Splash Belief Propagation: 
Efficient Parallelization Through Asynchronous Scheduling

Joseph Gonzalez, Yucheng Low, 
Carlos Guestrin, David O’Hallaron

Computers which cooperated closely on this project:

BigBro1, BigBro2, BigBro3, BigBro4, BigBro5, BigBro6, BiggerBro, BigBroFS 
Tashish01, Tashi02, Tashi03, Tashi04, Tashi05, Tashi06, …, Tashi30, 
parallel, gs6167, koobcam (helped with writing)
Natural Parallelism

- Synchronous parallel updates:

- Run until parameters stop changing
Popularity of Synchronous Parallelism

- Synchronous Belief Propagation
- Iterative Linear Solvers
- EM Methods (Co-EM)
- Alternating Minimization
- Synchronous “Gibbs Sampling”
- Affinity Propagation
- Random Walk Simulation for PageRank
- Map-Reduce for Machine Learning on Multicore
Explosion in “Map-Reduce” Parallelism:

- Natural Parallelism

- Set of Parameters

- Iteration

- Useful Work

- Changed

- Wasted Work

- Iteration

- Changed

- Iteration

- Changed

- Wasted Work
Objectives of Parallel Algorithms

- Parallelism
  - Expose independent computation

- Algorithmic Efficiency
  - Eliminate wasted computation

- Implementation Efficiency
  - Map computation to real hardware
Recipe for Efficient Adaptive Parallelism

Distributed Parallel Splash

A Case Study in Efficient Adaptive Parallelism

- Identify sequential dependencies (approximations?)
  - Chain Graphical Models
- Address sequential dependencies explicitly
  - Splash Operation
- Adaptively prioritize computation
  - Belief Residual Scheduling
- Efficiently map computation to hardware
  - Over-partitioning
Graphical Models and Parallelism

Graphical models provide a common language for general purpose parallel algorithms in machine learning.

A parallel inference algorithm would improve:

- Protein Structure Prediction
- Movie Recommendation
- Computer Vision

Inference is a key step in Learning Graphical Models
Graphical Model

- Represent the relationships between random variables

A: Alice
B: Bob

Pr(Cancer(B) = True | Smokes(A) = True & Friends(A,B) = True) = ?

Inference

Pr(Cancer(B) = True | Smokes(A) = True & Friends(A,B) = True) = ?
Belief Propagation (BP)

- Iterative message passing algorithm

Naturally Parallel Algorithm
Parallel Synchronous BP

- Given the old messages all new messages can be computed in parallel:

![Diagram of parallel synchronous BP with CPUs and message flow](image-url)
Given the old messages all new messages can be computed in parallel:
Sequential Computational Structure
Hidden Sequential Structure
Hidden Sequential Structure

Running Time:

\[
\frac{2n \text{ Messages Calculations}}{p \text{ Processors}} \times (n \text{ Iterations to Converge}) = \frac{2n^2}{p}
\]

- Time for a single parallel iteration
- Number of Iterations
Optimal Sequential Algorithm

<table>
<thead>
<tr>
<th>Naturally Parallel</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2n^2/p$</td>
</tr>
<tr>
<td></td>
<td>$p \leq 2n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forward-Backward</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2n$</td>
</tr>
<tr>
<td></td>
<td>$p = 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimal Parallel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n$</td>
</tr>
<tr>
<td></td>
<td>$p = 2$</td>
</tr>
</tbody>
</table>
Key Computational Structure

Naturally Parallel

\[ 2n^2/p \]
\[ p \leq 2n \]

Running Time

Inherent **Sequential** Structure

Requires Efficient Scheduling

Optimal Parallel

\[ n \]
\[ p = 2 \]
Parallelism by Approximation

True Messages

$\tau_\varepsilon$-Approximation

$\tau_\varepsilon$ represents the minimal sequential structure

$$\left| m_{9\to10} - m'_{9\to10} \right|_1 \leq \varepsilon$$
Theorem:
Using $p$ processors this algorithm achieves a $\tau_\varepsilon$ approximation in time:

$$O \left( \frac{n}{p} + \tau_\varepsilon \right)$$
The Splash Operation

Generalize the optimal chain algorithm:

1) Grow a BFS Spanning tree with fixed size
2) Forward Pass computing all messages at each vertex
3) Backward Pass computing all messages at each vertex
Running Parallel Splashes

Key Challenges:
1) How do we schedules Splashes?
2) How do we partition the Graph?

