(handle, outedges(i));
        % get the out going message
        oldoutedatamsg = outedata.msg;
        outedata.msg = vdata.belief ./ inedata.msg;
        outedata.msg = outedata.msg / sum(outedata.msg);
        % multiply by the edge factor and normalize
        outedata.msg = outedata.msg * outedata.binary;
        outedata.msg = outedata.msg / sum(outedata.msg);
        set_edge_data(handle, outedges(i), Outedata);
        % compute the residual;
        residual = sum(abs(outedata.msg - oldoutedatamsg));
        if (residual > 1E-5)
            add_task(handle, outv(i), 'bp_update', residual);
        end
    end
end
GraphLab

- Parallel abstraction tailored to Machine Learning

- Parallel framework compactly expresses
  - Data/computational dependencies
  - Iterative computation

- Achieves state-of-the-art parallel performance on variety of problems

- Easy to use
  - E.g., data partition, optimized communication, automatically configures & distributes over EC2,...
Future Work

- Distributed GraphLab
  - Robustness

- GPU GraphLab
  - Memory bus bottle neck
  - Warp alignment

- State-of-the-art performance for
  <Your Algorithm Here>.
Parallel GraphLab 1.1
Multicore Available Today
GraphLab in the Cloud soon...

Documentation... Code... Tutorials...

http://graphlab.ml.cmu.edu
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
    vdata = get_vertex_data(handle, currentvertex);
    vdata.belief = vdata.unary;
    for i = 1:length(inedges)
        indata = get_edge_data(handle, inedges(i));
        vdata.belief = vdata.belief .* indata.msg;
        vdata.belief = vdata.belief / sum(vdata.belief);
    end
    set_vertex_data(handle, currentvertex, vdata);
    for i = 1:length(inedges)
        indata = get_edge_data(handle, inedges(i));
        outedata = get_edge_data(handle, outedges(i));
        % get the outgoing message
        oldoutedatamsg = outedata.msg;
        outedata.msg = vdata.belief ./ indata.msg;
        outedata.msg = outedata.msg / sum(outedata.msg);
        % multiply by the edge factor and normalize
        outedata.msg = outedata.msg * outedata.binary;
        outedata.msg = outedata.msg / sum(outedata.msg);
        set_edge_data(handle, outedges(i), outedata);
        % compute the residual
        residual = sum(abs(outedata.msg - oldoutedatamsg));
        if (residual > 1E-5)
            add_task(handle, outv(i), 'bp_update', residual);
        end
    end
end
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
    vdata = get_vertex_data(handle, currentvertex);
    vdata.belief = vdata.unary;
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        vdata.belief = vdata.belief .* inedata.msg;
        vdata.belief = vdata.belief / sum(vdata.belief);
    end
    set_vertex_data(handle, currentvertex, vdata);
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        outedata = get_edge_data(handle, outedges(i));
        % get the out going message
        oldoutedatams = outedata.msg;
        outedata.msg = vdata.belief ./ inedata.msg;
        outedata.msg = outedata.msg / sum(outedata.msg);
        % multiply by the edge factor and normalize
        outedata.msg = outedata.msg * outedata.binary;
        outedata.msg = outedata.msg / sum(outedata.msg);
        set_edge_data(handle, outedges(i), outedata);
        % compute the residual;
        residual = sum(abs(outedata.msg - oldoutedatams));
        if (residual > 1E-5)
            add_task(handle, outv(i), 'bp_update', residual);
        end
    end
end

Gets the edge data on a particular in-edge
Matlab Interface

function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml

vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end

set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    outedata = get_edge_data(handle, outedges(i));
    % get the outgoing message
    oldoutedatamsg = outedata.msg;
    outedata.msg = vdata.belief ./ inedata.msg;
    outedata.msg = outedata.msg / sum(outedata.msg);

    % multiply by the edge factor and normalize
    outedata.msg = outedata.msg .* outedata.binary;
    outedata.msg = outedata.msg / sum(outedata.msg);
    set_edge_data(handle, outedges(i), outedata);
    % compute the residual
    residual = sum(abs(outedata.msg - oldoutedatamsg));
    if (residual > 1E-5)
        add_task(handle, outv(i), 'bp_update', residual);
    end
end
end

Computes the new belief by multiplying incoming messages with the unary potential.
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end
set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
inedata = get_edge_data(handle, inedges(i));
outedata = get_edge_data(handle, outedges(i));
% get the outgoing message
oldoutedatamsg = outedata.msg;
outedata.msg = vdata.belief ./ inedata.msg;
outedata.msg = outedata.msg / sum(outedata.msg);

% multiply by the edge factor and normalize
outedata.msg = outedata.msg * outedata.binary;
outedata.msg = outedata.msg / sum(outedata.msg);
set_edge_data(handle, outedges(i), outedata);
% compute the residual
residual = sum(abs(outedata.msg - oldoutedatamsg));
if (residual > 1E-5)
    add_task(handle, outv(i), 'bp_update', residual);
end
end
end

Updates the current vertex data
For each in/out edge pair

```matlab
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end
set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    outedata = get_edge_data(handle, outedges(i));

    % get the out going message
    oldoutdatamsg = outedata.msg;
    outedata.msg = vdata.belief ./ inedata.msg;
    outedata.msg = outedata.msg / sum(outedata.msg);

    % multiply by the edge factor and normalize
    outedata.msg = outedata.msg * outedata.binary;
    outedata.msg = outedata.msg / sum(outedata.msg);
    set_edge_data(handle, outedges(i), outedata);

    % compute the residual
    residual = sum(abs(outedata.msg - oldoutdatamsg));
    if (residual > 1E-5)
        add_task(handle, outv(i), 'bp_update', residual);
    end
end
end
```
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
    vdata = get_vertex_data(handle, currentvertex);
    vdata.belief = vdata.unary;
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        vdata.belief = vdata.belief .* inedata.msg;
        vdata.belief = vdata.belief / sum(vdata.belief);
    end
    set_vertex_data(handle, currentvertex, vdata);
    for i = 1:length(inedges)
        inedata = get_edge_data(handle, inedges(i));
        outedata = get_edge_data(handle, outedges(i));
        % get the outgoing message
        oldout datapmsg = outedata.msg;
        outedata.msg = vdata.belief ./ inedata.msg;
        outedata.msg = outedata.msg / sum(outedata.msg);
        % multiply by the edge factor and normalize
        outedata.msg = outedata.msg * outedata.binary;
        outedata.msg = outedata.msg / sum(outedata.msg);
        set_edge_data(handle, outedges(i), outedata);
        % compute the residual
        residual = sum(abs(outedata.msg - oldout datapmsg));
        if (residual > 1E-5)
            add_task(handle, outv(i), 'bp_update', residual);
        end
    end
end
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end
set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    outedata = get_edge_data(handle, outedges(i));
    % get the outgoing message
    oldoutedatamsg = outedata.msg;
    outedata.msg = vdata.belief ./ inedata.msg;
    outedata.msg = outedata.msg / sum(outedata.msg);

    % multiply by the edge factor and normalize
    outedata.msg = outedata.msg * outedata.binary;
    outedata.msg = outedata.msg / sum(outedata.msg);

    set_edge_data(handle, outedges(i), outedata);
    % compute the residual;
    residual = sum(abs(outedata.msg - oldoutedatamsg));
    if (residual > 1E-5)
        add_task(handle, outv(i), 'bp_update', residual);
    end
end
end

Sets the new outgoing edge data
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end
set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    outedata = get_edge_data(handle, outedges(i));
    % get the outgoing message
    oldoutedatamsg = outedata.msg;
    outedata.msg = vdata.belief ./ inedata.msg;
    outedata.msg = outedata.msg / sum(outedata.msg);

    % multiply by the edge factor and normalize
    outedata.msg = outedata.msg * outedata.binary;
    outedata.msg = outedata.msg / sum(outedata.msg);
    set_edge_data(handle, outedges(i), outdata);
    % compute the residual:
    residual = sum(abs(outdata.msg - oldoutedatamsg));
    if (residual > 1E-5)
        add_task(handle, outv(i), 'bp_update', residual);
    end
end
end

Computes the message residual
function bp_update(currentvertex, inedges, inv, outedges, outv, handle) %#eml
vdata = get_vertex_data(handle, currentvertex);

vdata.belief = vdata.unary;
for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    vdata.belief = vdata.belief .* inedata.msg;
    vdata.belief = vdata.belief / sum(vdata.belief);
end
set_vertex_data(handle, currentvertex, vdata);

for i = 1:length(inedges)
    inedata = get_edge_data(handle, inedges(i));
    outedata = get_edge_data(handle, outedges(i));
    % get the out going message
    oldoutedatamsg = outedata.msg;
    outedata.msg = vdata.belief ./ inedata.msg;
    outedata.msg = outedata.msg / sum(outedata.msg);
    % multiply by the edge factor and normalize
    outedata.msg = outedata.msg * outedata.binary;
    outedata.msg = outedata.msg / sum(outedata.msg);
    set_edge_data(handle,outedges(i),outedata);
    % compute the residual
    residual = sum(abs(outedata.msg - oldoutedatamsg));
    if (residual > 1E-5)
        add_task(handle, outv(i), 'bp_update', residual);
    end
end
end

If residual is large, schedule destination vertex
Matlab Interface

```
compile_update_function({'bpupdate'},
    vdata_example, edata_example, ...)

[newvdata, neweddata, newadj] =
    bp(vertexdata, edgedata, 
        adjacency_matrix, 
        scheduling_spec);
```

MRF Specification
Matlab Interface

bpupdate.m

compile_update_function({'bpupdate'},
            vdata_example, edata_example, ...)