- Partition the graph
- Schedule Splashes locally
- Transmit the messages along the boundary of the partition
Belief Residual Scheduling

Assign priorities based on the cumulative change in belief:

\[ r_v = \sum_{i=1}^{n} \left( \Delta \text{Message}_i - \text{Change}_i \right) + \sum_{i=1}^{m} \left( \Delta \text{Message}_i - \text{Change}_i \right) \]

A vertex whose belief has changed substantially since last being updated will likely produce informative new messages.
Where do we Splash?

- Assign priorities and use a scheduling queue to select roots:
Belief residuals can be used to **dynamically** reshape and resize Splashes:
Using **Splash Pruning** our algorithm is able to dynamically select the **optimal** splash size.
Example

Synthetic Noisy Image

Vertex Updates

Algorithm identifies and focuses on hidden sequential structure
Parallel Splash Algorithm

Fast Reliable Network

Theorem:

Given a uniform partitioning of the chain graphical model, Parallel Splash will run in time:

\[ O \left( \frac{n}{p} + T\epsilon \right) \]

retaining optimality.
Partitioning Objective

The partitioning of the factor graph determines:
- Storage, Computation, and Communication

Goal:
- Balance **Computation** and Minimize **Communication**

Ensure Balance

Comm. cost

CPU 1

CPU 2
Unknown Costs

- Determined by belief scheduling
- Depends on: graph structure, factors, ...
- Uninformed cuts could be highly imbalanced
Over-Partitioning

- Over-cut graph into $k*p$ partitions and randomly assign CPUs
  - Increase balance
  - Increase communication cost (More Boundary)

Without Over-Partitioning

k=6
Parallel Splash Algorithm

- Over-Partition factor graph
  - Randomly assign pieces to processors
- Schedule Splashes locally using belief residuals
- Transmit messages on boundary
Experiments

- Implemented in C++ using:
  - Pthreads in Shared Memory
  - MPI in Distributed Setting

- Systems used:
  - Shared Memory Setting: AMD Shanghai 4 x Quad Core
  - Distributed Setting: 15 Nodes with 2 x Quad Core Intel Xeon Processors with Gigabit Ethernet Switch

- Tested on Markov Logic Networks obtained from Alchemy [Domingos et al. SSPR 08]
Largest model where Synchronous BP Converges:

- 1K Binary Variables and 20K Factors
Largest model where Synchronous BP Converges:
- 1K Binary Variables and 20K Factors
Large model in which Synchronous BP does not converge.
Distributed: Large Graph

- UW-Systems
  - 8K Variables
  - 406K Factors
- Single Processor
  - Running Time:
    - 1 Hour
- Linear to Super-Linear up to 120 CPUs
  - Cache efficiency

Graph showing speedup vs. number of CPUs, with a linear line indicating better performance up to 120 CPUs.
Experimental results on large factor graphs:

- **Linear** to **super-linear** speed-up using up to 120 processors
Questions?

Joseph Gonzalez
jegonzal@cs.cmu.edu
Uniformed Cuts

Uninformed Cut

Greater imbalance & lower communication cost

Uninformed Cut

Update Counts

Optimal Cut

Work Imbalance

Communication Cost

Denoise  UW-Syst.

Denoise  UW-Syst.

Better

Better

Uninformed
Optimal
Over-Partitioning Results

- Provides a simple method to trade between work balance and communication cost.

**Work Imbalance**

- As the partition factor $k$ increases, the work imbalance decreases.

**Communication Cost**

- As the partition factor $k$ increases, the communication cost increases.

Better
Parallel Performance (Small Graph)

- UW-Languages
  - 1K Variables
  - 27K Factors
- Single Processor
  Running Time:
  - 1.5 Minutes
- Linear to Super-Linear up to 30 CPUs
  - Network costs quickly dominate short running-time
Ongoing Work

Generalized Framework for Asynchronous Parallel Machine Learning
Can we generalize this idea?

**Algorithm Properties:**

- Lots of Data
- Sparse Dependencies
- Iterative
- Asynchronous Prioritization

---

**MapReduce**
- Not Iterative
- Not Asynchronous

**MPI/Pthreads**
- Too low level
Functional Asynchronous Parallelism

Machine Learning Expert

- Data
- Graph
  - Data Dependencies
- Update Function
- Scheduler

GraphLab

- Many-core
- Small Cluster
- Data Center
- GPU?

Systems Expert