[newvdata, newedata, newadj] =
        bp(vertexdata, edgedata, adjacency_matrix, scheduling_spec);

Resultant Graph after running BP
Matlab to GraphLab Compiler Details

Graph datatype examples

vdata.belief = [1,1]
vdata.unary = [2,2]

Update functions

bpupdate.m

vdata.belief = [double(0)];
eml.varsize('vdata.belief',
[1 Inf]);
vdata.unary = [double(0)];
eml.varsize('vdata.unary',
[1 Inf]);

Type check restrictions and generate EML type descriptors

Matlab to C generation with EMLC

Parse output C code and generate
1. converters between emxArray and mxArray.
2. emxArray serialization and deserialization.

Generate binary, mex and m frontend code.

Makefile generation

Extensive use of C++ templates and preprocessor to identify graph datatypes and to wrap the update functions.
Gibbs Sampling

- Two methods for sequentially consistency:

**Scopes**
- Edge Scope

**Scheduling**
- Graph Coloring

```
graphlab(gibbs, edge, sweep);
graphlab(gibbs, vertex, colored);
```
Gibbs Sampling

- **Protein-protein interaction networks** [Elidan et al. 2006]
  - Pair-wise MRF
  - 14K Vertices
  - 100K Edges

- **10x Speedup**
- Scheduling reduces locking overhead

![Graph showing speedup vs. number of CPUs]

- Optimal
- Colored Schedule
- Round robin schedule
Lasso

L1 regularized Linear Regression

\[ Y \approx X\beta \]

**Shooting Algorithm** (Coordinate Descent)
Due to the properties of the update, full consistency is needed

\[
X = \begin{bmatrix}
1 & 2 & 1 \\
1 & 2 & 1 \\
\end{bmatrix}
\]
L1 regularized Linear Regression

\[ Y \approx X \beta \]

**Shooting Algorithm** (Coordinate Descent)
Due to the properties of the update, full consistency is needed.

\[ X = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 \end{bmatrix} \]
Lasso

L1 regularized Linear Regression

\[ Y \approx X \beta \]

**Shooting Algorithm** (Coordinate Descent)
Due to the properties of the update, full consistency is needed

\[ X = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 2 \end{bmatrix} \]

Finance Dataset from Kogan et al [2009].
Full Consistency

- **Optimal**
- **Sparse**
- **Dense**

Graph showing speedup vs. number of CPUs for different types of consistency.
Why does this work? (Open Question)
Did you compare against ___ Lasso

- We had trouble getting the standard Lasso implementations to run on the dataset
- But we will love to try it out
Comparing against Hadoop

- Hadoop is the current available implementation of Tom Mitchell’s group we are assisting.
- Now this is what they are using.
- Demonstrate that Hadoop (though popular) is not necessarily the best framework for your problem.
DAG Abstraction

- Computation represented as a Directed Acyclic Graph.
- Vertices represent “programs”. A program starts when data is received on all incoming edges, and outputs data on its outgoing edges.
Clock Systolic Array

- Processors are arranged in a directed graph.
- All processors compute, then transmit outgoing messages in sync.
Clocked Systolic Array

- Processors are arranged in a directed graph.
- All processors compute, then transmit outgoing messages in sync.
Read-Write Race

Update Function reads from in-edges and writes to out-edges
Update Function reads from in-edges and writes to out-edges

Read-Write Race
if write while an adjacent update function is reading
Compressed Sensing

Represent image as sparse linear combination of basis functions. Minimize reconstruction error.

**Interior Point method with Gaussian BP as linear solver**

50% random wavelet projection
Graphical Model Learning

3D retinal image denoising

Data Graph: 256x64x64

Update Function: Belief Propagation

Shared Data: Edge Potentials. Gradient step computed with Sync

Speedup

Better

Number of CPUs

Runtime

Inference

Gradient Step

Sync Frequency (Seconds)

Splash Schedule

Optimal

Approx. Priority Schedule

Better